

Topics in computer architecture

Utah data-driven dataflow architecture

P.J. Drongowski
SandSoftwareSound.net

von Neumann organization

- Most prevalent computer architecture
- Single central processing unit (CPU)
 - Central clock
 - CPU executes instructions in step with clock
 - CPU is the system master
 - Control and communications center for entire system
- Primary memory
 - Linearly addressable array of binary *words*
 - Each word is a fixed size
- CPU - memory *channel*
 - Set of parallel information paths
 - Exchange control signals, addresses, data
 - Sequential operation
 - Set control for read, send address, receive data
 - Set control for write, send address and data
 - Address is an index into the linear memory array
- Programming
 - Instructions and data
 - Binary codes
 - Stored in primary memory
 - Elemental operations on elemental operands
 - Program counter sequences instructions
 - (Un)conditional branches for loops and decisions
 - Arithmetic is parallel binary



Some exceptions

- Burroughs B5500, etc. stack addressable storage
- Burroughs B1800 does not have fixed logical word length
- IBM 370 series has special I/O channels
- CDC Star and Cray have multiple arithmetic processors

Self-consistency (harmony)

- "Recursive machines and computing technology,"
V.M. Glushkov, et al., IFIP Information Processing '74,
North Holland Publishing, 1974.
- von Neumann principles are self-consistent (re-enforcing)
 - Central clock and sequential transfers through channel
 - PC steps sequentially through linear memory array
 - Fixed word width and parallel arithmetic
- Revision of one or two principles will destroy harmony
- Intuitively self-consistency is a "good fit" of principles

Improving von Neumann speed

- Use faster components
 - Increase speed of CPU, memory, channel
 - Bounded by physical limitations
- Exploit locality
 - Keep operands in local cache or general registers
 - Reduces number of channel operations
- Concurrency
 - Increase number of processor - memory channels
 - ◇ Multiple address and data streams
 - ◇ Typically found in vector supercomputers
 - Increase input/output concurrency
 - ◇ Perform computation and I/O simultaneously
 - ◇ Add "direct memory access" channel
 - ◇ Increases speed of bulk data transfers
 - Pipelining
 - ◇ Degree of concurrency limited by number of stages
 - ◇ Pipe disrupted by change in control flow
 - ◇ Disruptions due to data dependency hazards
- Synchronization and arbitration must be provided
 - ◇ Processes (and devices) must co-ordinate
 - ◇ Interrupts are a synchronization mechanism
 - ◇ Would like to virtualize devices into OS processes

Further problems

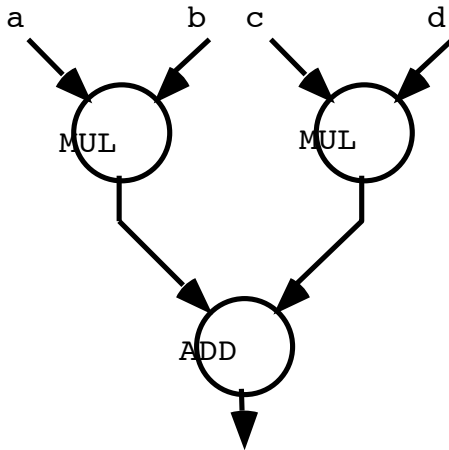
- Gap between machine and HL programming language
 - ◊ Implement features of HLL in hardware
 - ◊ Semantic clash
 - ◊ Recursion is usually expensive
- Linear organization does not fit application data model
 - ◊ Leads to construction of sophisticated storage managers
- Sequential centralized control
 - ◊ Complicates operating system design
 - ◊ Processes must be forced into single instruction stream
 - ◊ Nondeterministic effects must be suppressed
 - ◊ Inhibits extensibility
- Throughput depends on component speed
 - ◊ Wire delay is now dominant
 - ◊ Compounded by chip partition

Recursive machine principles

- No limit on complexity of operators and operands
 - ◊ Operands may be numbers, strings, vectors, whatever
 - ◊ New types recursively mapped to primitive structures
 - ◊ Implies a "procedural" data model (i.e. executable data)
- Program elements executed when operands are available
 - ◊ Use memory associativity to detect "ready" elements
- Memory structure is reprogrammable
 - ◊ Convert data and programs to internal form
 - ◊ New types and machines are recursively definable
 - ◊ One level implemented in terms of other levels
- No limit to number of machine elements
 - ◊ Performance tuning by adding and removing elements
 - ◊ Replace broken machines dynamically
- Organization is flexible, re-programmable
 - ◊ Communication switches needed between RC elements
 - ◊ Switches are programmable
 - ◊ Note: An RC element is a processor-memory pair

Data driven dataflow computing

- Initiate activity when necessary information is available
- Example: $(a \perp b) + (c \perp d)$



- DDN dataflow program is a cyclic *bipartite* directed graph
 - Graph vertices can be divided into two subsets
 - All edges in graph connect only the two subsets
 - No edge connects two vertices within same subset
- Cell
 - Graph node (vertex)
 - Operation to be performed
 - Operation determines number of inputs and outputs
 - Operation determines the type of arguments accepted
- Arc
 - Connects a producer cell to a consumer cell
 - Arc is a FIFO queue of (conceptually) infinite length
 - Carry messages called *tokens*
 - Message can be almost any data structure
- Firing rule
 - Set of inputs that govern firing is a *firing set*
 - Cell is *fireable* when firing set is satisfied
 - May fire when data is available at all inputs in firing set
 - To fire, remove item from head of each input queue
 - Remove tokens only from firing set
 - Compute the result
 - Send result item on outgoing arc(s)

Characteristics

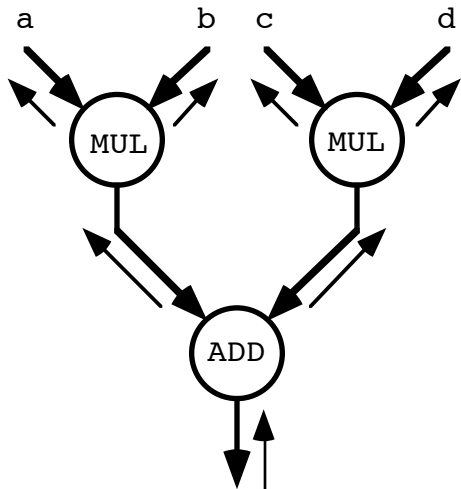
- Concurrency
 - Horizontal (MUL's are independent)
 - Pipelined (due to FIFO queues between cells)
 - May be exploited if multiple functional units available
 - *Maximally parallel*
- Functional program
 - Data is a value not a storage location (state)
 - Not history dependent
 - Result depends solely on input values
 - No side-effects making verification easier
- Nonprocedural program
 - Execution order not strictly dependent on lexical order
 - Order is dependent on functional relationships
 - Can be described denotationally
- von Neumann programming
 - Control driven; rigidly sequences
 - Uses explicit addresses
 - Program directly manipulates memory state
 - Tight binding between data and memory location

```
move      a, register-0  
multiply b, register-0  
move      c, register-1  
multiply d, register-1  
add       register-1, register-0  
move      register-0, result
```

- Data driven programming
 - Functional and nonprocedural
 - No memory cells or addresses
 - Memory is embedded in store semantics of arcs
 - Only data dependency affects sequencing

Demand driven dataflow

- Two types of messages are sent over arcs
 - *Demands* flow from outputs back to inputs
 - *Result values* flow forward in data driven



- Advantages
 - Avoids unnecessary computation
 - Example: *if - then - else*
 - Data driven computes all expressions
 - Either *then* or *else* part will be discarded
 - Demand driven computes only needed value
 - Conserves physical computing resources
 - Can "represent" infinite data structures (objects)
 - Provide a net to compute an infinite stream
 - Just demand next value in stream when needed
 - Lazy evaluation
 - Postpone generation of item until needed
 - Potentially an item may never be needed
 - Again, this conserves physical computing resources
 - Example: Lazy cons (LISP)
 - cons joins two subtrees into one tree
 - Often one subtree is immediately discarded
 - Lazy cons waits until needed subtree is selected
- Disadvantages
 - Twice as many messages must be sent
 - Data driven - demand *always* exists at cell outputs

Co-ordination of parallel processing

- Programs must co-ordinate exchange of information
 - Program execution is explicitly sequenced
 - System level behavior is asynchronous
 - Mixture causes a conceptual mismatch
 - Example: Interrupt mechanism vs. virtual device process
- Dataflow naturally co-ordinates processes
- Logical and physical change of context is not needed

Centralized systems

- Current multiprocessors engender "global state" approach
- All processes share common global state
- Runs counter to "divide and conquer" software design
- Centralized systems often use a common clock
- Clock must be distributed to system components
- Clock skew is sensitive to distance between components
- Longer propagation time means slower performance
- Novel physical packaging has its limitations
- Expansion difficult due to synchronous time constraints

Decentralization and buildability

- Extensibility
 - ◊ Incrementally add more processing modules
 - ◊ Addition should not require change to operating system
 - ◊ Plug in new modules without "tuning"
 - ◊ Extensions available in small quanta
- Characteristics of a fully distributed system
 - ◊ No module can determine total system state
 - ◊ No module can enforce simultaneity in other modules
- Key is decentralization and asynchronous operation

Implementation technology

- von Neumann era
 - Vacuum tube technology
 - Minimize components
 - Active circuitry is expensive
 - Delay of circuitry dominated; wire delay insignificant
 - "Hardware is expensive, software is cheap"
- Economic principles not valid anymore

VLSI technology

- Production cost for custom chip is \$80,000 to \$300,000
- Production goal
 - 250,000 parts at \$7 to \$10 per part
 - Part must be used in large volume
 - Keep number of distinct part types low
- Speed (dominated by signal drive into conducting path)
 - Delay proportional to signal path capacitance
 - Off-chip 100pF versus 1pF on-chip fan-out
 - Perform as much function as possible on-chip
 - Keep on-chip wires short
 - Exploit physical locality!
- Number of pins
 - Manufacturing cost is roughly linear with pin count
 - Can decrease times by time division multiplexing
 - Performance penalty is incurred by multiplexing
 - Disadvantages of increased pin count
 - Decreased yield due to bonding problems
 - More silicon area for connection pads and drivers
- Other advantages
 - Increased system reliability due to reduced part count
 - Decreased (system level) power consumption
 - Lower maintenance cost through chip replacement
- Try to achieve very high logic to pin ratio

Data Driven Machine 1 (DDM1)

- Alan L. Davis, University of Utah
- Low level machine language (DDN's)
- Goals and principles
 - General purpose computing
 - Improve performance through concurrency
 - Potential concurrency in program must be very high
 - Large number of function units must be available
 - DDN's contain "fine grain" concurrency
 - Compatible with VLSI technology
 - Use large number of identical processing sites
 - Each site should ideally be one part type
 - Extensible
 - Increase performance by adding plug-in modules
 - Distributed, asynchronous control
 - Hierarchical organization
 - Simplifies conceptual complexity at each level
 - Verification by induction for uniform systems
 - Natural superior-inferior relationship
 - Helps resolve resource management
 - Leads to a physical tree of processing modules
 - Logical extensibility
 - Recursive hierarchy
 - Structure is same at all levels of hierarchy
 - Implies that same module can be used at all levels
 - Physical recursion ends at logically deepest resources
 - Fan-out from level to execute independent operations
 - Depth used to pipeline operations

Architecture

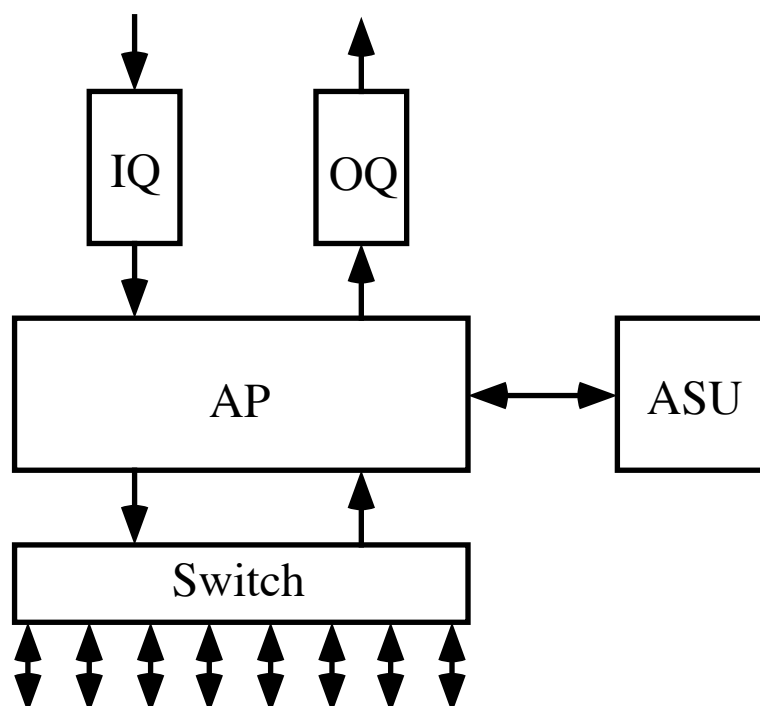
- Architecture is recursively organized PSE's
- Processor - store element (PSE)
 - Fundamental computational unit
 - Substructure
 - Processor module (P)
 - Atomic processor (AP)
 - Eight PSE at next lower level of hierarchy
 - Local storage module (S)
 - PSE can execute any machine language program
- Atomic processor module (AP)
 - No substructure (i.e., no lower PSE's)
 - Bottom of hierarchy consists solely of AP's
- Atomic storage unit (ASU)
 - No substructure
 - Size of store is arbitrary
 - Higher level stores should be slower and larger
 - Ability of parent should be greater than sum of children

Hierarchy

- All AP's are identical regardless of level
- Higher levels are more powerful
 - Higher levels have more substructure
 - Implies more internal concurrent processing capability
- Non-recursive structure
 - Physical tree of PSE's
 - Possibility of eight children per PSE
 - Leaf nodes are just atomic processors and storage units
- Depth of the tree is arbitrary; fan-out is fixed at eight
- Extend machine by adding more levels
- Each PSE is asynchronous so no tuning is needed
- Communication is direct; topology is fixed

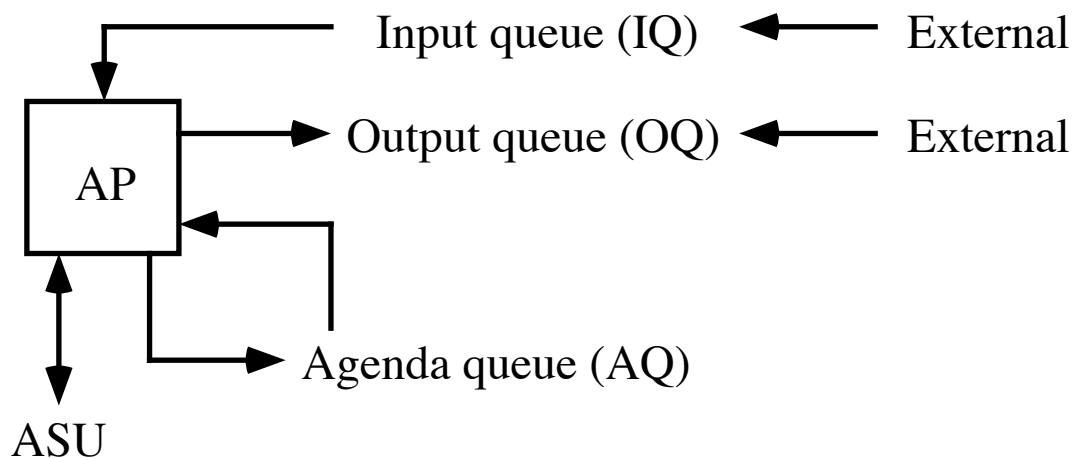
PSE structure

- Inter-PSE links are direct
 - PSE's exchange variable length character string msg's
 - Upward messages are switched like an arbiter
 - Downward messages contain header for distribution
 - Hardware switch is independent unit
- Serial communications
 - More applicable for VLSI implementation (pin count)
 - Messages are next fixed width information units
 - Permits substitution of other module implementations
 - Local performance is lost
 - Try to regain performance through high parallelism
- Physical queues placed between levels of PSE's
 - Facilitate pipelining
 - Increase physical module independence
 - Sender must only wait when queue is full
 - Choose queue size for average message length
- System will not deadlock
 - Strict hierarchical control
 - Restricted process structure
- Physical link characteristics
 - Two wire request-acknowledge control link
 - Four wire character-width data bus



Atomic processor

- DDN to be loaded arrives as a message via the IQ
- Data items also arrive via the IQ
 - Data items has two fields
 - Destination identifier
 - Item value
 - AP interprets the destination identifier
 - If resident in ASU, route to local cell
 - Otherwise, pass to next switch
- If destination cell becomes fireable, fire it
 - Use incoming data item as needed
 - Avoids unnecessary store operation in ASU
- AP produces a result with two fields
 - Destination of the result message
 - Result value
- If result destination is resident, then send it to the AQ
- Otherwise, send the result message to the OQ
- Eventually, process the AQ (agenda queue)
 - Process like arriving messages from the IQ
 - When AQ is empty, then wait on the IQ
 - IQ before AQ strategy
 - Do as much locally as possible before new work



Atomic storage unit

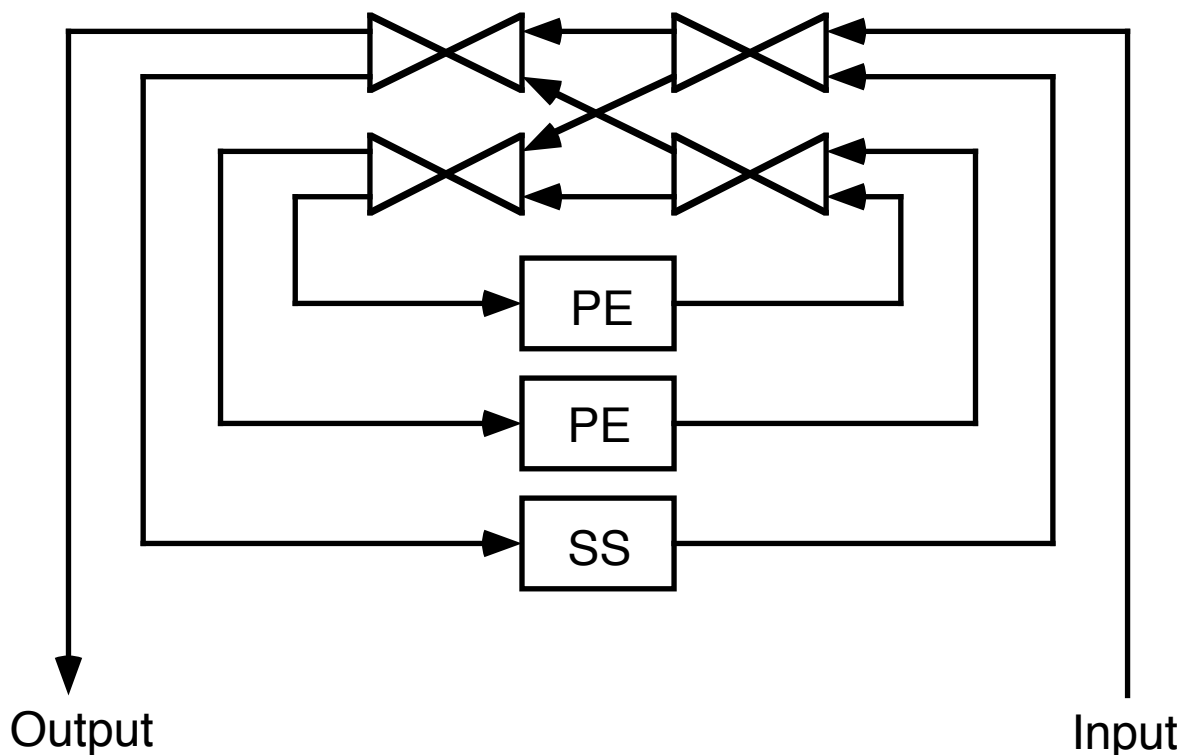
- Use the Barton Storage Model (TSM)
- File system with the following commands
 - Initialize
 - Skip - cursor skips over the current field under cursor
 - Insert - inserts character or file before current position
 - Delete - delete the character or file at cursor position
 - Assign - assigns character (file) to current character (file)
 - Read - reads the character or file selected by cursor
 - Head - position to leading '(' of parent field
 - Shift - increment the cursor
 - AIndex
 - Absolute index
 - Index from first character in the store
 - RIndex
 - Relative index
 - Index from current cursor position
- ASU implementation
 - 4K by 4 bit character store using RAM chips
 - Data exchanges use four-cycle self-timed handshake
 - "Mapping unit" can speed up indexing operations

Comments on DDM1

- DDM1 has lots of internal concurrency
- Internal state is impossible to save and restore
- Transfer fields, not characters
- Vector processing is sensitive to number representation
- ASU design was not easily extensible in size
- Fixed tree structure causes load imbalance
- Not enough empirical data on decomposition overhead
- Failure of PSE will cause failure of subtree
- Certain SP-graphs cause less than full pipelining

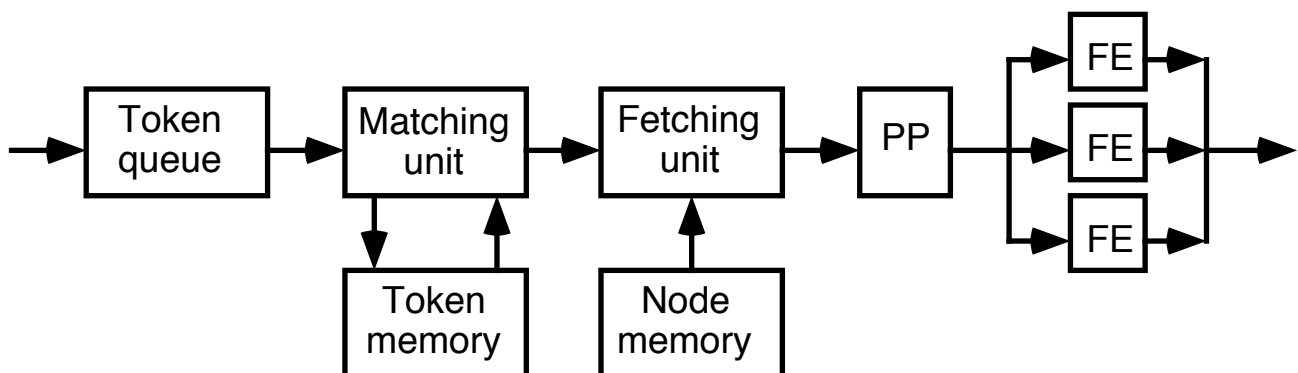
Manchester dataflow machine

- "The Manchester prototype dataflow computer," J.R. Gurd, C.C. Kirkham, and I. Watson, CACM, Vol. 28, No. 1, January 1985, pg. 34 - 52
- History
 - Project started in 1976
 - Prototype processing element completed in 1981
 - Constructed 1 million element structure store in 1986
- Characteristics
 - Each processing element (PE) is a ring-like structure
 - Machine is expanded by "layering" PE rings
 - PE's intercommunicate via an exchange switch
 - Successive levels of distribution, buffering, arbitration
 - Tokens contain routing information in name field
 - Buffering mitigates effects of message interference
 - Allows routing around fault processing elements
 - Exchange switch also provides external I/O interfaces



Processing element (PE)

- Each processing element is a four stage pipeline
- Units are internally synchronous
- Inter-unit communication is asynchronous
- All datapaths are parallel
 - Sizes of tokens and packets are fixed
 - Packet is 166 bits wide!
- Token queue
 - Provides rate balancing
 - FIFO buffer containing up to 32K tokens
- Matching unit
 - Matches incoming tokens for dyadic operations
 - Efficient implementation is crucial
 - Sparse VM with <destination,tag> pair as address
 - Hardware uses a hashing mechanism (vs. associative)
 - If address cell is empty (match fails), store token
 - If cell contains token, send both tokens to fetching unit
- Fetching unit
 - Combines tokens with node info into executable packet
 - Node can specify one or two destinations for result
 - Prototype accomodates up to 64K nodes
- Functional unit
 - Preprocessor (PP) interprets tag-handling instructions
 - Functional elements (FE) are microprogrammed
 - Instruction times: 3 μ s min, 6 μ s ave, 30 μ s max
 - Throughput rate is irregular (need rate balancing)



Data structures

- A data structure may be transmitted over an arc
- Transmission is one element per token at a time
- Different elements are distinguished by index field in tag
- Elements can be produced and accessed in any order
- Allows concurrent processing of structure elements
- *Streams*
 - Elements are produced and processed in order
 - *Non-strict*
 - Elements can be read before producing complete stream
 - Use tag manipulation to gain efficiency

Structure store (SS)

- Structure copying useful only for small data structures
- Fixed-size structures can be stored in *structure store*
- *Create-structure* operation
 - Reserves memory area
 - Returns pointer to structure
- *Write-element* operation
 - Stores value part of a token
 - Tag and destination are not stored
- *Read-element* operation
 - Returns value of an element if it is available
 - Otherwise (not available)
 - Append request to list of pending read requests
 - Fulfill request when the element is written
 - Allows storage of non-strict structures
- Garbage collection (via reference counters)