



Streams

- A **stream** is a list that ends in an unbound variable
 - $S = a|b|c|d|S2$
 - A stream can be extended with new elements as long as necessary
 - The stream can be closed by binding the end to nil
- A stream can be used as a **communication channel** between two threads
 - The first thread adds elements to the stream
 - The second thread reads the stream

Programming with streams



- This program displays the elements of a stream as they appear:

```
proc {Disp S}  
  case S of X|S2 then {Browse X} {Disp S2} end  
end  
declare S  
thread {Disp S} end
```

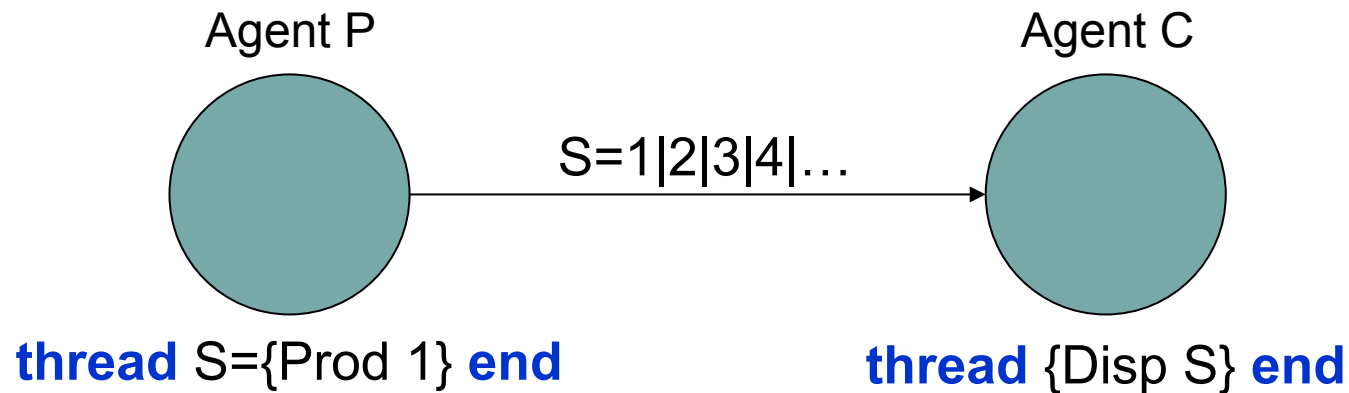
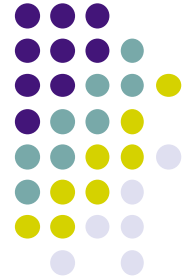
- We can add elements gradually:
 declare S2 **in** S=a|b|c|S2
 declare S3 **in** S2=d|e|f|S3
- Try it yourself!

Producer/ consumer (1)



- A **producer** generates a stream of data
fun {Prod N} {Delay 1000} N|{Prod N+1} **end**
 - The {Delay 1000} slows down execution enough to observe it
- A **consumer** reads the stream and performs some action (like the Disp procedure)
- A producer/consumer program:
declare S
thread S={Prod 1} **end**
thread {Disp S} **end**

Producer/ consumer (2)



- Each circle is **a concurrent activity that reads and writes streams**
 - We call this an **agent**
- Agents P and C communicate through stream S
 - The first thread creates the stream, the second reads it



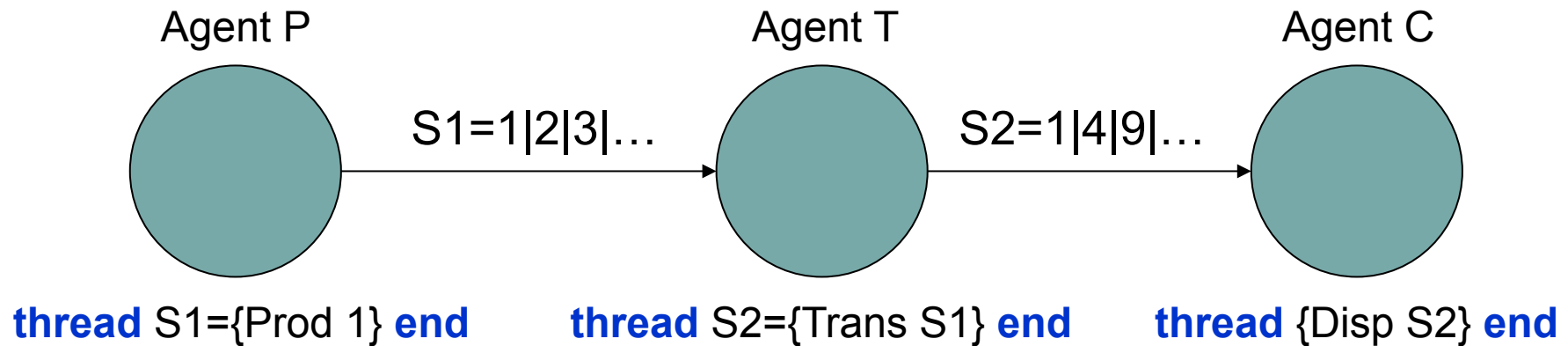
Pipeline (1)

- We can add more agents between P and C
- Here is a **transformer** that modifies the stream:

```
fun {Trans S}  
  case S of X|S2 then X*X|{Trans S2} end  
end
```
- This program has three agents:

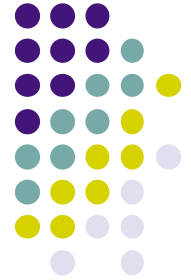
```
declare S1 S2  
thread S1={Prod 1} end  
thread S2={Trans S1} end  
thread {Disp S2} end
```

Pipeline (2)



- We now have three agents
 - The producer (agent P) creates stream S1
 - The transformer (agent T) reads S1 and creates S2
 - The consumer (agent C) reads S2
- The pipeline is a very useful technique!
 - For example, it is **omnipresent in operating systems since Unix**

Agents



- An agent is a concurrent activity that reads and writes streams
 - The simplest agent is **a list function executing in one thread**
 - Since list functions are tail-recursive, the agent can execute with a fixed memory size
 - This is **the deep reason why single assignment is important:** it makes tail-recursive list functions, which makes deterministic dataflow into a practical paradigm
- All list functions can be used as agents
 - All functional programming techniques can be used in deterministic dataflow
 - Including higher-order programming! In the next lesson will see more examples of the power of the model.