OCCAM

OCCAM enables an application to be described in terms of

— concurrent processes
— communication channels

Each process describes the behavior of one component of the implementation

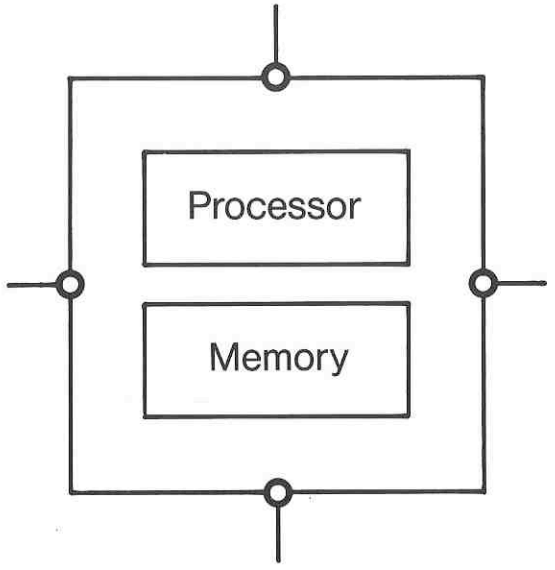
Each channel describes a connection between components

VLSI

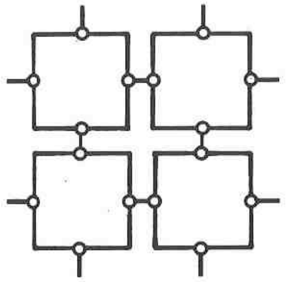
Many identical devices can be manufactured economically

OCCAM can be implemented using identical VLSI devices, each programmed with an OCCAM process

Programmable component
– Transputer



Serial communication links



inmos

OCCAM

Same language used for:

Transputer System description
Programming of individual Transputers

Language primitives chosen to ensure efficient distributed implementation

Transputer designed to match OCCAM primitives

Transputer and OCCAM

OCCAM program describes system using

- concurrent processes
- communication channels

OCCAM program can be implemented:

– process	<–>	Transputer
– channel	<–>	Link

Transputer and OCCAM

Also

– many concurrent processes	<–>	Transputer
– channels	<–>	Memory locations

The same OCCAM program can be implemented by

– many Transputers	(high performance)
– one Transputer	(low cost)

OCCAM

OCCAM programs are built from three primitive processes:

v:=e	assign expression e to variable v
c ! e	output expression e to channel c
c ? v	input variable v from channel c

The primitive processes are combined to form constructs:

SEQ	sequence
IF	conditional
WHILE	iteration
PAR	parallel
ALT	alternative

A construct is itself a process, and may be used as a component of another construct.

Communication

Channel is

- point-to-point
- one way

Communication is synchronised

- one process waits for the other

When both processes are ready

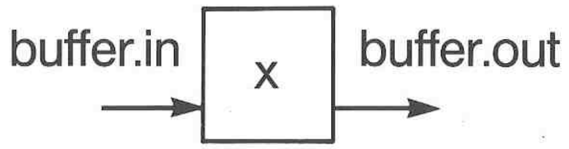
- data is copied
- both processes continue

Sequential Constructor

The sequential constructor causes processes to be executed one after another

Example:

```
WHILE TRUE
  VAR x:
  SEQ
    buffer.in ? x
    buffer.out ! x
```



This simple buffer repeatedly inputs a value, then outputs it.

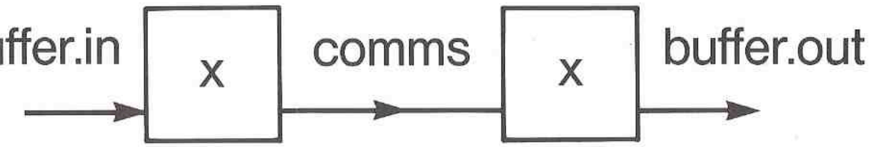
The sequential constructor causes the output to take place after the input is completed.

Parallel Constructor

The parallel constructor causes processes to be executed together

Example:

```
CHAN comms:
PAR
  WHILE TRUE
    VAR x:
    SEQ
      buffer.in ? x
      comms ! x
  WHILE TRUE
    VAR x:
    SEQ
      comms ? x
      buffer.out ! x
```



Here two simple buffers are executed together, allowing up to two values to be buffered.

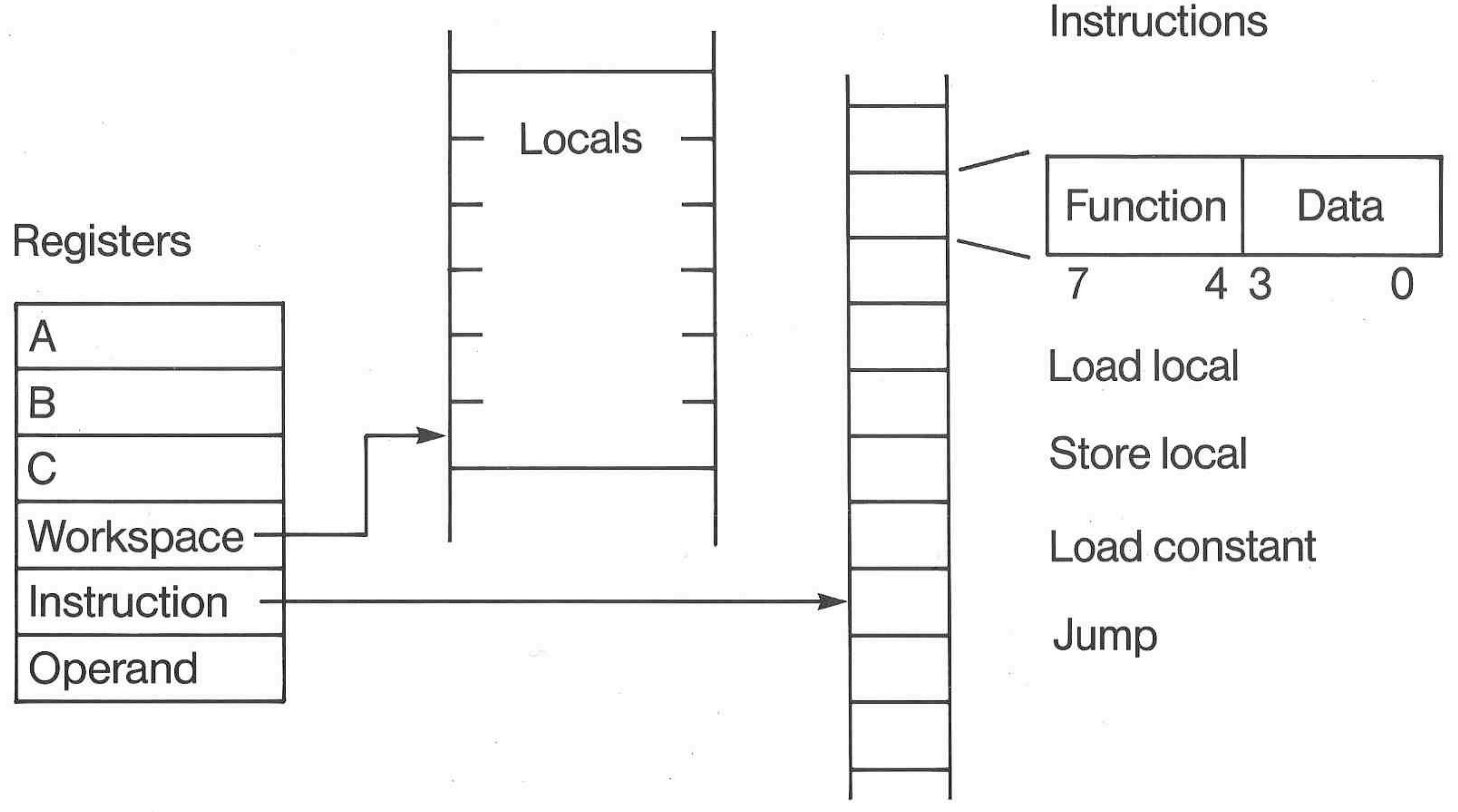
Sequential Processes

Programmable sequential computer efficiently implements simple programs:

variables
expression evaluation
assignment
SEQ
IF
WHILE

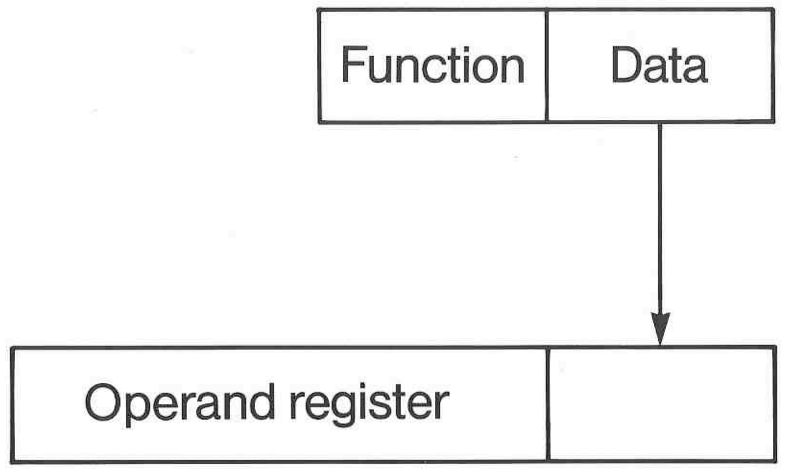
Transputer processor provides a simple set of instructions for sequential program execution

Sequential Execution



inmos

Forming a Long Operand



Load data into operand register

PFIX

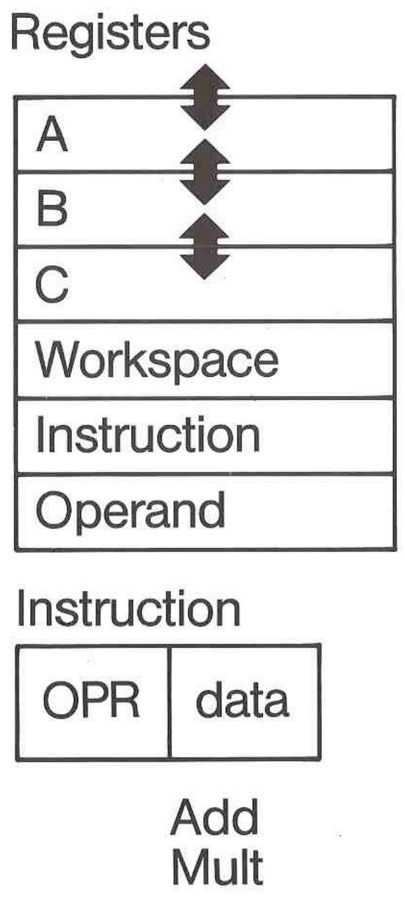
Shift operand register
left four places

Other instructions

Perform function
using operand register

Clear operand register

Expression Evaluation



Expressions are evaluated on a short stack

No need to specify registers

Compiler introduces necessary temporary variables

Addresses also evaluated on stack

OPR operand selects an operation on the stack

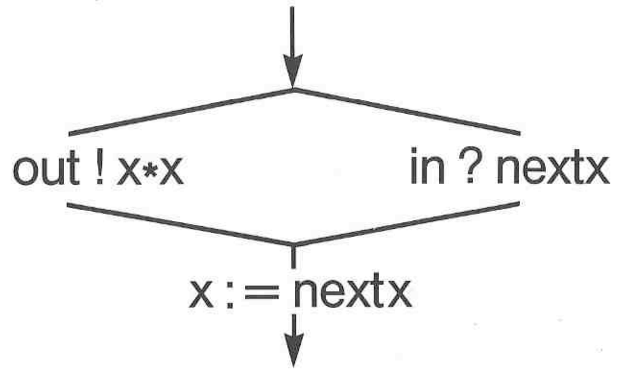
Prefixing OPR expands the number of operations

Concurrent Processes

OCCAM process executed by a single transputer may consist of any number of concurrent processes

A sequential process may include a parallel:

```
SEQ
  PAR
    out ! x*x
    in ? nextx
  x := nextx
```



Space for concurrent processes is allocated by the OCCAM compiler

=> No storage allocation overheads

Special instruction ensures correct termination of PAR

inmos

Concurrent Process Execution

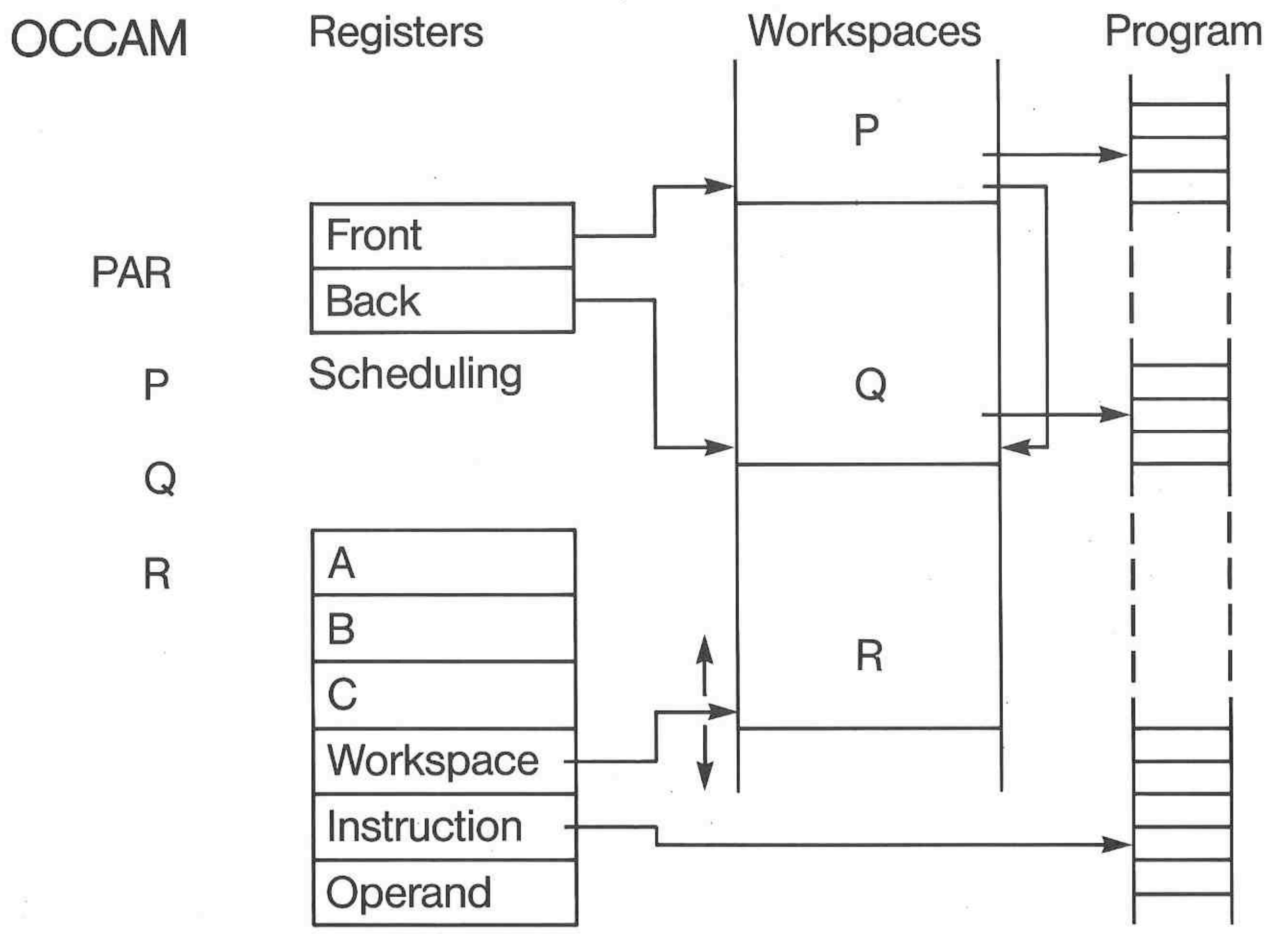
Transputer executes concurrent processes using a linked list of processes awaiting execution

At any time, a concurrent process may be

active	– being executed
	– on the list awaiting execution
inactive	– waiting (ready) to input
	– waiting (ready) to output
	– waiting until a specified time

Inactive processes do not consume any processor time

Parallel Execution



Communication

OCCAM input and output are implemented directly
by transputer instructions

Channel between two processes can be implemented by

- word in memory
- serial link

Same instructions are used in each case

Communication

OCCAM channel:

a one way communication path between two concurrent processes

Communication is synchronised and unbufferred:

when both the inputting and outputting processes are ready the data is copied

=>

Channel needs

no message queue
no process queue
no data buffer

Internal Communication

Any memory location can be used as an internal channel

Both processes must be ready before communication takes place

When first process becomes ready:

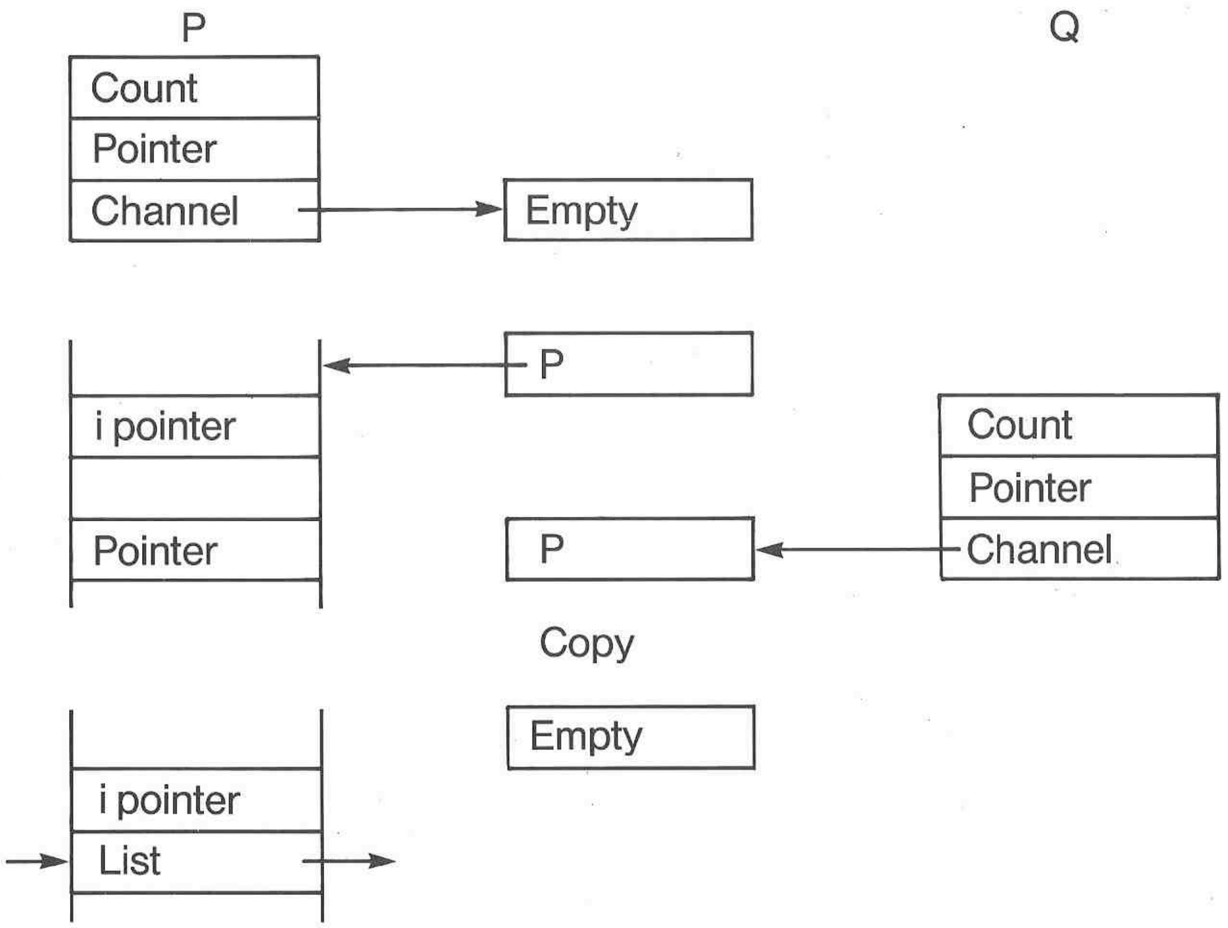
- it is descheduled
- its identity is stored in the channel

When second process becomes ready:

- message copied by the processor
- first process rescheduled
- channel returned to empty state

Either the inputting or the outputting process may become ready first

Communication on a Transputer



inmos

External Communication

Each transputer link provides one channel in each direction

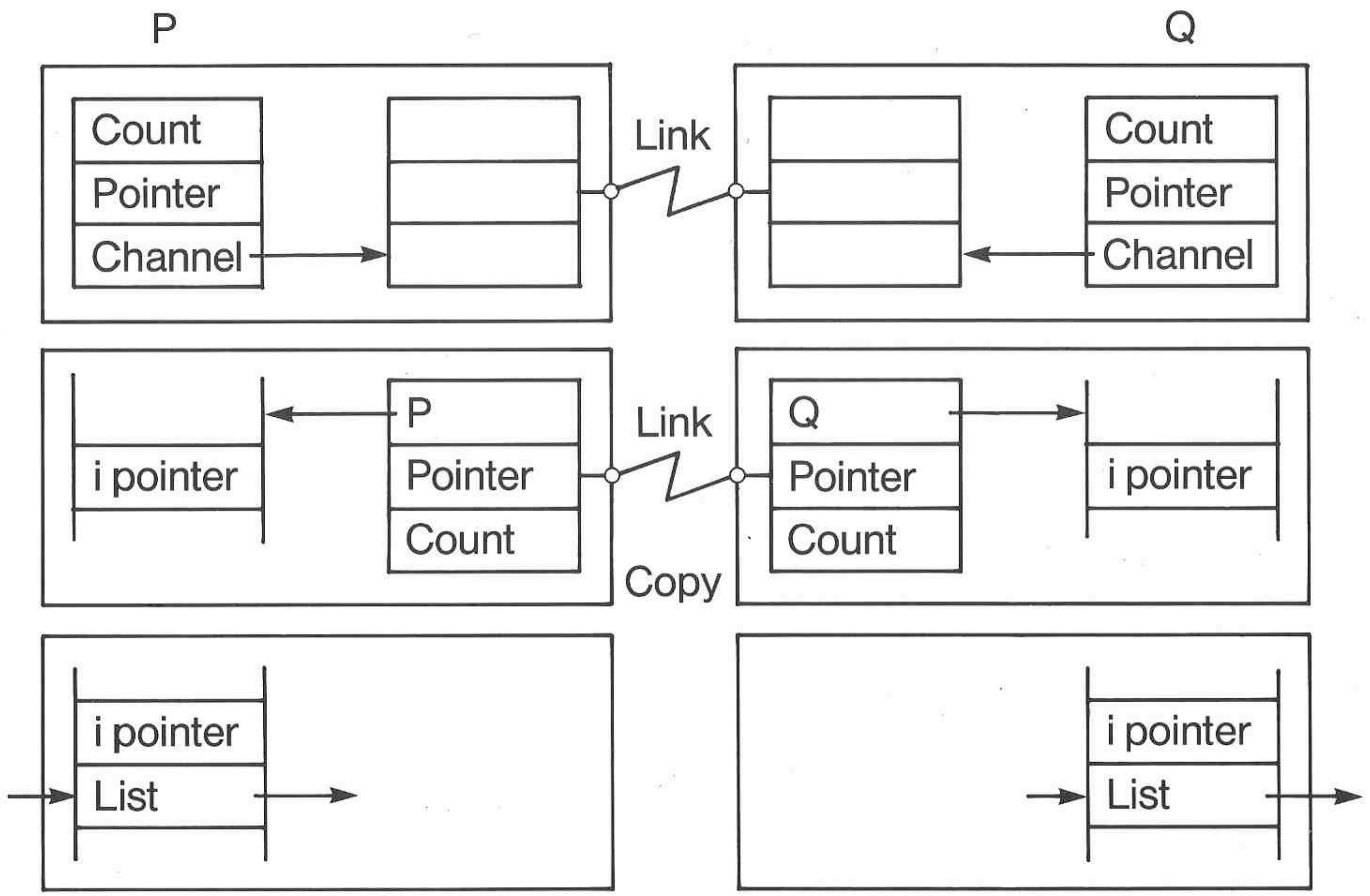
The transfer of data is performed by autonomous link controllers in the transputer

Both inputting and outputting processes

– descheduled whilst transfer takes place
– rescheduled when transfer complete

Either the inputting or the outputting process may become ready first

Communication Between Transputers



Transputer Link

Synchronised communication
- data must be acknowledged
- need at least one signal wire in each direction

Transputer Serial Link
- only one signal wire in each direction
- one OCCAM channel in each direction

Signal wire carries
- data packets for one channel
- acknowledge packets for the other

Data:

1	1	Data	0

Acknowledge

1	0

Protocol

Message transferred as a sequence of bytes
=> wordlengths of sender and receiver may differ

Each byte acknowledged before the next is sent
=> need only a one byte buffer to receive message

Acknowledge may be sent as soon as reception starts,
provided that
– there is a process waiting for input
– there is room to buffer another byte

=> transmission may be continuous

Performance

OCCAM	Microseconds

```
PAR
  P
  Q
```
	P + Q + 1

```
PAR
  c ! x
  c ? y
```
	1.5