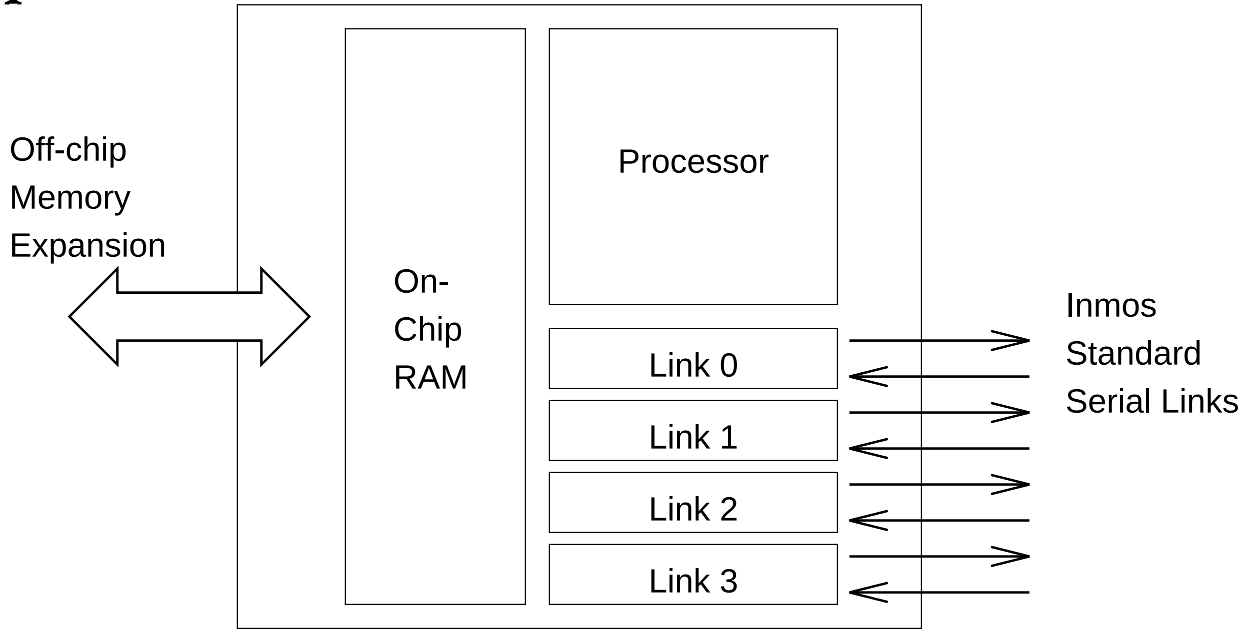
The Transputer

The INMOS Transputer - (SGS-Thompson)

- A Single Chip Microprocessor:
  - CPU
  - RAM
  - I/O
- A Building Block For Parallel Processors:
  - Virtual Concurrency
  - Message Passing
  - Occam Engine

The Transputer

Transputer Structure

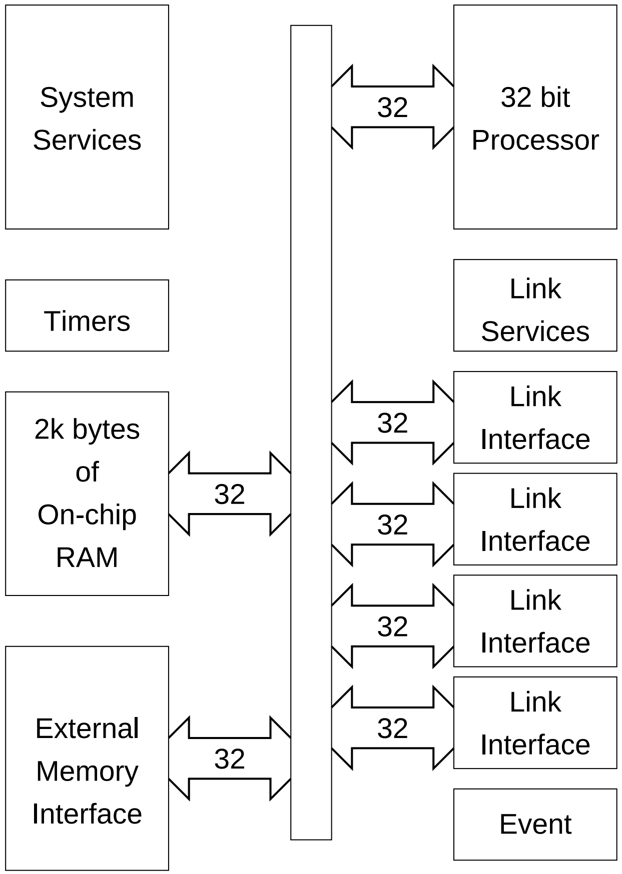


- T414 Transputer:
  - 32 bit CPU
  - 2 kbytes On-chip RAM
  - 4 INMOS Serial Links.

The Transputer - A Single Chip Microprocessor

- Designed for embedded processing and parallel computing.
- Minimum overheads for support circuitry.
  - Minimum requirement is:
    Transputer + Power supply + 5 MHz Clock.
  - *42 Transputers on 9″ by 9″ PCB.*
- Memory interface makes for easy connection to external RAM (if present).
- External ROM is seldom required except in standalone configurations.

T414


System
Services
Timers
2k bytes
of
On-chip
RAM
External
Memory
Interface
32
32
32
32 bit
Processor
Link
Services
Link
Interface
32
Link
Interface
32
Link
Interface
32
Link
Interface
32
Event

Processor

RISC

- Instructions are 8 bits long:
  - 4 bits Function Code
  - 4 bits Data/Operation Code
- Where longer data words or operation codes are required, we use the Prefix instruction to concatenate data nibbles prior to function execution.
- 70% of executed functions are encoded in a single byte.

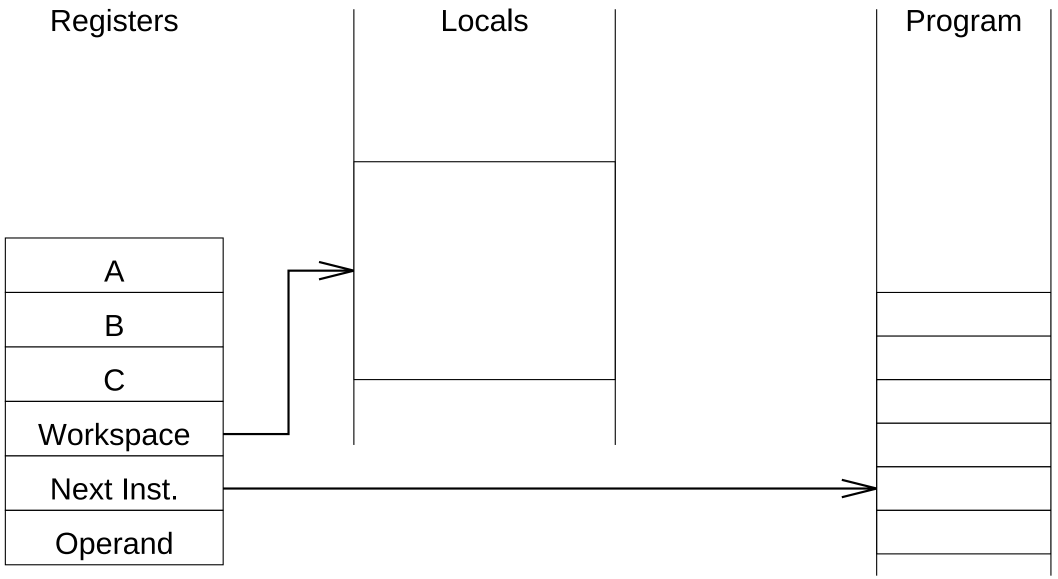
  i.e. without use of the Prefix instruction.
- Many single byte instructions take only one cycle to complete. (e.g. `ldc 0`)

Processor

Support for Virtual Concurrency

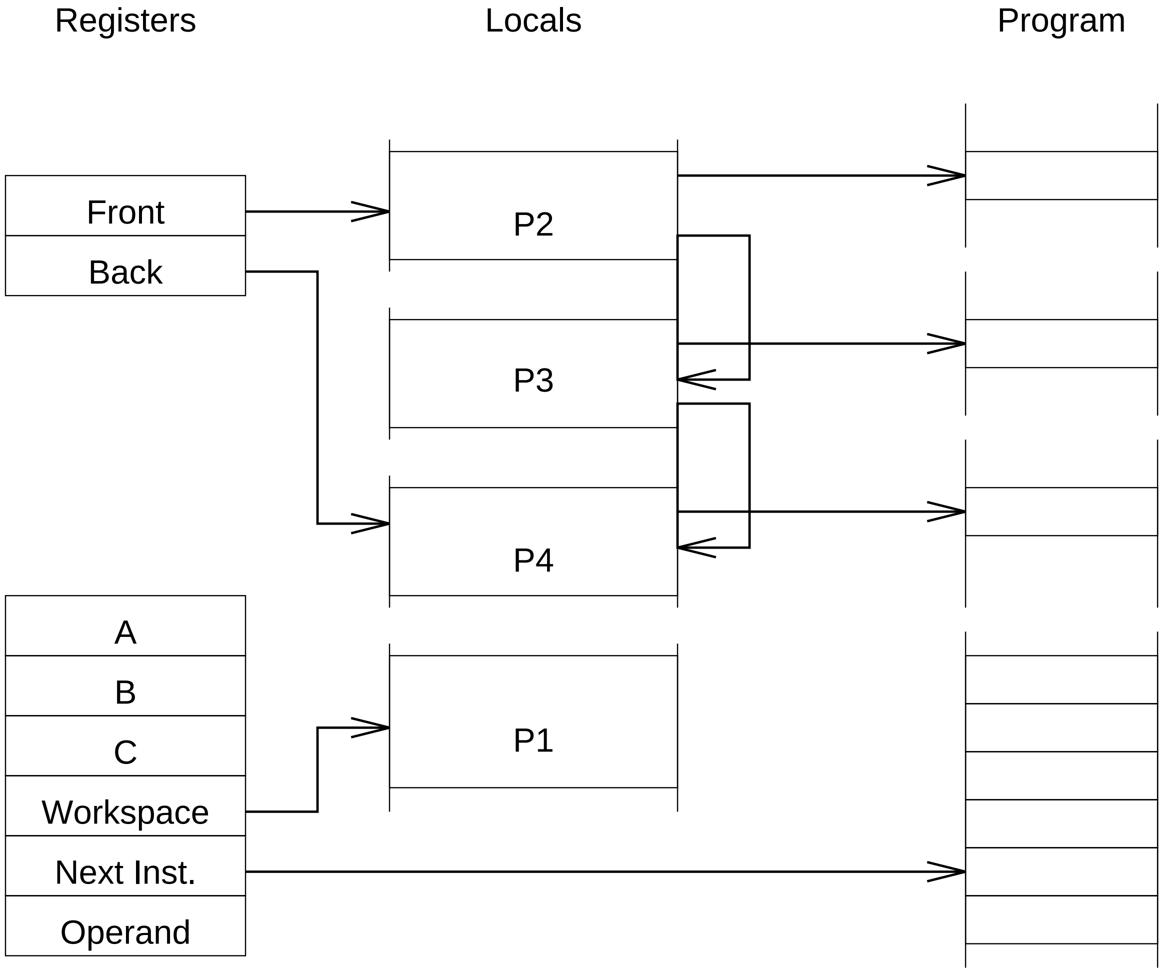
- Microcoded Scheduler
  - Faster & Simpler than scheduler in software kernel.
- A Linked List of Active Processes.
  - Scheduler cycles through list.
- Minimum of Internal Registers.
  - Allows rapid process switching.

Process being Executed



- Three Register Stack : A B C
- Workspace Pointer points to Process Local Workspace.
- Instruction Pointer points to Next Instruction.
- Operand Register stores concatenated Data Words or Operation Codes.

Active Process Queue

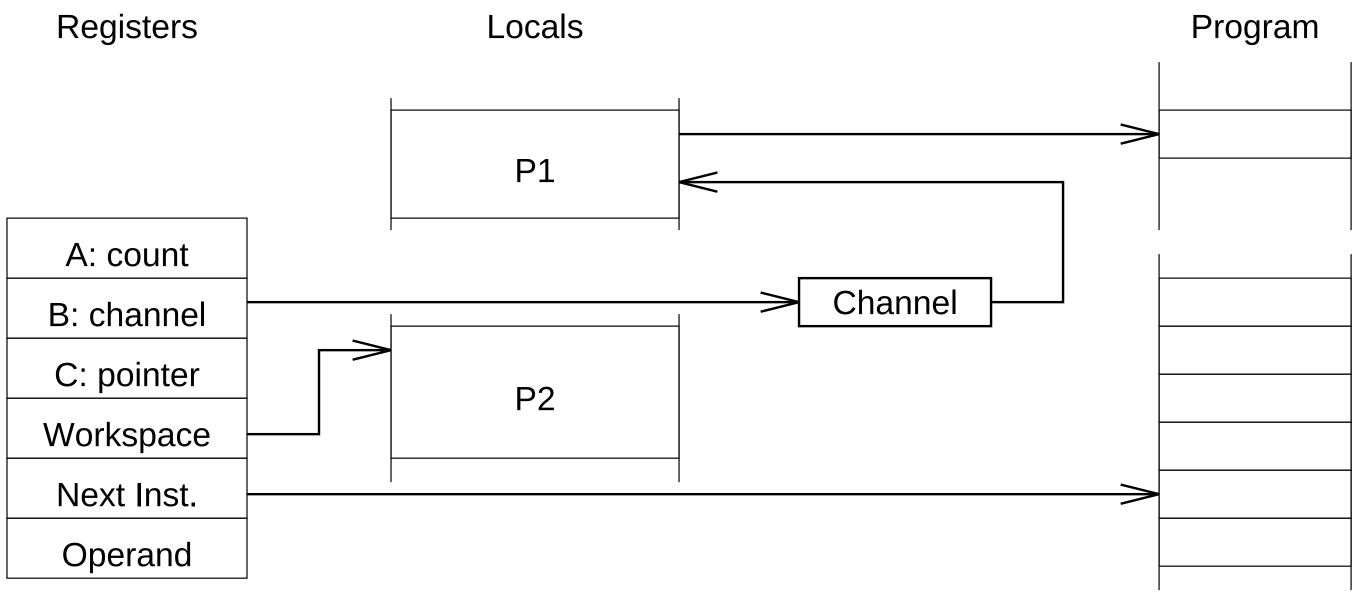


Active Process Queue

- Process **P1** is Active and Executing.
- Processes **P2-P4** form the Active Process Queue.
- When **P1** comes to the end of its timeslice:
  - **P1** completes any arithmetic operation, leaving the stack empty.
  - **P1**'s Instruction Pointer is stored in its workspace.
  - **P1**'s Workspace is added to the back of the active queue.
  - **P2**'s Workspace is taken from the front of the queue.
  - **P2**'s Workspace Pointer and Instruction Pointer are loaded into the appropriate registers.

  Rescheduling is complete.

Channel Communication



- Soft Channel is a single memory location - initially empty.

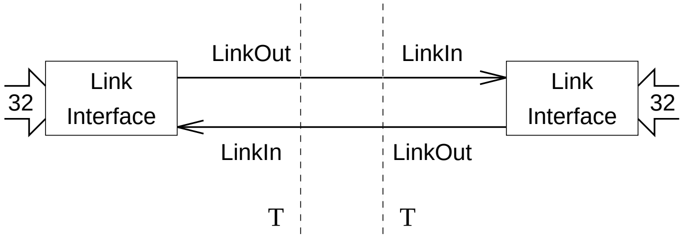
Channel Communication

- When **P1** attempts to access channel it is empty.
  - **P1**'s data pointer (from C) is stored in its own workspace.
  - **P1** is descheduled, its workspace pointer is placed in the channel - not added to active process queue.

- When **P2** attempts to access channel it contains **P1** workspace pointer.
  - **P2** uses this pointer to access **P1**'s data pointer and performs the transfer - whether it be read or write.
  - **P2** then adds **P1** to the back of the active process queue.
  - **P2** returns channel to empty state.

Links

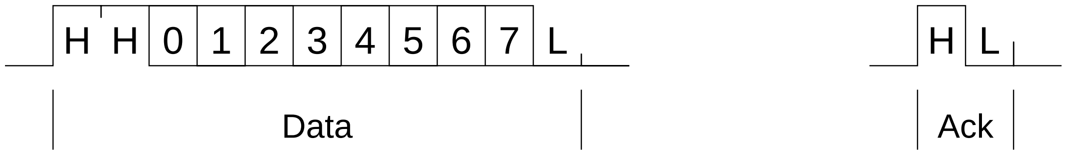
- T414 has 4 INMOS Serial Links



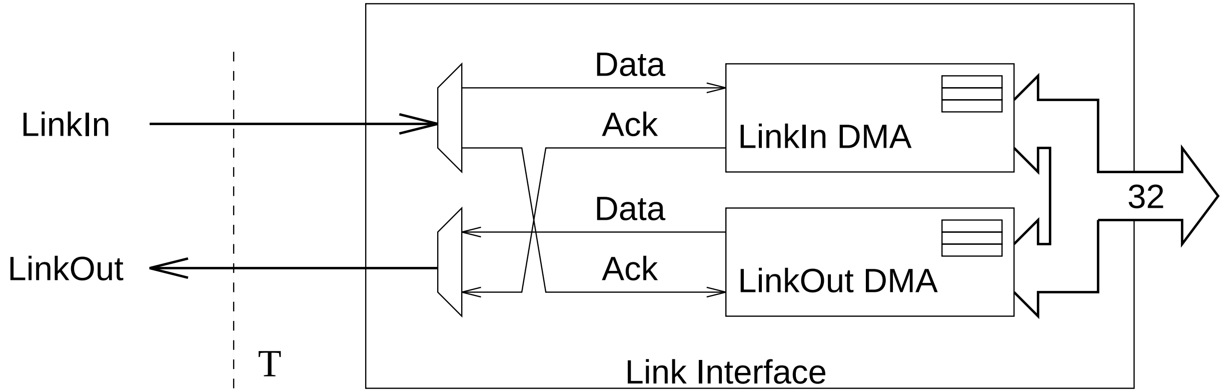
- Links bit serial and asynchronous.
- Links are bi-directional supporting:
  - LinkIn channel & LinkOut channel.
- Link interface is autonomous:
  - Uses DMA for data transfer.
  - Allows processing to continue.
  - All eight channels can be in use at the same time.

Links

All transfers are acknowledged.



Data and Ack are multiplexed onto a single line.



Links Transfer Rate

- Link speed is selectable:
  - 5MHz 10MHz 20MHz (for T414)
- T414 transmits Ack after it receives end of Data.
  - Achieves uni-directional transfer rate of 0.8 Mbytes/sec.
- T425 can transmit Ack as soon as it recognizes Data.
  - Thus Data and Ack can be overlapped.
  - Achieves uni-directional transfer rate of 1.8 Mbytes/sec.
  - Achieves bi-directional transfer rate of 2.4 Mbytes/sec.

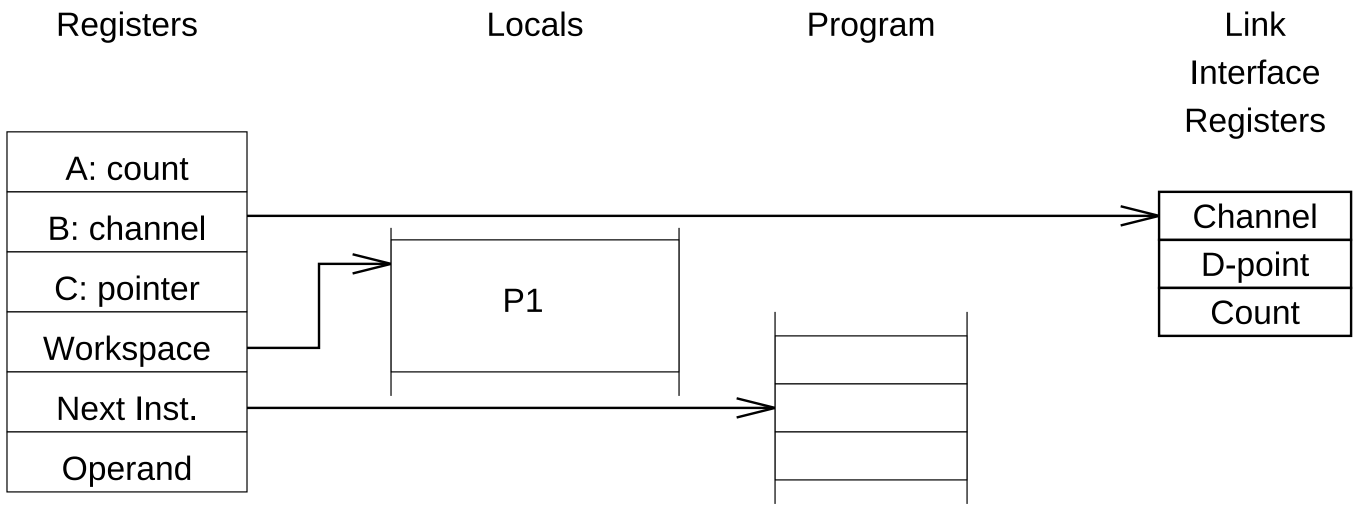
External Channel Communication

- The same instructions, `in` and `out`, are used for internal and external communications.
- Links are mapped onto the lowest eight memory locations:

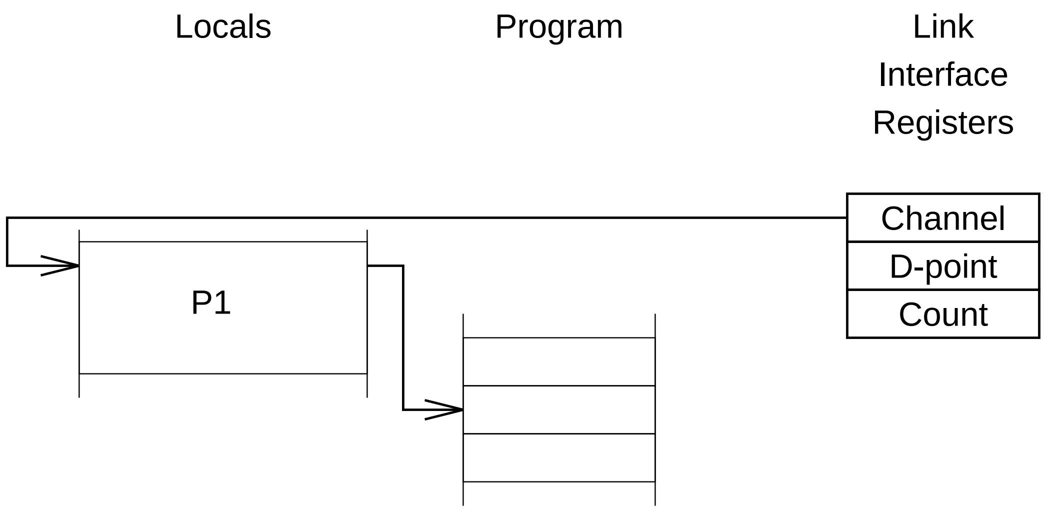
#08	Event
#07	Link 3 Input
#06	Link 2 Input
#05	Link 1 Input
#04	Link 0 Input
#03	Link 3 Output
#02	Link 2 Output
#01	Link 1 Output
#00	Link 0 Output

- The processor traps `in` and `out` to these locations and adjusts its behaviour accordingly.

External Channel Communication



- When process **P1** attempts to access the channel, it delegates the data transfer task to the link interface.
  - **P1**'s data pointer (from C) and data count (from A) are transferred to the link interface registers.



- **P1** is descheduled leaving its workspace pointer in the link Channel register.

- The other process on the other Transputer will do the same.
- Data is transferred between the two link interfaces.
  - Note that the receiving link interface will not acknowledge the first byte until the `in` instruction has been executed.
- When transfer is complete the separate processes are rescheduled.

High Priority Processes

- Two Linked Lists of Processes
- Low priority processes
  - Executed only when no high priority processes are active.
  - Execute until:
    - - - Completion.
    - - - Wait on communication or timer.
    - - - Second timeslice tick - every 1ms.
    - - - Ousted by high priority process.
- High priority processes.
  - Execute until:
    - - - Completion.
    - - - Wait on communication or timer.

Fast Interrupt

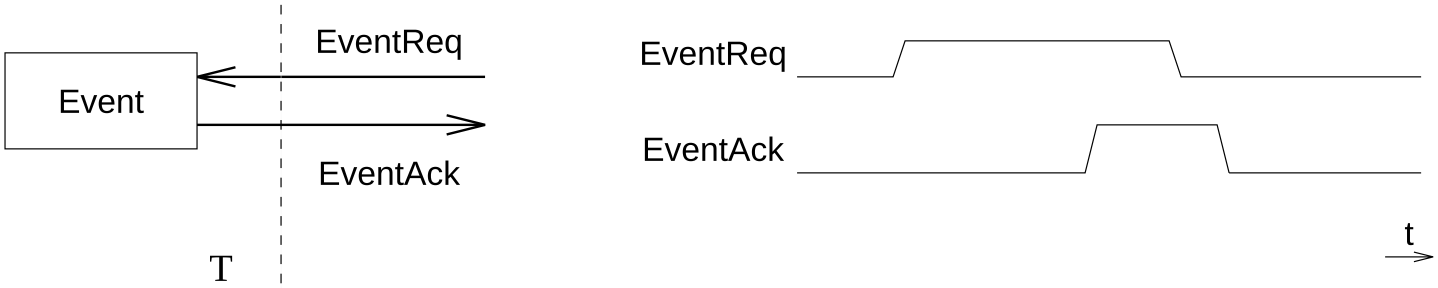
- High priority process waits on external communication.
- High priority process interrupts low priority process.
  - Wait for end of non-interruptible instruction.
  - Doesn't wait for end of expression evaluation.
  - Store State in memory (effectively memory mapped shadow registers).

#11	EregIntSaveLoc
#10	STATUSIntSaveLoc
#0F	AregIntSaveLoc
#0E	BregIntSaveLoc
#0D	CregIntSaveLoc
#0C	IptrIntSaveLoc
#0B	WdescIntSaveLoc

- Typical Latency 19 Processor Cycles.

Event Interface

- We have seen that external communication can generate interrupts.
- For more conventional interrupts we have the Event Interface.



- The event interface behaves like an input channel but conveys no data.

```
Event ? dummy    -- wait on EventReq signal.
```

Timers

- Two 32 bit Timers which tick periodically.
- Clock0
  - Accessible only to high priority processes.
  - Ticks once every microsecond.
- Clock1
  - Accessible only to low priority processes.
  - Ticks once every 64 microseconds.
- Timer can be interrogated or waited upon.
  - `Timer ? v            -- assign v := Timer.`
  - `Timer ? AFTER time   -- wait on Timer >= time.`
- Each timer supports an ordered queue of waiting processes.

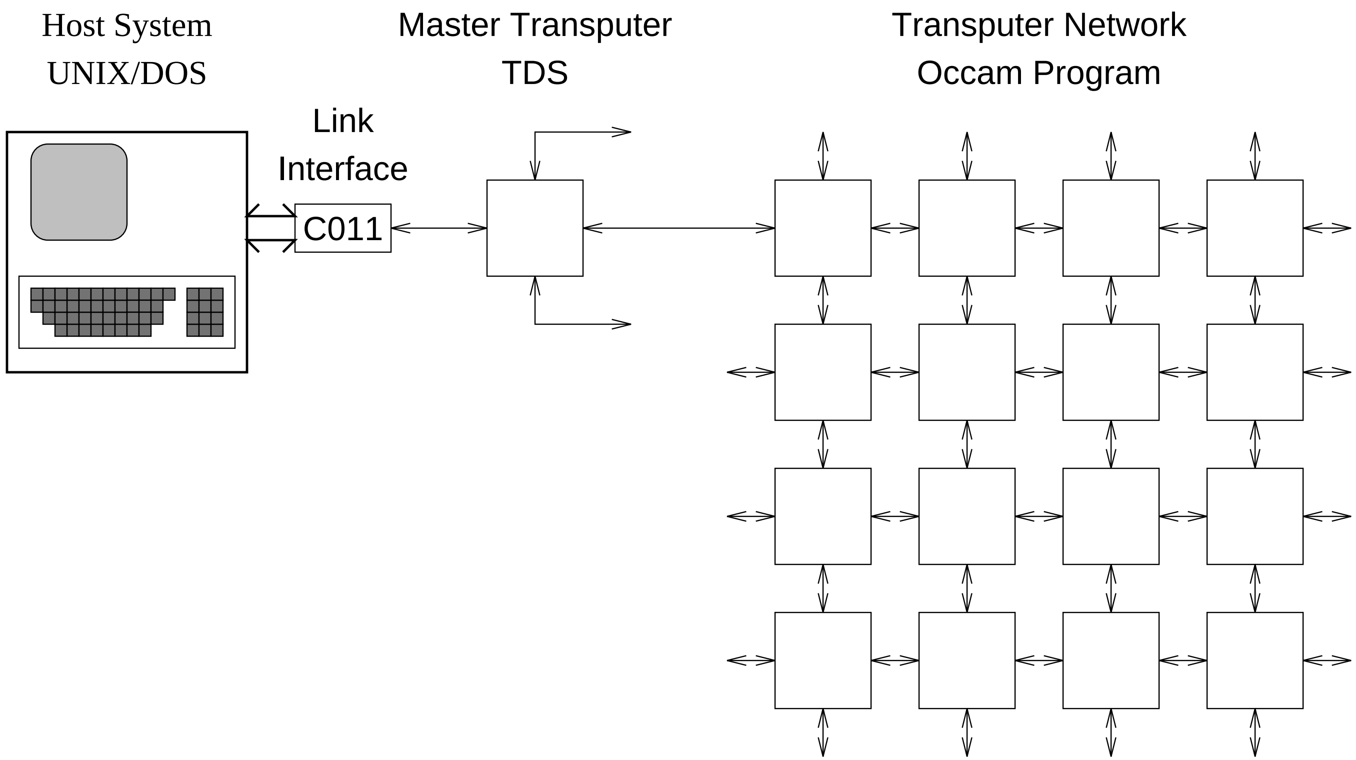
Boot from Link

Transputer behaviour on reset:

- BootFromRom = 1
  - Transputer starts executing code at address at top of memory.
- BootFromRom = 0
  - Transputer waits on data from any link.
  - Receives length data followed by program data from any link.
  - Places program data in RAM and executes it.

In this way we can load code from one Transputer to the next through a complex network. Frequently the code will be stored on a host machine, there is no need for any Transputer to have any ROM.

The Transputer

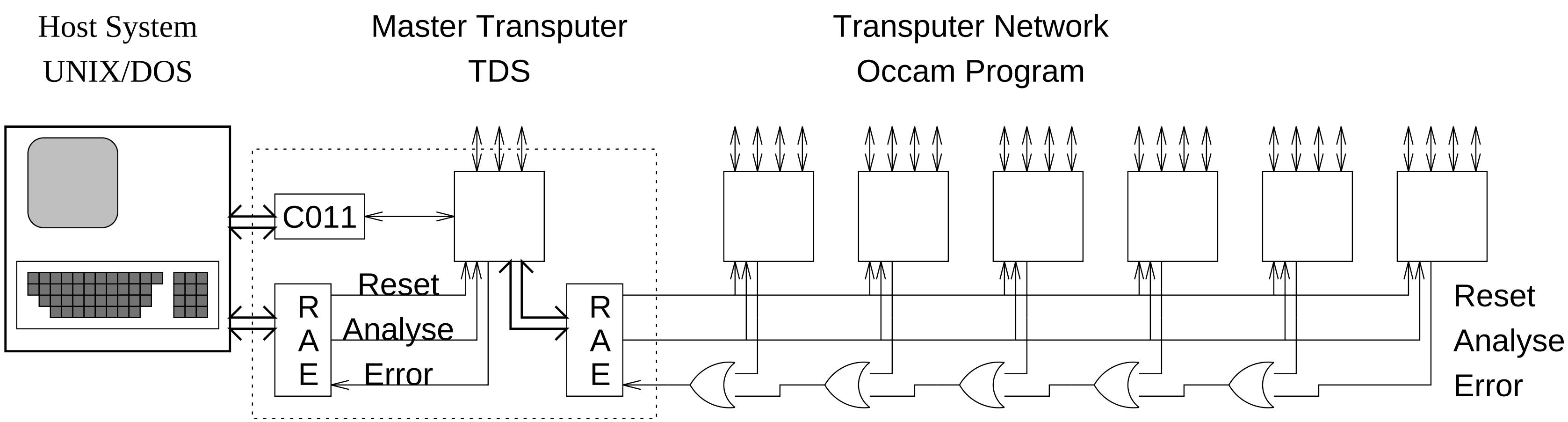


Simple Transputer System

- TDS is downloaded from the host system to the master Transputer.

- The Occam program is downloaded from master to network.

System Services

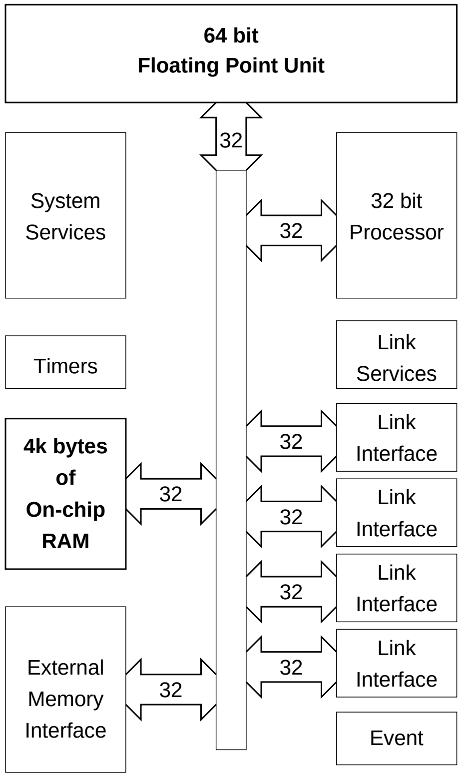


Reset Analyse & Error

- Host controls master with a memory mapped RAE interface.
- Master controls network with a similar interface.
- Errors are 'OR'ed to give single network error signal to master.

The Transputer

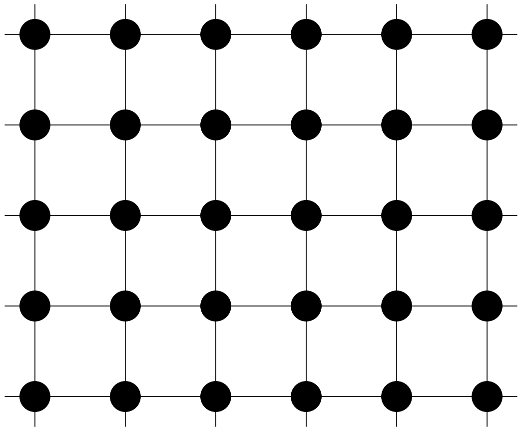
T800

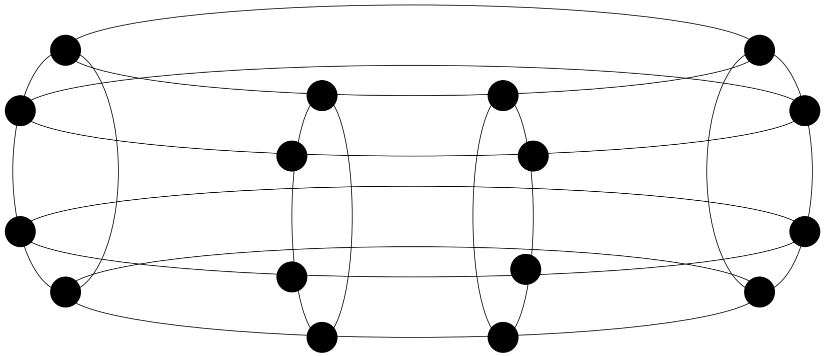


- 4 kbytes RAM (more compact)
- 64 bit Floating Point Unit - Extended Instruction Set.

Networks

- Transputer has a fixed *valency* of 4.
- Hence the choice of networks is limited.



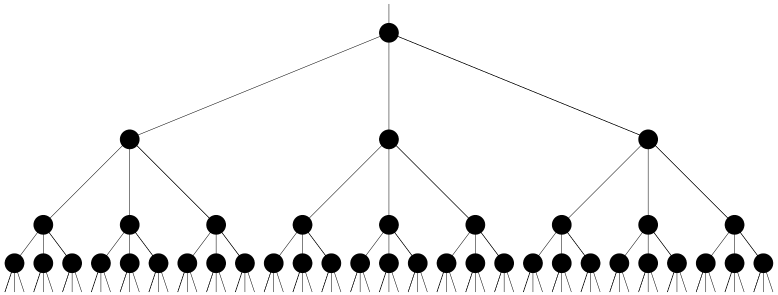


- Grid is natural choice for exploiting data parallelism over arrays.
- We partition data to exploit *Geometric Parallelism*.

N.B. We must usually break one link for communication with a master Transputer if we use a 2D Torus.

Networks

- Ternary Tree Structure:

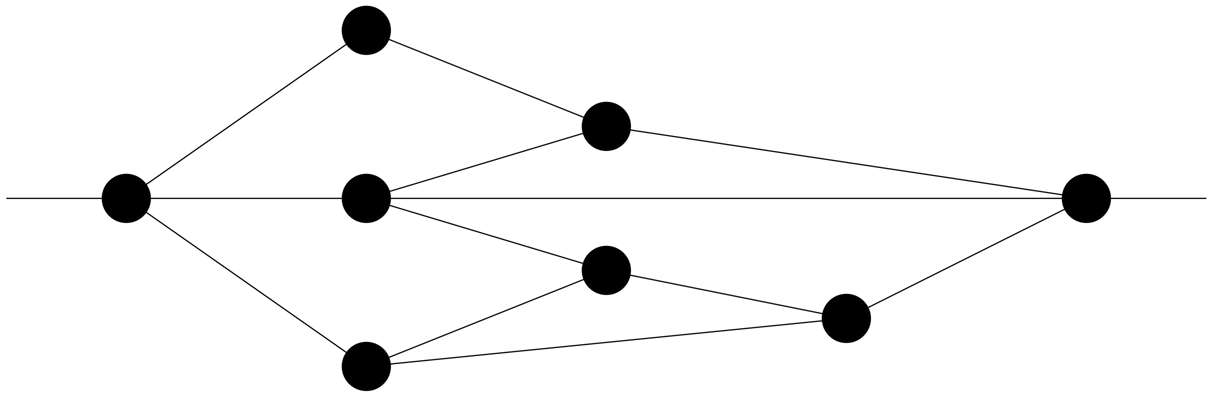


- Used in processor farms:
  - Discrete bundles of processing are farmed out by the master processor to any available processor.
  - The processes may be identical, but must be independent (no inter-process communication).

N.B. Ternary trees are seldom used in packet routing systems because of the root *hot spot*.

Networks

- Algorithm Mapping.



- The processors are arranged to match the data flow of the algorithm.
- The code rather than the data is distributed across the processors.
- The result is a pipeline type multiprocessor.

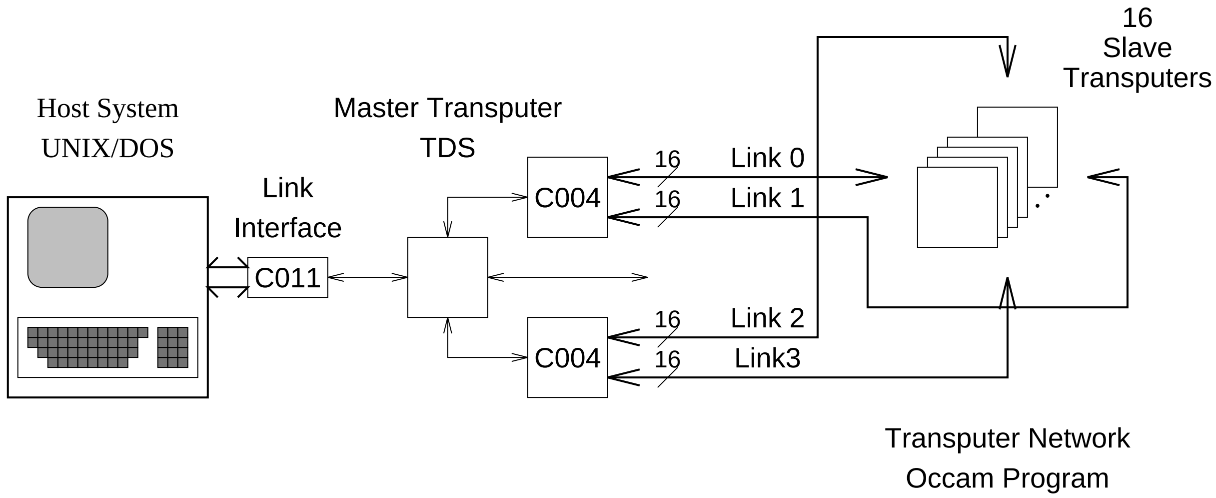
Frequently used in embedded applications and signal processing.

Reconfigurable Networks

Given that Software Packet Routing is relatively expensive in Transputer networks and manual rewiring is not a serious option we need a reconfigurable Transputer network.

- C004 Programmable Link Switch.
  - 32x32 crossbar switch for 32 INMOS Serial links.
  - Supports any *one to one* link mapping.
  - Controlled by another INMOS Serial link.

Reconfigurable System



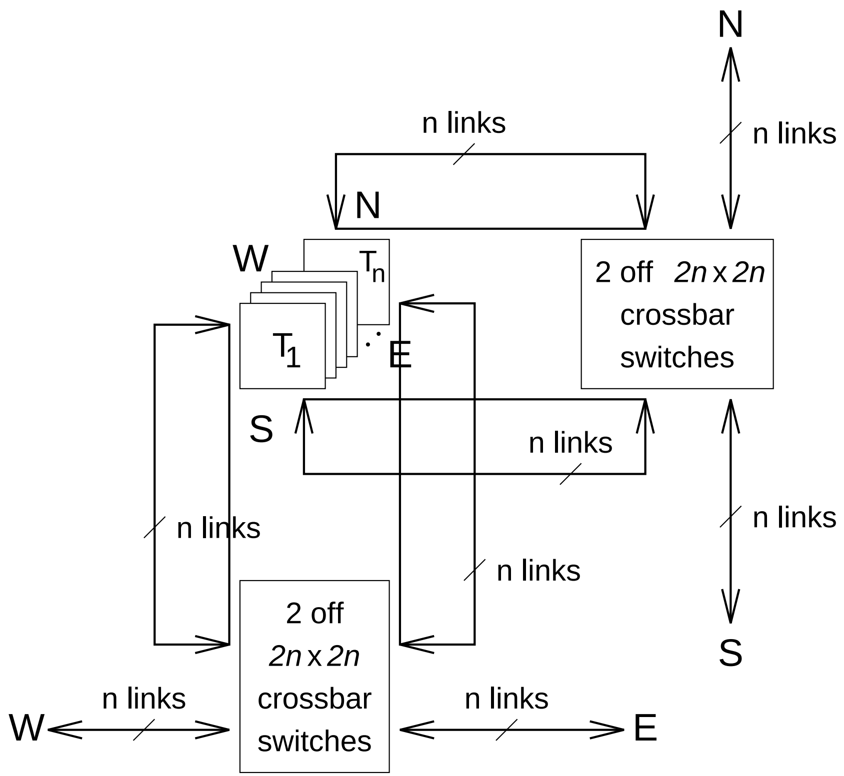
- Master Transputer controls 2 C004 link switches.
- We can connect any link 0 or 1 to any link 0 or 1.
- We can connect any link 2 or 3 to any link 2 or 3.

N.B. We must connect master boot link manually - no C004 links left.

Supernode Project

ESPRIT 1085

- Supernode of n Transputers:

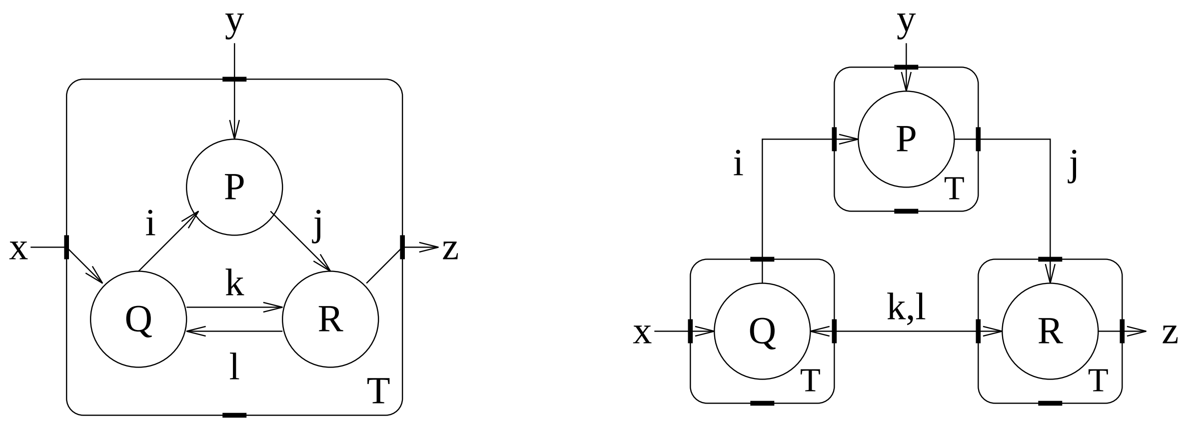


The Transputer

- Supernode Connectivity
  - We can connect any link N to any link S.
  - We can connect any link E to any link W.
  - It can be shown that we can create any graph of n labeled Transputers.
- Automatic configuration from algorithm topology!
- Supernode has $4n$ external links and n processors.
  - Processing to communications ratio is unchanged.
- For bigger systems we connect multiple supernodes.

Occam Implementation on Transputers

A single Occam program should run on one Transputer or on many Transputers.

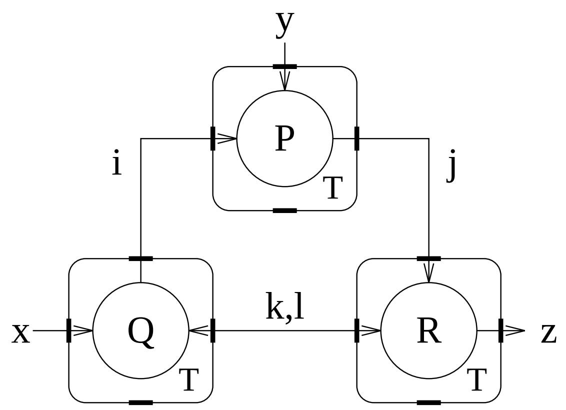


- This allows the debugging of programs in the comparatively simple environment of a single Transputer, before porting to multiple Transputers.

Occam Implementation on Transputers

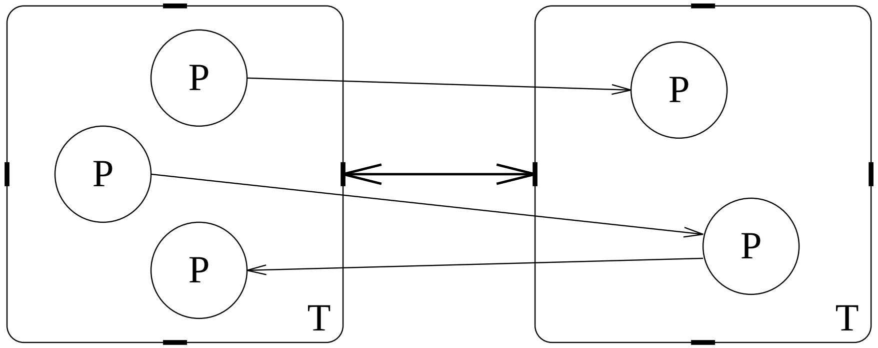
```
PLACED PAR
  PROCESSOR 0 T4
    PLACE y AT LinkIn1 :
    PLACE i AT LinkIn0 :
    PLACE j AT LinkOut2 :
    P(y,i,j)
  PROCESSOR 1 T4
    PLACE x AT LinkIn0 :
    PLACE l AT LinkIn2 :
    PLACE i AT LinkOut1 :
    PLACE k AT LinkOut2 :
    Q(x,l,i,k)
  PROCESSOR 2 T4
    PLACE j AT LinkIn1 :
    PLACE k AT LinkIn0 :
    PLACE z AT LinkOut2 :
    PLACE l AT LinkOut0 :
    R(j,k,z,l)
```

Configuration

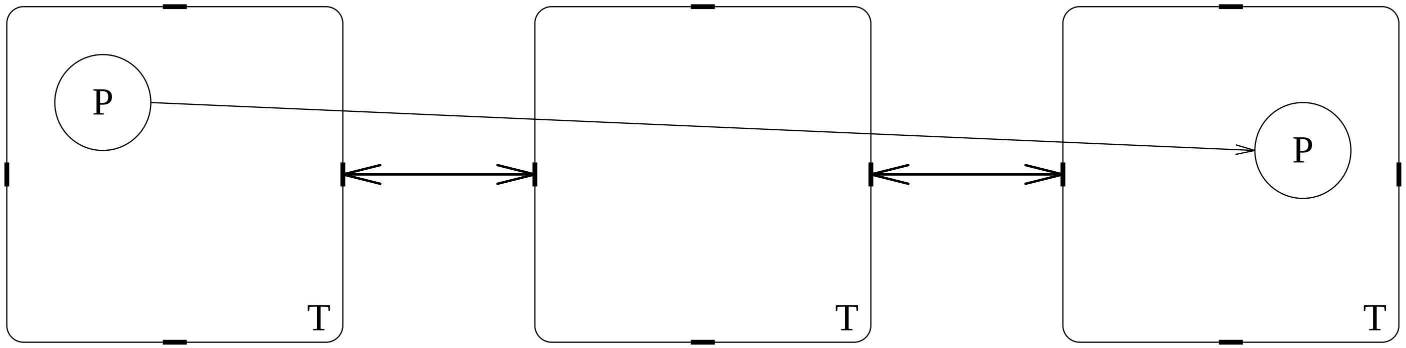


Restrictions of Occam Implementation on Transputers

- Only two Occam channels (one in each direction) can be mapped onto a Transputer link.



- Communicating processes must be on adjacent Transputers. Occam doesn't support implicit message forwarding



Restrictions of Occam Implementation on Transputers

Process Placement:

- Only *Separately Compiled* procedures can be used in `PLACED PAR`.
  - This is a reasonable restriction as it prevents common variable access between these procedures.
- Configuration is only permitted at the top level of the program.
  - Thus there is no facility for `PLACED PAR` within `SEQ`.
  - These *Remote Procedure Calls* would be beneficial if difficult to implement.

```
SEQ
  IN ? A
  IN ? B
  C := A * B
  PAR
    P(C)
    Q(C)
```

PUMA Project

Parallel Universal Message-passing Architectures

Southampton CCG and Others

- This project has looked at re-writing Occam for a Transputer network with hardware or software message routing.
- The project has also addressed the problems of remote procedure calls.
- The result is a *Distributed Occam* which is closer to the ideal for an MIMD machine.