Paradigmas Fundamentales de Programación

Flujos y programación con flujos

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

Plan

Plan

Universidad del Valle

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

- Creando un flujo:

declare Xs Xs2 in
Xs=0|1|2|3|4|Xs2

- Incrementando un flujo:

declare Xs Xs2 in
Xs=0|1|2|3|4|Xs2

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

- Creando un flujo:

```
declare Xs Xs2 in
Xs=0|1|2|3|4|Xs2
```

- Incrementando un flujo:

```
declare Xs3 in
Xs2=5|6|7|Xs3
```

- El productor crea flujos, los consumidores leen el flujo.

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

- Creando un flujo:

```
declare Xs Xs2 in
Xs=0|1|2|3|4|Xs2
```

- Incrementando un flujo:

```
declare Xs3 in
Xs2=5|6|7|Xs3
```

- El productor crea flujos; los consumidores leen el flujo.

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

- Creando un flujo:

```
declare Xs Xs2 in
Xs=0|1|2|3|4|Xs2
```

- Incrementando un flujo:

```
declare Xs3 in
Xs2=5|6|7|Xs3
```

- El productor crea flujos; los consumidores leen el flujo.

La comunicación por flujos (1)

Flujos

- La técnica más útil para programación concurrente: utilización de flujos para la comunicación entre hilos.
- Un flujo es una lista de mensajes potencialmente ilimitada, i.e., es una lista cuya cola es una variable de flujo de datos no-ligada.
- Envío de mensaje: extender el flujo con un elemento (ligar la cola a una pareja que contiene el mensaje y una variable nueva no-ligada).
- Recepción de mensaje: leer un elemento del flujo.
- Objeto flujo: hilo que se comunica a través de flujos.

Flujos en acción

- Creando un flujo:

```
declare Xs Xs2 in
Xs=0|1|2|3|4|Xs2
```

- Incrementando un flujo:

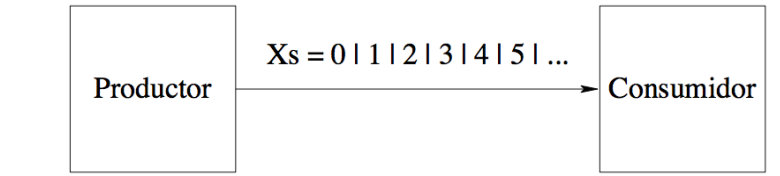
```
declare Xs3 in
Xs2=5|6|7|Xs3
```

- El productor crea flujos; los consumidores leen el flujo.

Plan

La comunicación por flujos (2)

Ejemplo: suma de enteros



Productor

```
fun {Generar N Limite}
   if N<Limite then
      N|{Generar N+1
                 Limite}
   else nil end
end
```

Consumidor

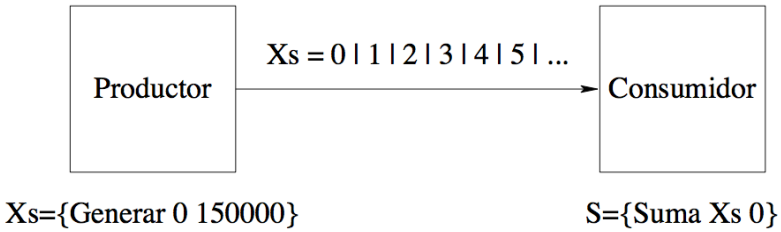
```
fun {Suma Xs A}
   case Xs
   of X|Xr then
      {Suma Xr A+X}
   [] nil then A
   end
end
```

Uso

```
local Xs S in
   thread Xs={Generar 0 150}
   end
   thread S={Suma Xs 0} end
   {Browse S}
end
```

La comunicación por flujos (2)

Ejemplo: suma de enteros



Productor

```
fun {Generar N Limite}
   if N<Limite then
      N|{Generar N+1
                 Limite}
   else nil end
end
```

Consumidor

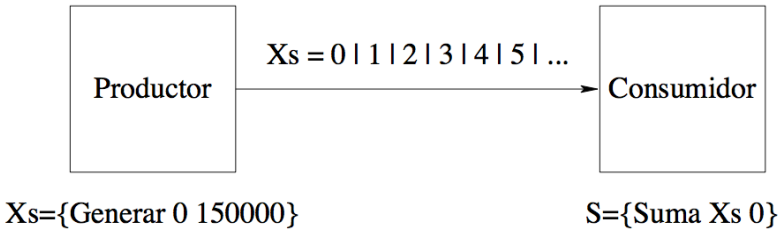
```
fun {Suma Xs A}
   case Xs
   of X|Xr then
      {Suma Xr A+X}
   [] nil then A
   end
end
```

Uso

```
local Xs S in
   thread Xs={Generar 0 150}
   end
   thread S={Suma Xs 0} end
   {Browse S}
end
```

La comunicación por flujos (2)

Ejemplo: suma de enteros



Productor

```
fun {Generar N Limite}
   if N<Limite then
      N|{Generar N+1
                 Limite}
   else nil end
end
```

Consumidor

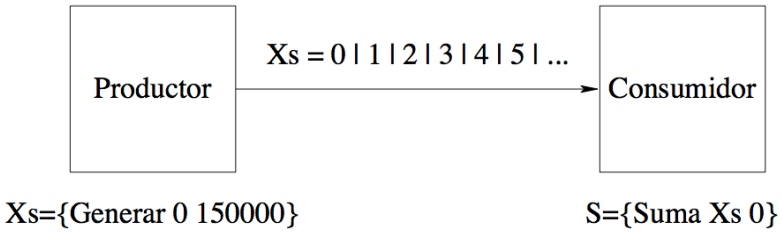
```
fun {Suma Xs A}
   case Xs
   of X|Xr then
      {Suma Xr A+X}
   [] nil then A
   end
end
```

Uso

```
local Xs S in
  thread Xs={Generar 0 150}
  end
  thread S={Suma Xs 0} end
  {Browse S}
end
```

La comunicación por flujos (2)

Ejemplo: suma de enteros



Productor

```
fun {Generar N Limite}
   if N<Limite then
      N|{Generar N+1
                 Limite}
   else nil end
end
```

Consumidor

```
fun {Suma Xs A}
   case Xs
   of X|Xr then
      {Suma Xr A+X}
   [] nil then A
   end
end
```

Uso

```
local Xs S in
  thread Xs={Generar 0 150}
  end
  thread S={Suma Xs 0} end
  {Browse S}
end
```

Plan

La comunicación por flujos (3)

Múltiples lectores

```
local Xs S1 S2 S3 in
   thread Xs={Generar 0 150000} end
   thread S1={Suma Xs 0} end
   thread S2={Suma Xs 0} end
   thread S3={Suma Xs 0} end
end
```

- Los consumidores no interfieren con los otros porque ellos realmente no “consumen” el flujo; sólo lo leen.

La comunicación por flujos (3)

Múltiples lectores

```
local Xs S1 S2 S3 in
   thread Xs={Generar 0 150000} end
   thread S1={Suma Xs 0} end
   thread S2={Suma Xs 0} end
   thread S3={Suma Xs 0} end
end
```

- Los consumidores no interfieren con los otros porque ellos realmente no “consumen” el flujo; sólo lo leen.

La comunicación por flujos (3)

Múltiples lectores

```
local Xs S1 S2 S3 in
   thread Xs={Generar 0 150000} end
   thread S1={Suma Xs 0} end
   thread S2={Suma Xs 0} end
   thread S3={Suma Xs 0} end
end
```

- Los consumidores no interfieren con los otros porque ellos realmente no “consumen” el flujo; sólo lo leen.

Transductores y canales

- Transductor: objeto flujo que lee el flujo del consumidor y crea otro flujo que es leido por un consumidor/transductor.
- Canal: secuencia de objetos flujo, cada uno de los cuales alimenta al siguiente.
- Filtro: transductor sencillo.

La comunicación por flujos (3)

Múltiples lectores

```
local Xs S1 S2 S3 in
   thread Xs={Generar 0 150000} end
   thread S1={Suma Xs 0} end
   thread S2={Suma Xs 0} end
   thread S3={Suma Xs 0} end
end
```

- Los consumidores no interfieren con los otros porque ellos realmente no “consumen” el flujo; sólo lo leen.

Transductores y canales

- Transductor: objeto flujo que lee el flujo del consumidor y crea otro flujo que es leido por un consumidor/transductor.
- Canal: secuencia de objetos flujo, cada uno de los cuales alimenta al siguiente.
- Filtro: transductor sencillo.

La comunicación por flujos (3)

Múltiples lectores

```
local Xs S1 S2 S3 in
   thread Xs={Generar 0 150000} end
   thread S1={Suma Xs 0} end
   thread S2={Suma Xs 0} end
   thread S3={Suma Xs 0} end
end
```

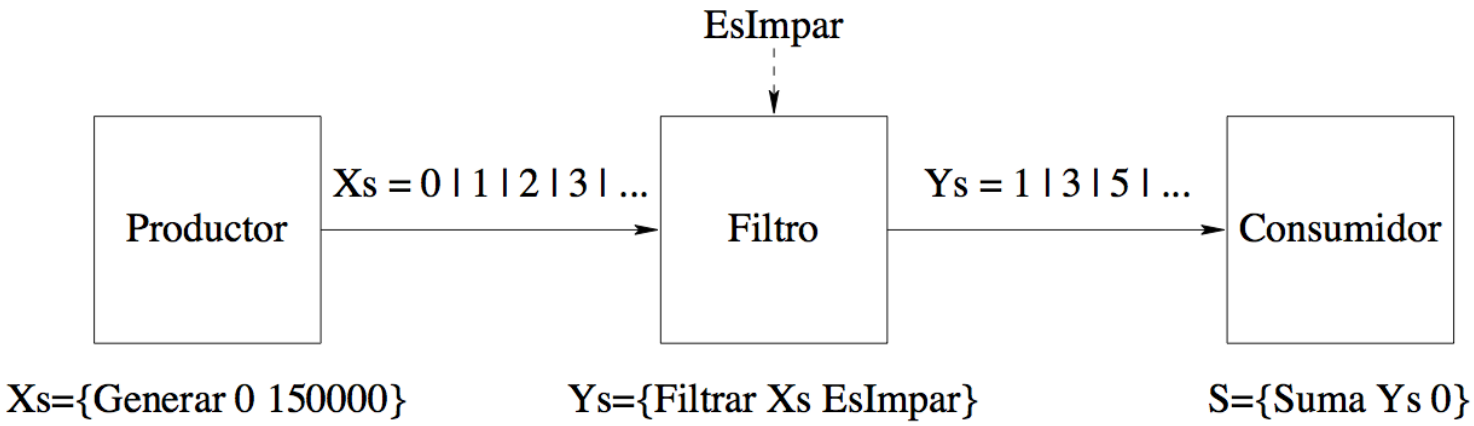
- Los consumidores no interfieren con los otros porque ellos realmente no “consumen” el flujo; sólo lo leen.

Transductores y canales

- Transductor: objeto flujo que lee el flujo del consumidor y crea otro flujo que es leido por un consumidor/transductor.
- Canal: secuencia de objetos flujo, cada uno de los cuales alimenta al siguiente.
- Filtro: transductor sencillo.

Plan

La comunicación por flujos (4)



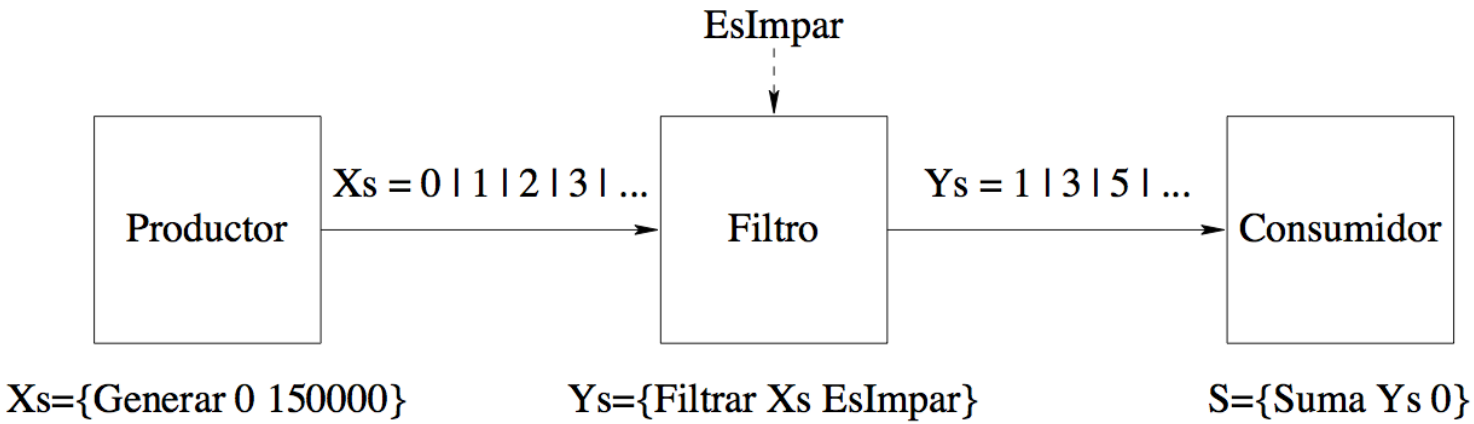
Filtro

```
local Xs Ys S in
   thread Xs={Generar 0 150000} end
   thread Ys={Filter Xs EsImpar} end
   thread S={Suma Ys 0} end
   {Browse S}
end
```

```
EsImpar

fun {EsImpar X}
   X mod 2 \= 0
end
```

La comunicación por flujos (4)



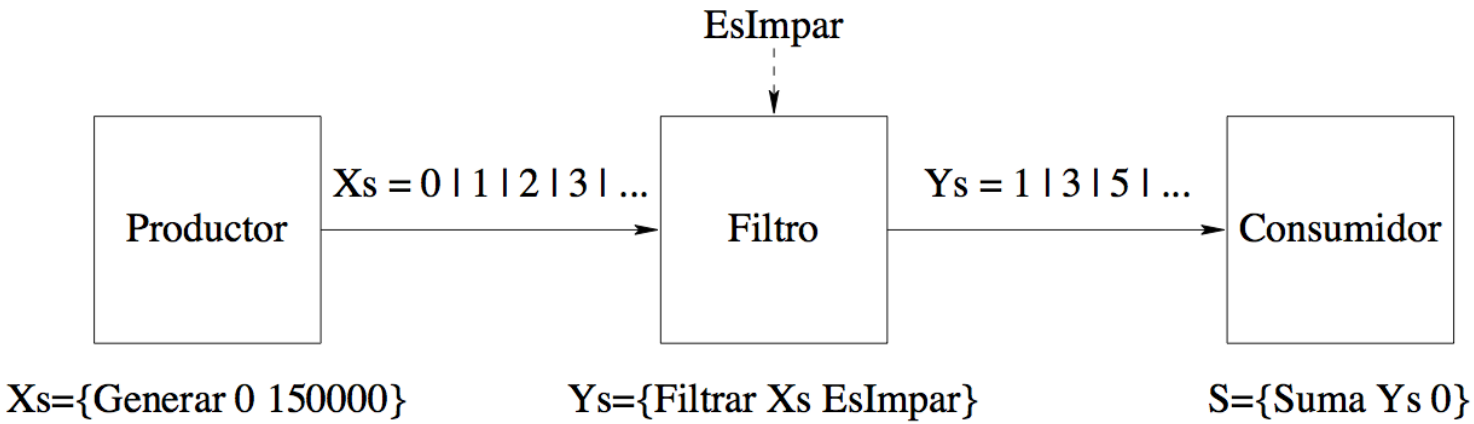
Filtro

```
local Xs Ys S in
   thread Xs={Generar 0 150000} end
   thread Ys={Filter Xs EsImpar} end
   thread S={Suma Ys 0} end
   {Browse S}
end
```

```
EsImpar

fun {EsImpar X}
   X mod 2 \= 0
end
```

La comunicación por flujos (4)



Filtro

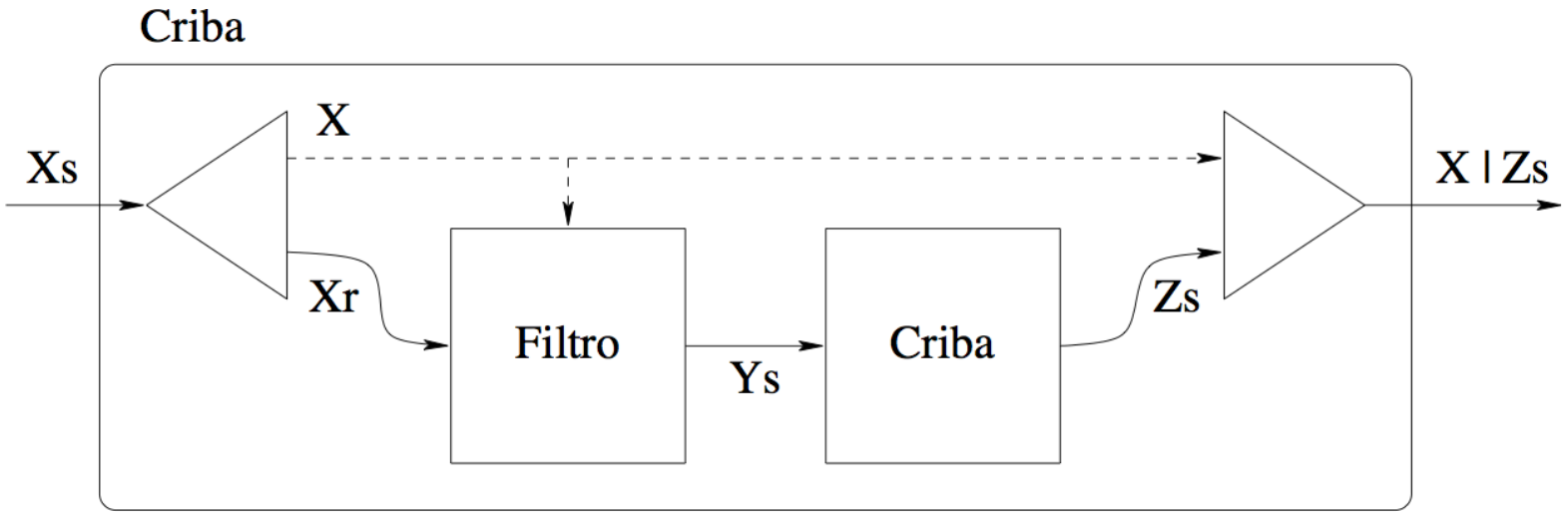
```
local Xs Ys S in
   thread Xs={Generar 0 150000} end
   thread Ys={Filter Xs EsImpar} end
   thread S={Suma Ys 0} end
   {Browse S}
end
```

```
EsImpar

fun {EsImpar X}
   X mod 2 \= 0
end
```

Plan

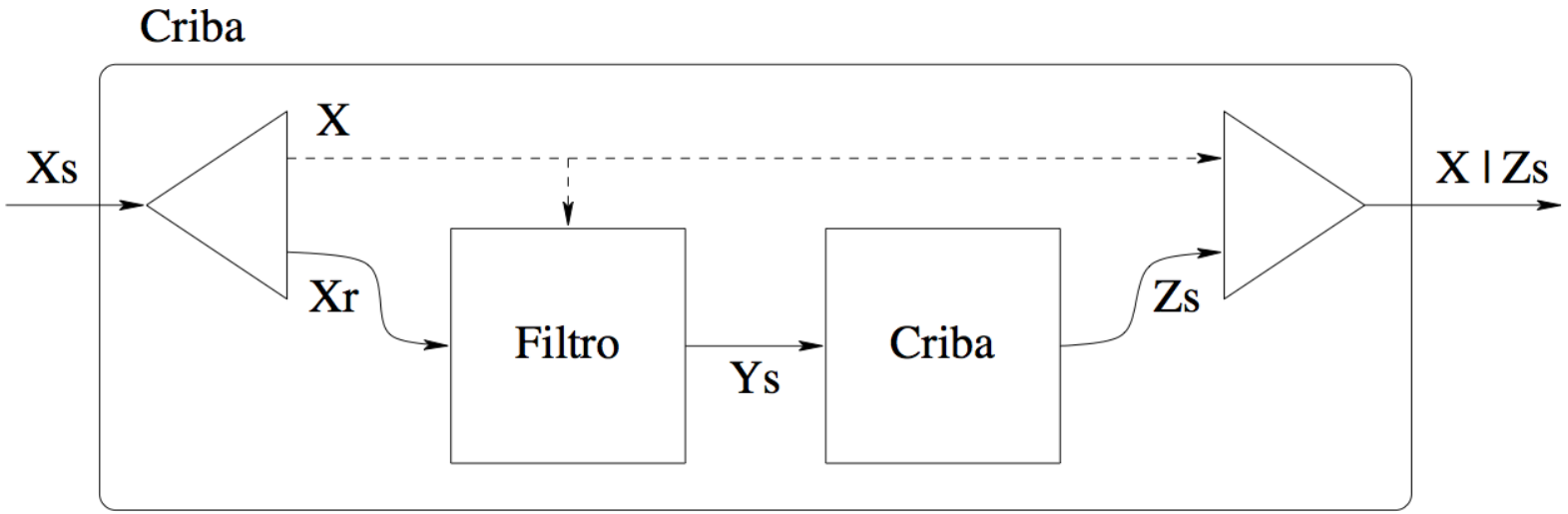
La comunicación por flujos (5)



La criba de Eratóstenes

- Criba: Selección o elección de lo que interesa.
- Primos: Naturales sin divisores distintos de 1 y el número.
- Filtrar sucesivamente los números no primos del flujo, hasta que sólo queden números primos.
- Los filtros se crean dinámicamente.

La comunicación por flujos (5)

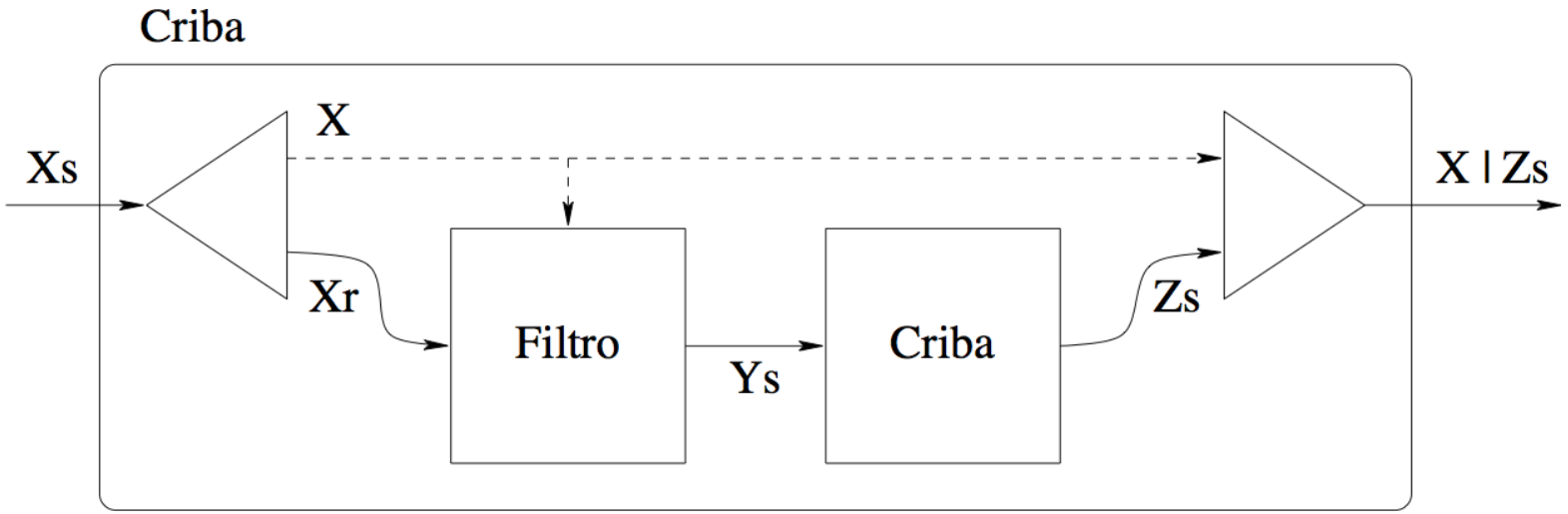


La criba de Eratóstenes

- Criba: Selección o elección de lo que interesa.
- Primos: Naturales sin divisores distintos de 1 y el número.
- Filtrar sucesivamente los números no primos del flujo, hasta que sólo queden números primos.
- Los filtros se crean dinámicamente.

Universidad del Valle

La comunicación por flujos (5)

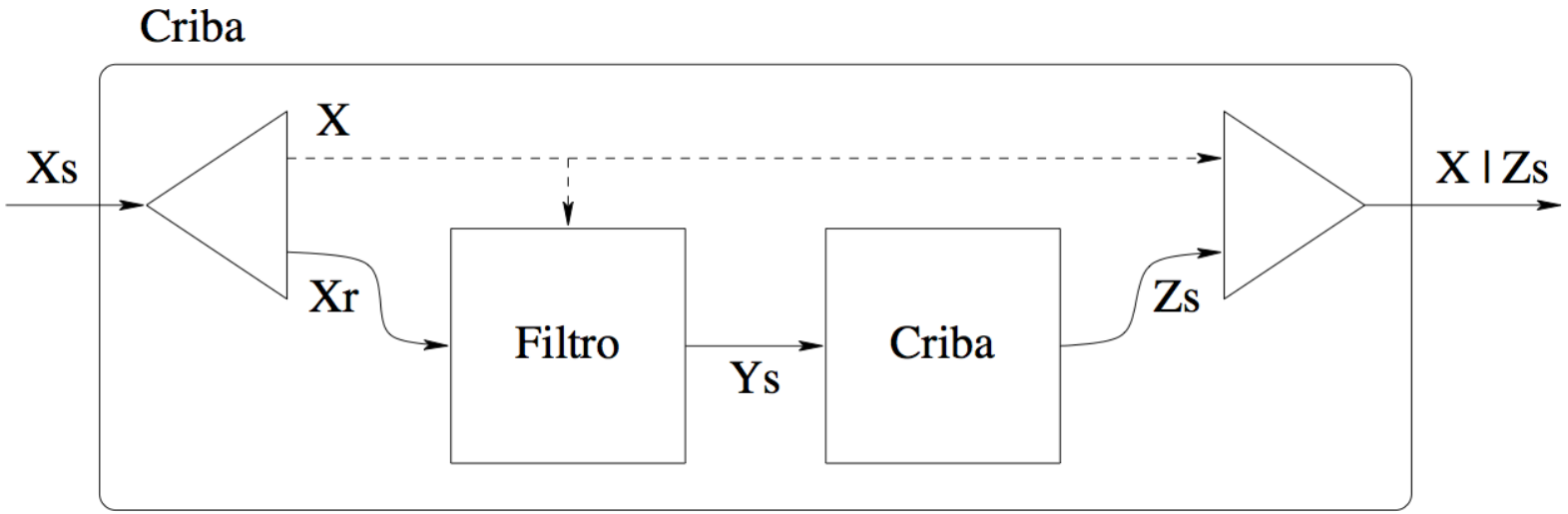


La criba de Eratóstenes

- Criba: Selección o elección de lo que interesa.
- Primos: Naturales sin divisores distintos de 1 y el número.
- Filtrar sucesivamente los números no primos del flujo, hasta que sólo queden números primos.
- Los filtros se crean dinámicamente.

Universidad del Valle

La comunicación por flujos (5)



La criba de Eratóstenes

- Criba: Selección o elección de lo que interesa.
- Primos: Naturales sin divisores distintos de 1 y el número.
- Filtrar sucesivamente los números no primos del flujo, hasta que sólo queden números primos.
- Los filtros se crean dinámicamente.

La comunicación por flujos (6)

La criba en Oz

```
fun {Criba Xs}
   case Xs
   of nil then nil
   [] X|Xr then Ys in
      thread
       Ys={Filter
            Xr
            fun {$ Y}
             Y mod X \= 0
            end}
      end
      X|{Criba Ys}
   end
end
```

Uso de la criba

```
local Xs Ys in
 thread Xs={Generar 2
            100000}
 end
 thread Ys={Criba Xs} end
 {Browse Ys}
end
```

La comunicación por flujos (6)

La criba en Oz

```
fun {Criba Xs}
   case Xs
   of nil then nil
   [] X|Xr then Ys in
      thread
       Ys={Filter
            Xr
            fun {$ Y}
             Y mod X \= 0
            end}
      end
      X|{Criba Ys}
   end
end
```

Uso de la criba

```
local Xs Ys in
 thread Xs={Generar 2
            100000}
 end
 thread Ys={Criba Xs} end
 {Browse Ys}
end
```

Plan

La comunicación por flujos (7)

Control de flujo

- ¿Qué pasa si el productor genera elementos a mayor velocidad de la que el consumidor puede consumirlos?
- Los elementos no consumidos se pueden llegar a apilar y a monopolizar los recursos del sistema.
- Una solución: limitar la rata a la cual el productor genera elementos nuevos, de manera que alguna condición global se satisfaga.
- ¿Cómo implementarlo?: concurrencia dirigida por la demanda, Memoria limitada y prioridades sobre los hilos (poco elegante).

Concurrencia dirigida por la demanda

La comunicación por flujos (7)

Control de flujo

- ¿Qué pasa si el productor genera elementos a mayor velocidad de la que el consumidor puede consumirlos?
- Los elementos no consumidos se pueden llegar a apilar y a monopolizar los recursos del sistema.
- Una solución: limitar la rata a la cual el productor genera elementos nuevos, de manera que alguna condición global se satisfaga.
- ¿Cómo implementarlo?: concurrencia dirigida por la demanda, Memoria limitada y prioridades sobre los hilos (poco elegante).

Concurrencia dirigida por la demanda

La comunicación por flujos (7)

Control de flujo

- ¿Qué pasa si el productor genera elementos a mayor velocidad de la que el consumidor puede consumirlos?
- Los elementos no consumidos se pueden llegar a apilar y a monopolizar los recursos del sistema.
- Una solución: limitar la rata a la cual el productor genera elementos nuevos, de manera que alguna condición global se satisfaga.
- ¿Cómo implementarlo?: concurrencia dirigida por la demanda, Memoria limitada y prioridades sobre los hilos (poco elegante).

Concurrencia dirigida por la demanda

- También llamada ejecución perezosa.
- El productor sólo genera elementos cuando el consumidor explícitamente lo solicita.

La comunicación por flujos (7)

Control de flujo

- ¿Qué pasa si el productor genera elementos a mayor velocidad de la que el consumidor puede consumirlos?
- Los elementos no consumidos se pueden llegar a apilar y a monopolizar los recursos del sistema.
- Una solución: limitar la rata a la cual el productor genera elementos nuevos, de manera que alguna condición global se satisfaga.
- ¿Cómo implementarlo?: concurrencia dirigida por la demanda, Memoria limitada y prioridades sobre los hilos (poco elegante).

Concurrencia dirigida por la demanda

- También llamada ejecución perezosa.
- El productor sólo genera elementos cuando el consumidor explícitamente lo solicita.
- Mecanismo de sincronización: variables de flujo de datos.

La comunicación por flujos (7)

Control de flujo

- ¿Qué pasa si el productor genera elementos a mayor velocidad de la que el consumidor puede consumirlos?
- Los elementos no consumidos se pueden llegar a apilar y a monopolizar los recursos del sistema.
- Una solución: limitar la rata a la cual el productor genera elementos nuevos, de manera que alguna condición global se satisfaga.
- ¿Cómo implementarlo?: concurrencia dirigida por la demanda, Memoria limitada y prioridades sobre los hilos (poco elegante).

Concurrencia dirigida por la demanda

- También llamada ejecución perezosa.
- El productor sólo genera elementos cuando el consumidor explícitamente lo solicita.
- Mecanismo de sincronización: variables de flujo de datos.

La comunicación por flujos (7)

Control de flujo

- ¿Qué pasa si el productor genera elementos a mayor velocidad de la que el consumidor puede consumirlos?
- Los elementos no consumidos se pueden llegar a apilar y a monopolizar los recursos del sistema.
- Una solución: limitar la rata a la cual el productor genera elementos nuevos, de manera que alguna condición global se satisfaga.
- ¿Cómo implementarlo?: concurrencia dirigida por la demanda, Memoria limitada y prioridades sobre los hilos (poco elegante).

Concurrencia dirigida por la demanda

- También llamada ejecución perezosa.
- El productor sólo genera elementos cuando el consumidor explícitamente lo solicita.
- Mecanismo de sincronización: variables de flujo de datos.

La comunicación por flujos (7)

Control de flujo

- ¿Qué pasa si el productor genera elementos a mayor velocidad de la que el consumidor puede consumirlos?
- Los elementos no consumidos se pueden llegar a apilar y a monopolizar los recursos del sistema.
- Una solución: limitar la rata a la cual el productor genera elementos nuevos, de manera que alguna condición global se satisfaga.
- ¿Cómo implementarlo?: concurrencia dirigida por la demanda, Memoria limitada y prioridades sobre los hilos (poco elegante).

Concurrencia dirigida por la demanda

- También llamada ejecución perezosa.
- El productor sólo genera elementos cuando el consumidor explícitamente lo solicita.
- Mecanismo de sincronización: variables de flujo de datos.

La comunicación por flujos (8)

Productor con control

```
proc {DGenerar N Xs}
   case Xs of X|Xr then
      X=N
      {DGenerar N+1 Xr}
   end
end
```

Consumidor

```
fun {DSuma ?Xs A Limit}
   if Limit>0 then
      X|Xr=Xs
   in
      {DSuma Xr A+X Limit-1}
   else A end
end
```

Uso

```
local Xs S in
 thread {DGenerar 0
                  Xs}
 end
 thread S={DSuma Xs
                 0
                 150}
 end
 {Browse S}
end
```

- El consumidor controla el flujo.
- Ejecución perezosa (lazy).

La comunicación por flujos (8)

Productor con control

```
proc {DGenerar N Xs}
   case Xs of X|Xr then
      X=N
      {DGenerar N+1 Xr}
   end
end
```

Consumidor

```
fun {DSuma ?Xs A Limit}
   if Limit>0 then
      X|Xr=Xs
   in
      {DSuma Xr A+X Limit-1}
   else A end
end
```

Uso

```
local Xs S in
 thread {DGenerar 0
                  Xs}
 end
 thread S={DSuma Xs
                 0
                 150}
 end
 {Browse S}
end
```

- El consumidor controla el flujo.
- Ejecución perezosa explícita.

La comunicación por flujos (8)

Productor con control

```
proc {DGenerar N Xs}
   case Xs of X|Xr then
      X=N
      {DGenerar N+1 Xr}
   end
end
```

Consumidor

```
fun {DSuma ?Xs A Limit}
   if Limit>0 then
      X|Xr=Xs
   in
      {DSuma Xr A+X Limit-1}
   else A end
end
```

Uso

```
local Xs S in
 thread {DGenerar 0
                  Xs}
 end
 thread S={DSuma Xs
                 0
                 150}
 end
 {Browse S}
end
```

- El consumidor controla el flujo.
- Ejecución perezosa explícita.

La comunicación por flujos (8)

Productor con control

```
proc {DGenerar N Xs}
   case Xs of X|Xr then
      X=N
      {DGenerar N+1 Xr}
   end
end
```

Consumidor

```
fun {DSuma ?Xs A Limit}
   if Limit>0 then
      X|Xr=Xs
   in
      {DSuma Xr A+X Limit-1}
   else A end
end
```

Uso

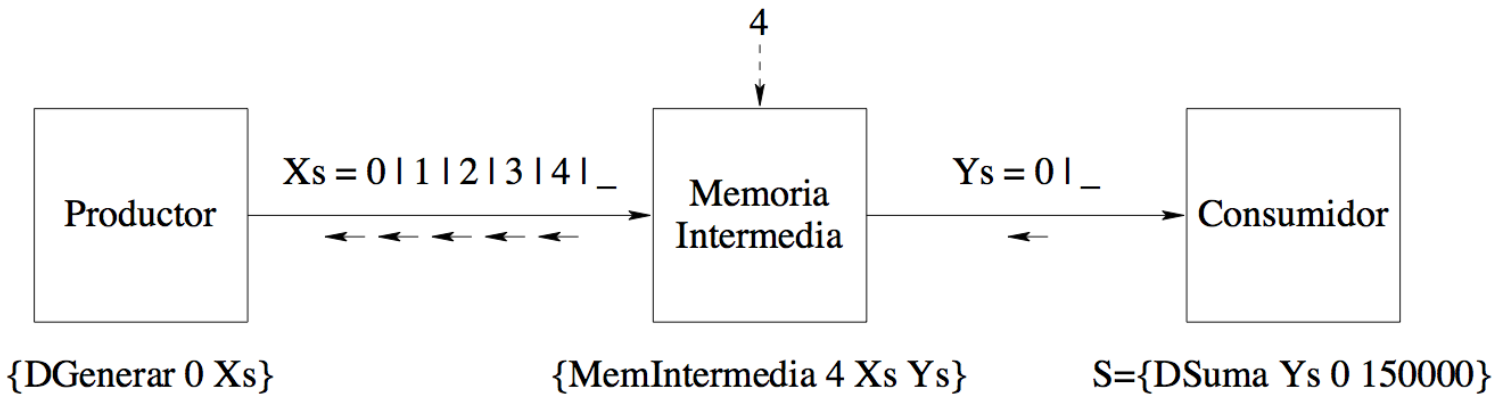
```
local Xs S in
 thread {DGenerar 0
                  Xs}
 end
 thread S={DSuma Xs
                 0
                 150}
 end
 {Browse S}
end
```

- El consumidor controla el flujo.
- Ejecución perezosa explícita.

La comunicación por flujos (9)

Control del flujo con una memoria intermedia limitada

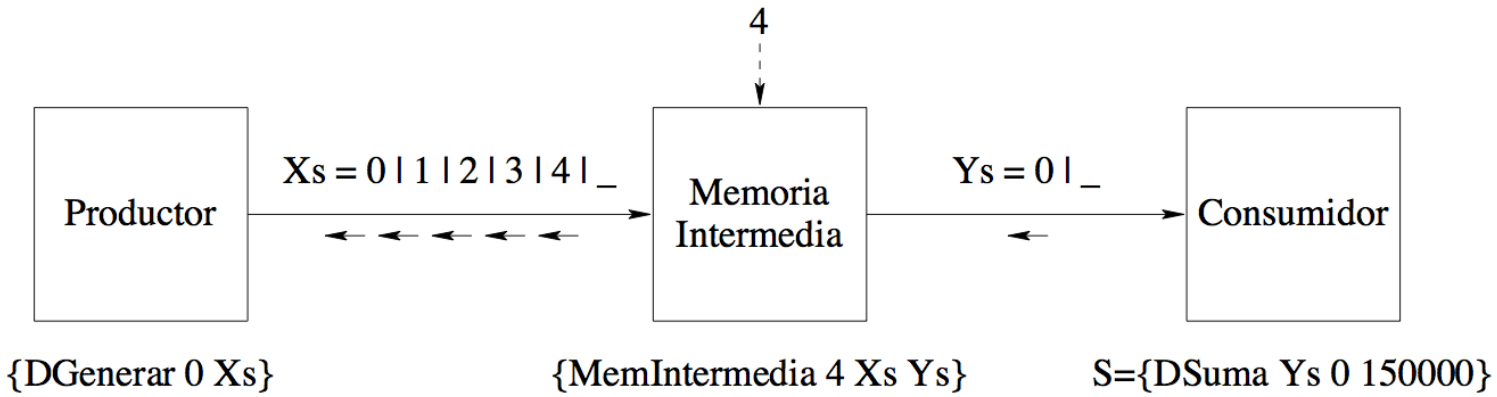
- Ejecución ansiosa: el productor está completamente libre $\longrightarrow$ Explosión de recursos.
- En la ejecución perezosa, el productor está completamente restringido $\longrightarrow$ reducción del rendimiento.



La comunicación por flujos (9)

Control del flujo con una memoria intermedia limitada

- Ejecución ansiosa: el productor está completamente libre $\longrightarrow$ Explosión de recursos.
- En la ejecución perezosa, el productor está completamente restringido $\longrightarrow$ reducción del rendimiento.
- Memoria intermedia limitada: Combinación de ejec. ansiosa y perezosa.
- Memoria intermedia limitada: transductor que almacena elementos hasta en un número máximo, digamos n.

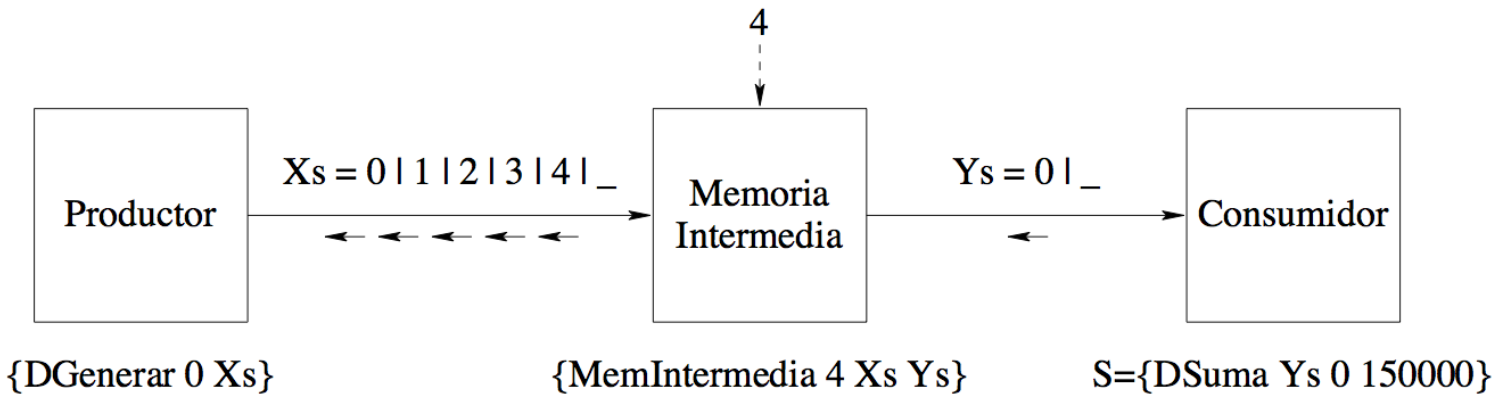


La comunicación por flujos (9)

Control del flujo con una memoria intermedia limitada

- Ejecución ansiosa: el productor está completamente libre $\longrightarrow$ Explosión de recursos.
- En la ejecución perezosa, el productor está completamente restringido $\longrightarrow$ reducción del rendimiento.

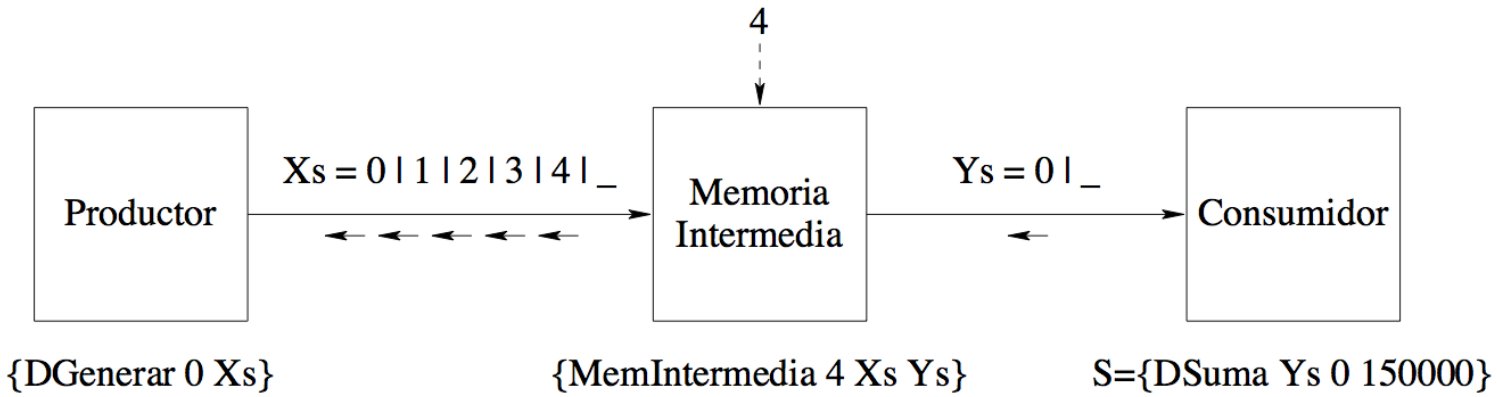
- Memoria intermedia limitada: Combinación de ejec. ansiosa y perezosa.
- Memoria intermedia limitada: transductor que almacena elementos hasta en un número máximo, digamos n.



La comunicación por flujos (9)

Control del flujo con una memoria intermedia limitada

- Ejecución ansiosa: el productor está completamente libre $\longrightarrow$ Explosión de recursos.
- En la ejecución perezosa, el productor está completamente restringido $\longrightarrow$ reducción del rendimiento.
- Memoria intermedia limitada: Combinación de ejec. ansiosa y perezosa.
- Memoria intermedia limitada: transductor que almacena elementos hasta en un número máximo, digamos n.

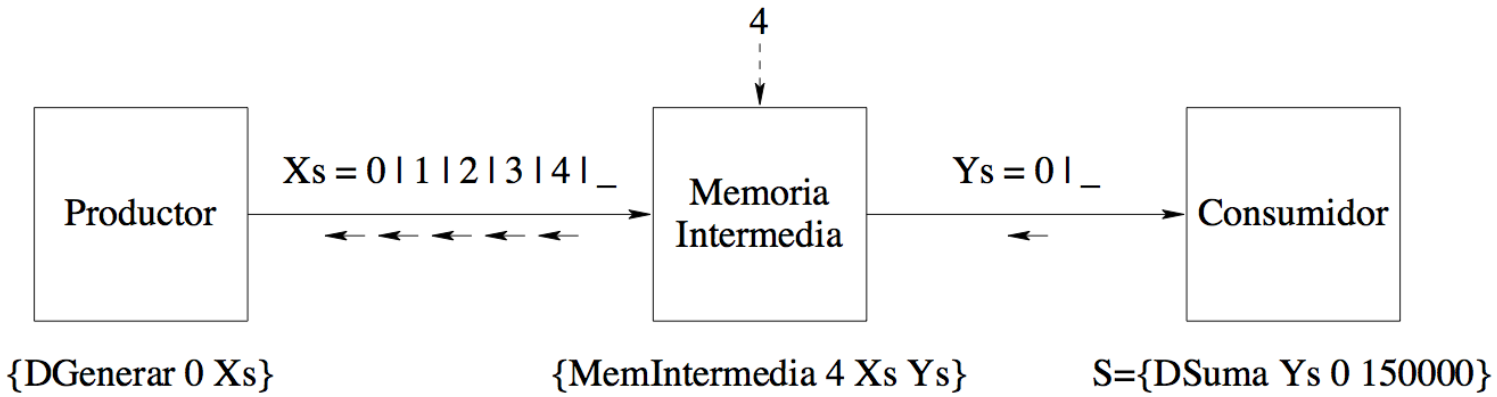


La comunicación por flujos (9)

Control del flujo con una memoria intermedia limitada

- Ejecución ansiosa: el productor está completamente libre $\longrightarrow$ Explosión de recursos.
- En la ejecución perezosa, el productor está completamente restringido $\longrightarrow$ reducción del rendimiento.

- Memoria intermedia limitada: Combinación de ejec. ansiosa y perezosa.
- Memoria intermedia limitada: transductor que almacena elementos hasta en un número máximo, digamos n.



La comunicación por flujos (10)

Código memoria intermedia limitada

```
proc {MemIntermedia N ?Xs Ys}
   fun {Iniciar N ?Xs}
      if N==0 then Xs
      else Xr in Xs=_|Xr {Iniciar N-1 Xr} end
   end
   proc {CicloAtender Ys ?Xs ?Final}
      case Ys of Y|Yr then Xr Final2 in
         Xs=Y|Xr
         Final=_|Final2
         {CicloAtender Yr Xr Final2}
      end
   end
   Final={Iniciar N Xs}
in
   {CicloAtender Ys Xs Final}
end
```

La comunicación por flujos (11)

Ejemplo de uso

```
local Xs Ys S in
   thread {DGenerar 0 Xs} end
   thread {MemIntermedia 4 Xs Ys} end
   thread S={DSuma Ys 0 150000} end
   {Browse Xs} {Browse Ys}
   {Browse S}
end
```