Multithreaded architectures

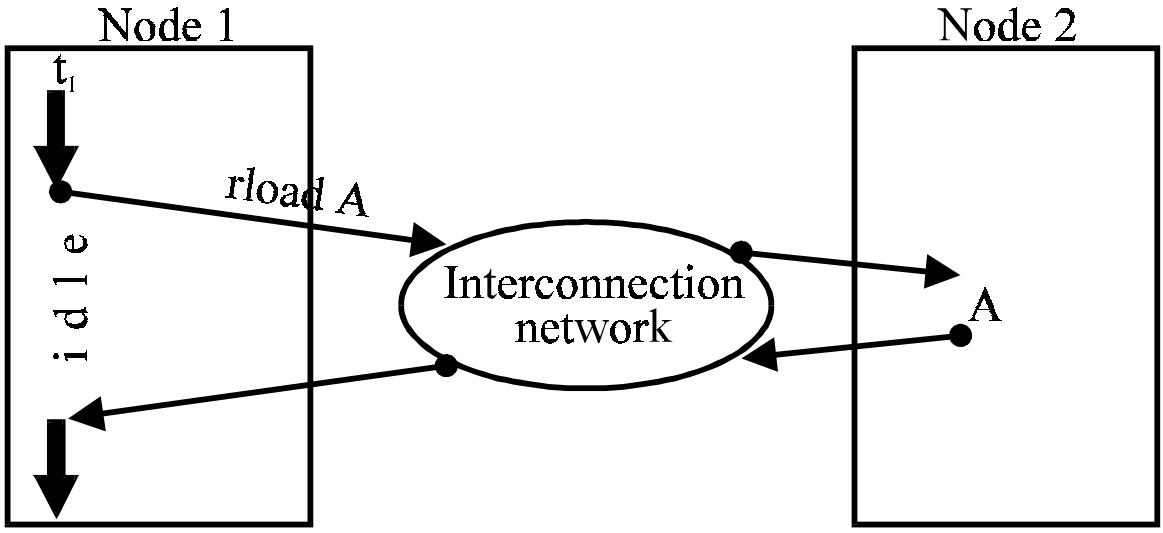
- **Two fundamental issues in parallel systems**

  1. Tolerate and hide latency of remote loads

     predictable latency (distance between nodes)

  2. Tolerate and hide idling due to synchronisation

     unpredictable latency



- **Solutions:**

  1. Cache memory

  2. Prefetching

  3. ***Threads and fast context switching:*** the only solution to both issues

Multithreaded architectures

Principles of multithreading

- **Goal: keep processor utilization as high as possible**

$$U = \frac{P}{T} = \frac{P}{P + I + S}$$

U = processor utilization

P = processing time

T = total time

I = idling

S = context switching

- **Objective of multithreading technique:**

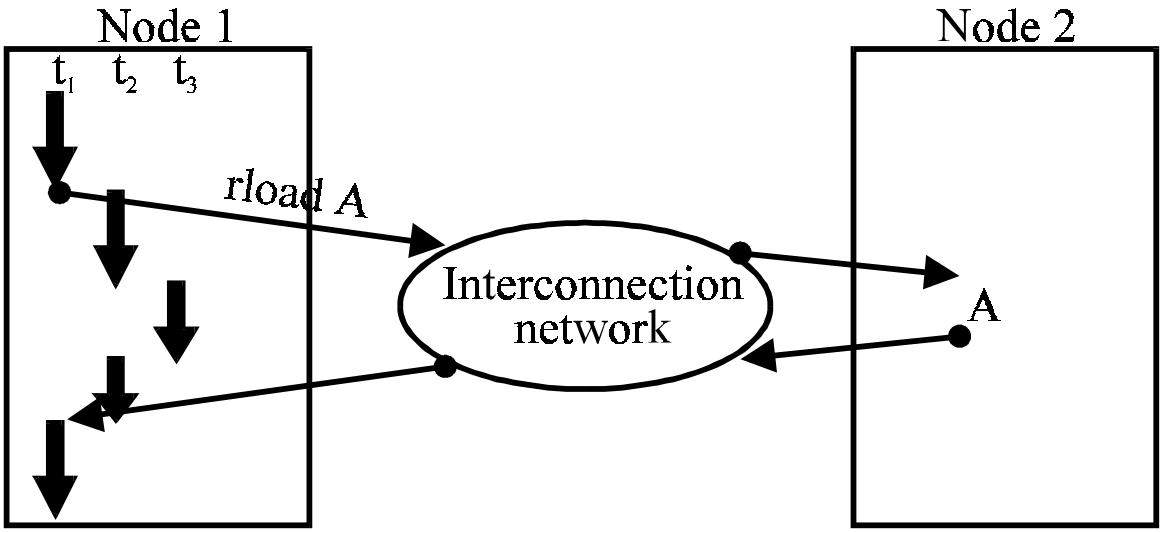
reduce I without increasing S

Conditions:

1. S<I
2. large set of threads

Multithreaded architectures

The remote load problem again



- By starting new threads (fast context switch) the processor can be held busy
- It can tolerate predictable and unpredictable latencies as well

Multithreaded architectures

Design space of multithreaded architectures

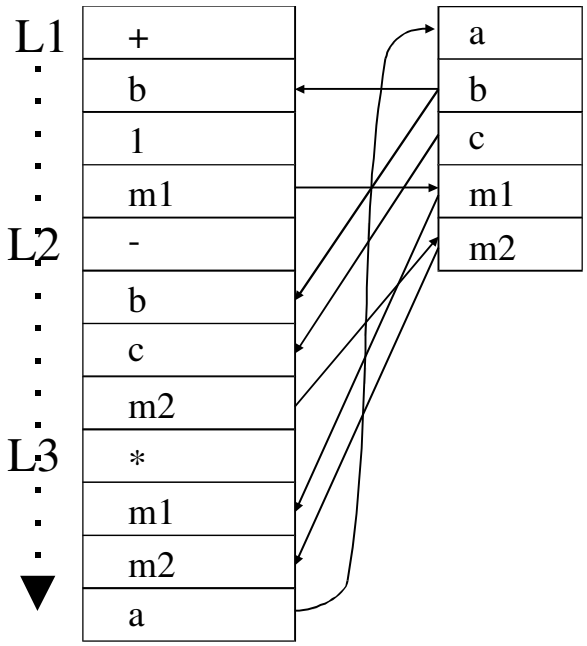
1. Computational model
   a. von Neumann (squential control flow)
   b. Hybrid von Neumann/dataflow
      - parallel control flow based on control operators
      - parallel control flow based on control tokens
2. Granularity
   a. fine-grain
   b. coarse-grain
3. Memory organization
   a. physical shared memory
   b. distributed shared memory
   c. cache-coherent distributed shared memory
4. Number of threads per processor
   a. small (4-10)
   b. middle (10-100)
   c. large (over 100)

Multithreaded architectures

Control flow models

1. Sequential
   - control is based on the textual order of instructions
   - or by explicit GOTO, CALL, JUMP, etc. instructions
   - hardware support: program counter

Example: compute a=(b+1)*(b-c)

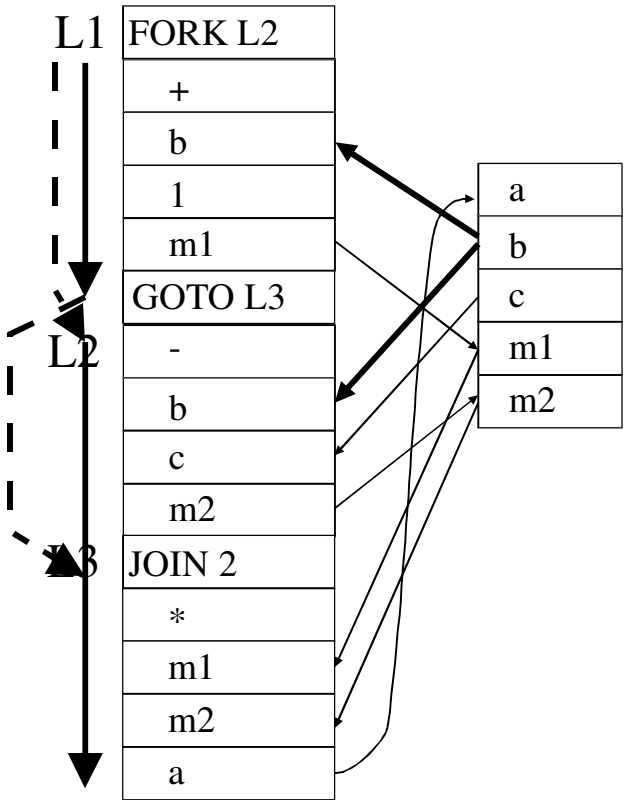


Multithreaded architectures

Control flow models

2. Parallel based on control operators
   - explicit creation of parallel threads (FORK/JOIN)
   - synchronisation on reusable memory cells is needed

Example: compute a=(b+1)*(b-c)

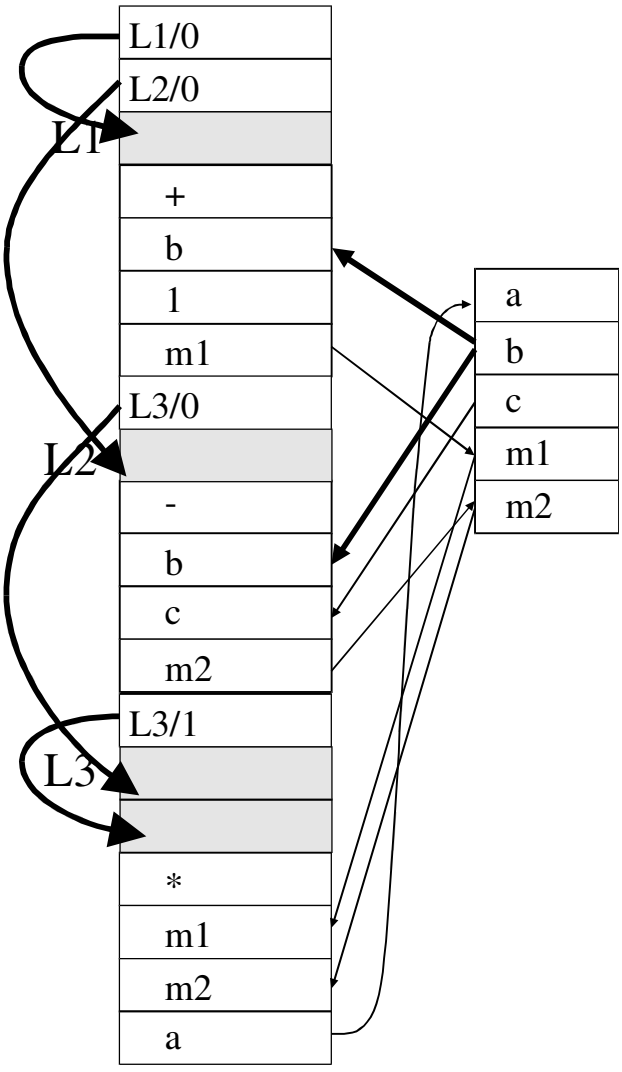


Multithreaded architectures

Control flow models

3. Parallel based on control tokens
   - each instruction explicitly specifies its successor
   - an instruction is executed when it has received the required control tokens

Example: compute a=(b+1)*(b-c)

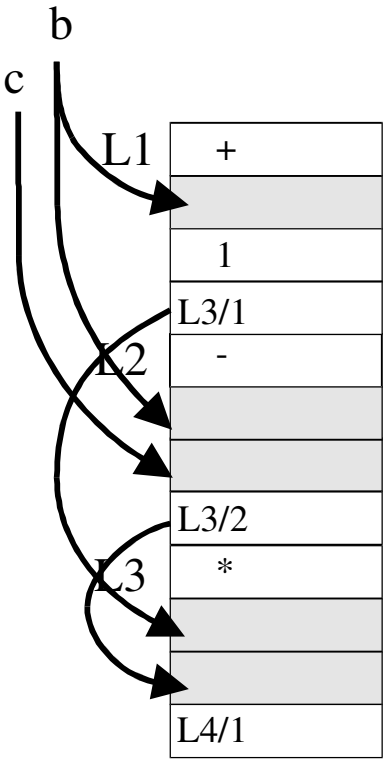


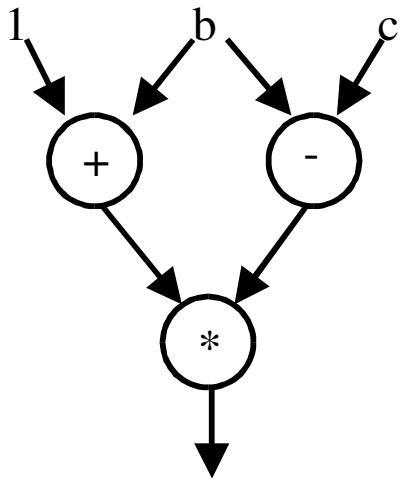
Multithreaded architectures

Data flow models

- control is tied to the flow of data
- an instruction is executed if all the data needed for the execution is available
- no concept of shared, rewritable data storage

Example: compute a=(b+1)*(b-c)





Multithreaded architectures

Thread-level (multithreaded) architectures			
von Neumann based	Hybrid von Neumann/dataflow		
	RISC-like	Decoupled architectures	Macro dataflow architectures
HEP TERA MIT Alewife & Sparcle	P-RISC *T	USC McGill MGDA & SAM	MIT Hybrid Machine EM-4

Multithreaded architectures

Denelcor HEP

Features

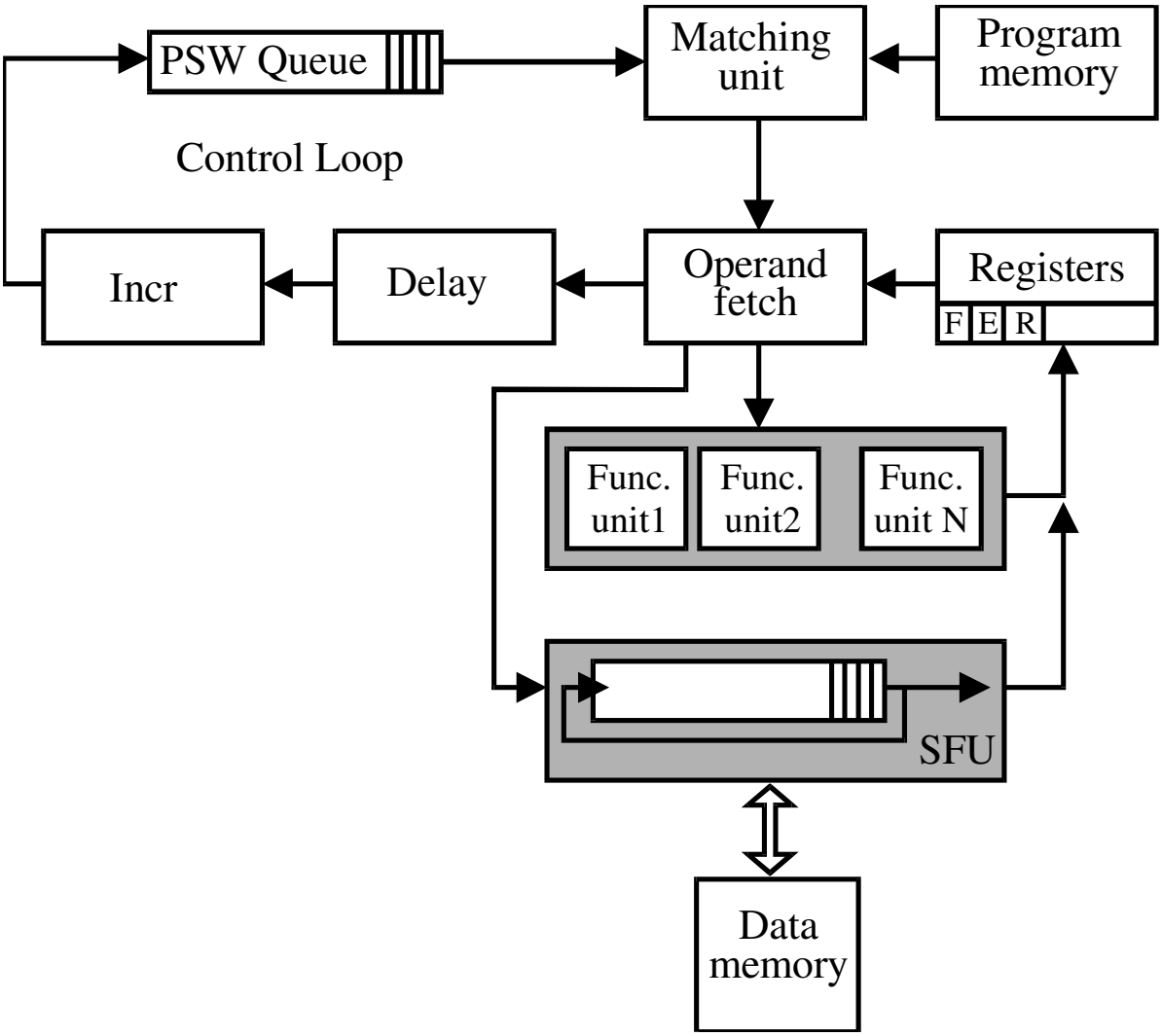
- very first parallel architecture designed with objectives of tolerating memory latency and synchronisation problems
- physically shared memory multiprocessor with up to 16 processors and 128 memory units
- a fine-grain interleaving of threads (instruction level)
- memory latency is solved by the Scheduler Function Unit
- synchronisation among threads is via registers (2048)

Limitations

- threads can have only one outstanding memory request
- bubbles may appear in the processor pipeline at synchronisation
- maximum 64 threads per processor
- the processor is badly utilized when the available parallelism is limited

Multithreaded architectures

Denelcor HEP



Multithreaded architectures

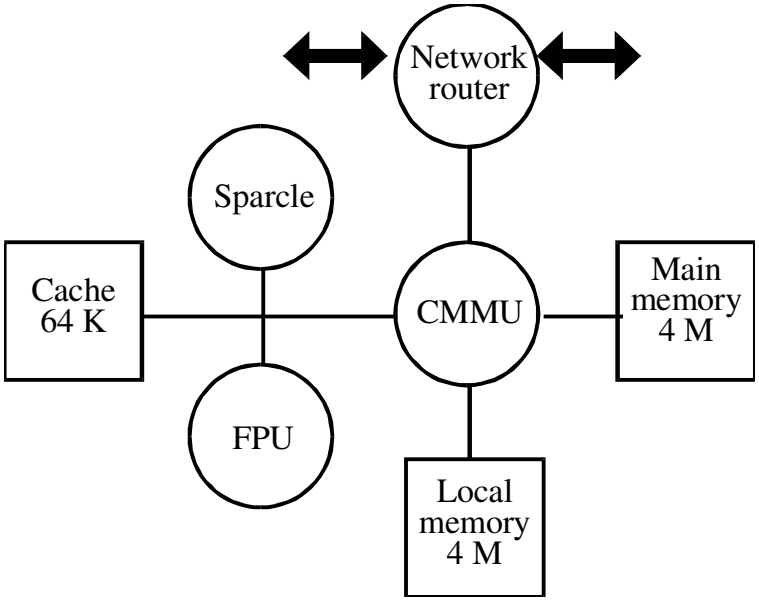
Tera

- a modern version of the architecture of the Denelcor HEP processor
- distributed shared memory system
- up to 256 processors, 512 memory modules, 256 I/O cache units, 256 I/O processors and 2816 communication nodes
- 128 threads per processor
- context switching extremely fast, due to 41 registers (64-bit each) per thread
- the multithreaded Tera processor has the same advantages and drawbacks as the HEP processor

Multithreaded architectures

MIT Alewife & Sparcle

- large-scale cache coherent distributed shared memory
- maximum 512 processing nodes in a two-dimensional mesh topology
- each node consists of a Sparcle processor, 4-Mbyte main memory, 64-Kbyte cache, 4-Mbyte local memory, floating point unit, communication and memory management unit (CMMU), and a network router/switch.

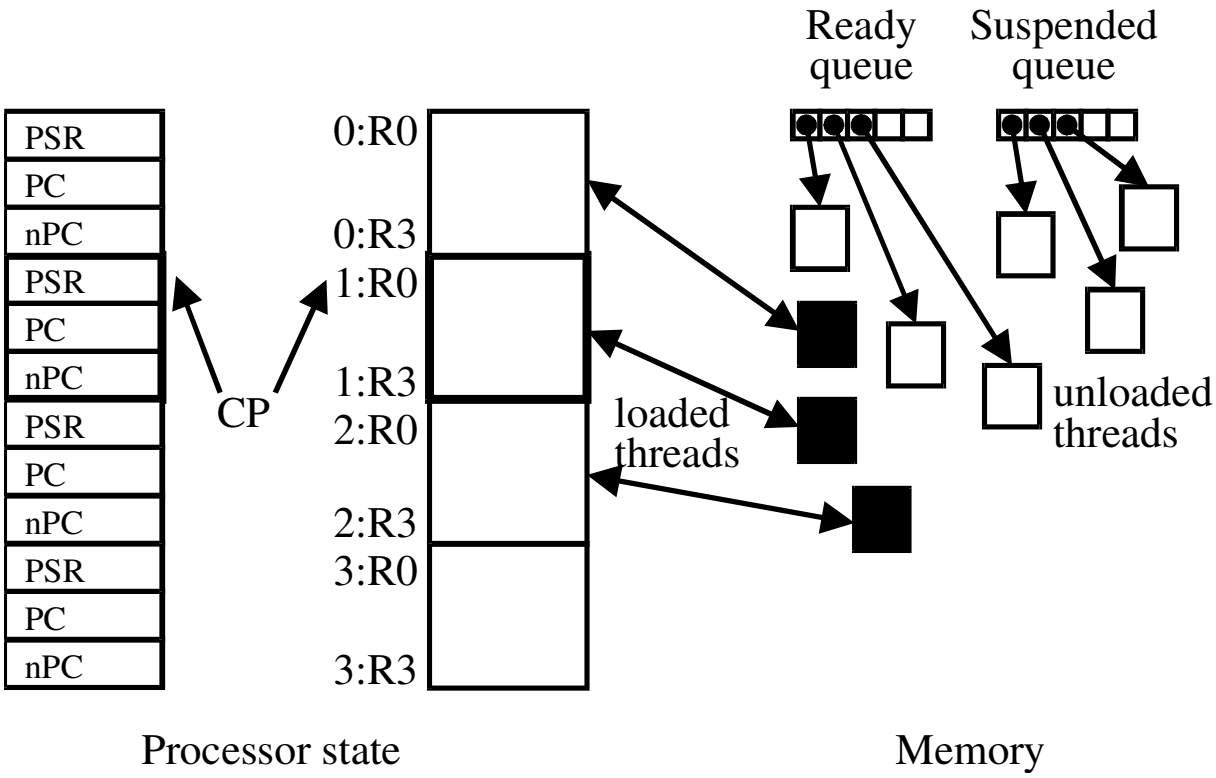


- physically distributed memory
- logically shared memory
- hardware-supported cache coherence
- hardware-supported user-level message-passing
- integration of multithreading capabilities with hardware-supported cache coherence

Multithreaded architectures

The Sparcle processor (a modification of Sparc)

- memory latency tolerance by fast context switching
- fine-grain synchronisation
- efficient user-level message-passing



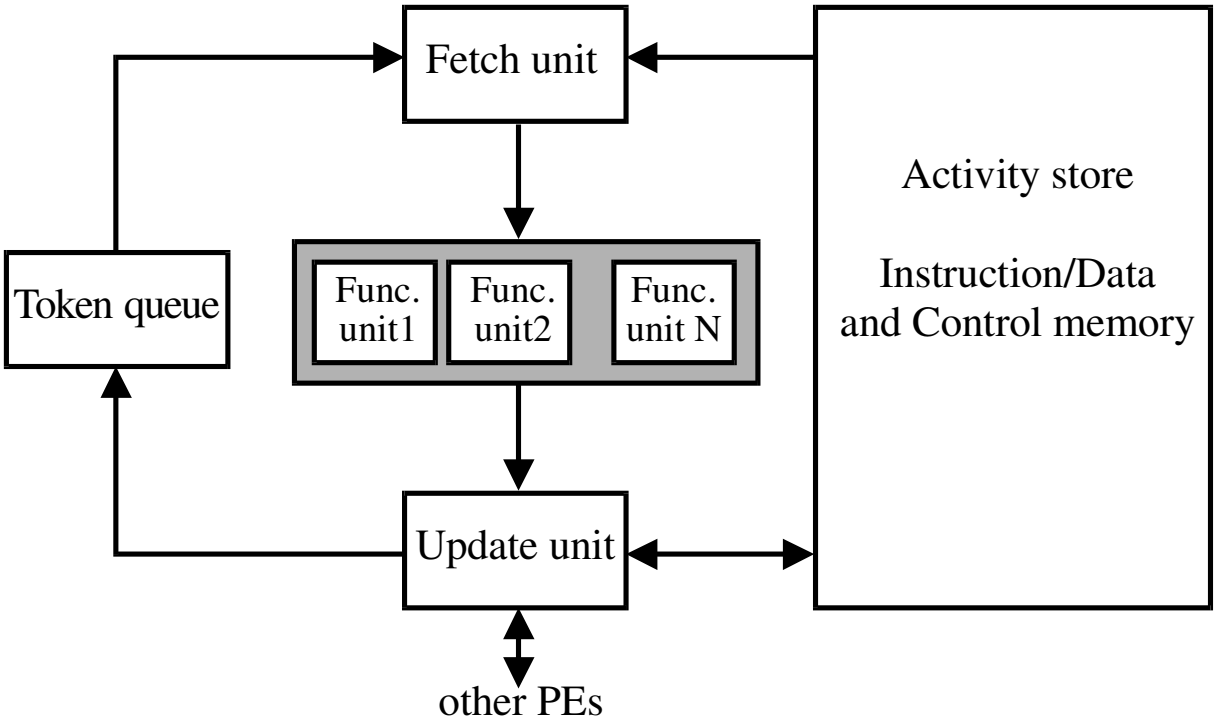
Multithreaded architectures

Summary and comparison of von Neumann based multithreaded systems

	HEP (Tera)	MIT Alewife
Processor design	specially designed	modified commodity RISC processor
Granularity	fine-grain instruction level thread interleaving each instruction from different threads in the 8-stage pipeline thread switching is 1 clock cycle	coarse-grain executes the same thread as far as possible each instruction from the same thread thread switching is min. 14 clock cycles frequency of remote memory access should be reduced
Memory organization	no cache dance-hall shared memory	cache is needed distributed shared memory with coherent caches
Support for synchronisation	full/empty bit in memory busy waiting superfluous memory traffic full/empty/reserved bits in registers bubbles in the processor pipeline	extended full/empty bit (J-struct, L-struct) thread suspension in queues no extra memory traffic
Number of threads	should be high to keep the pipeline busy (64)	number of loaded threads must be low (3)
Available parallelism	if low, low processor utilization	no matter, few threads are enough
Sequential behaviour	very poor processor utilization	good processor utilization as far as no remote memory access is needed

Multithreaded architectures

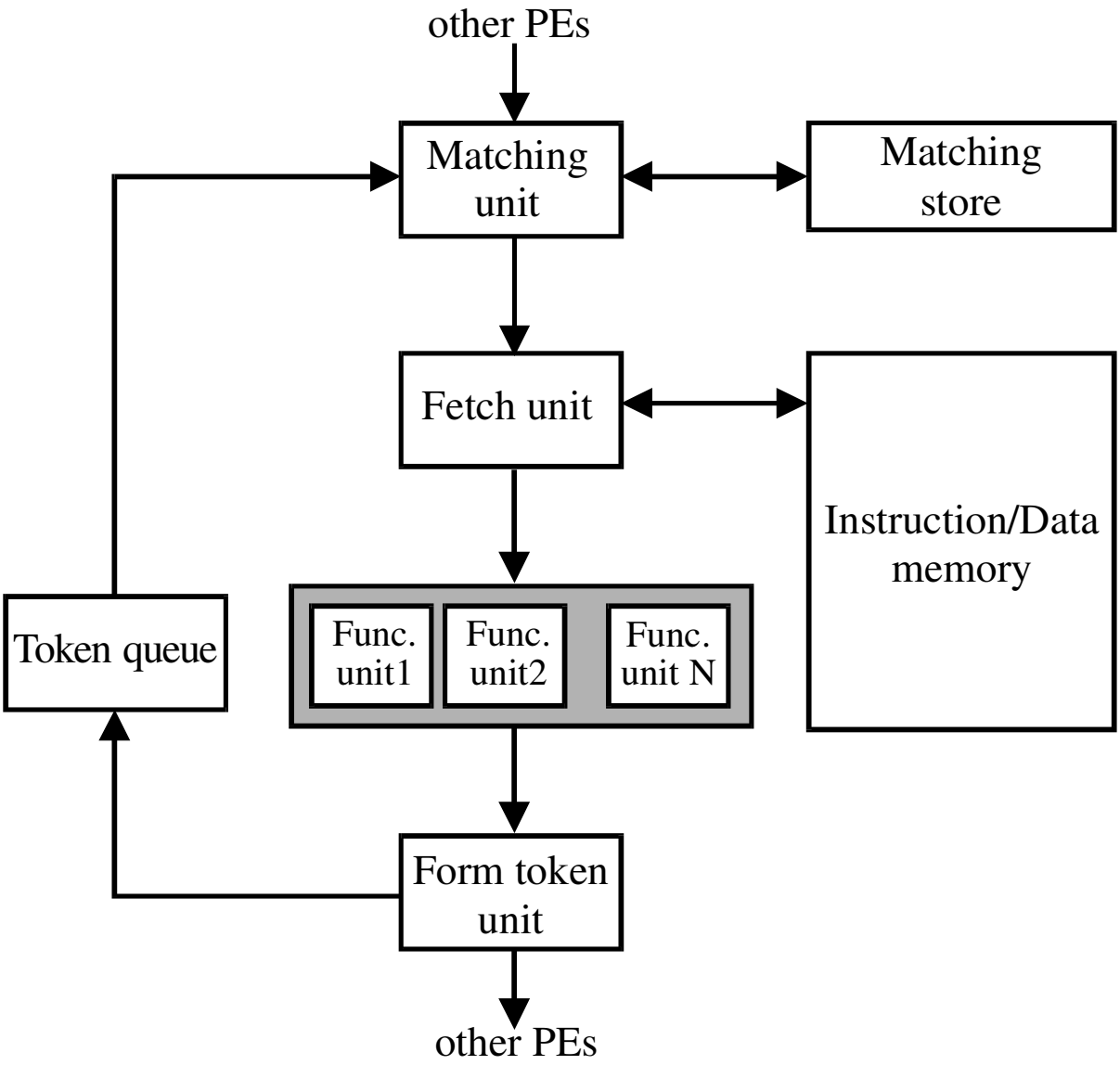
Dataflow architectures - static dataflow architectures



- A node can fire if there is a token on each input arc and there is no token on its output arcs
- Token : <value, <function, target node>, destination arc>
- Tokens of a target node match if their <function, target node> fields are the same
- Activity store: Instruction/Data and Control memory
- Control memory frames hold the presence bits for all operands and target_node/destination arc fields. A node can fire if all presence bits in the frame are set
- The parallelism is strongly limited

Multithreaded architectures

Dataflow architectures - tagged-token dataflow machines

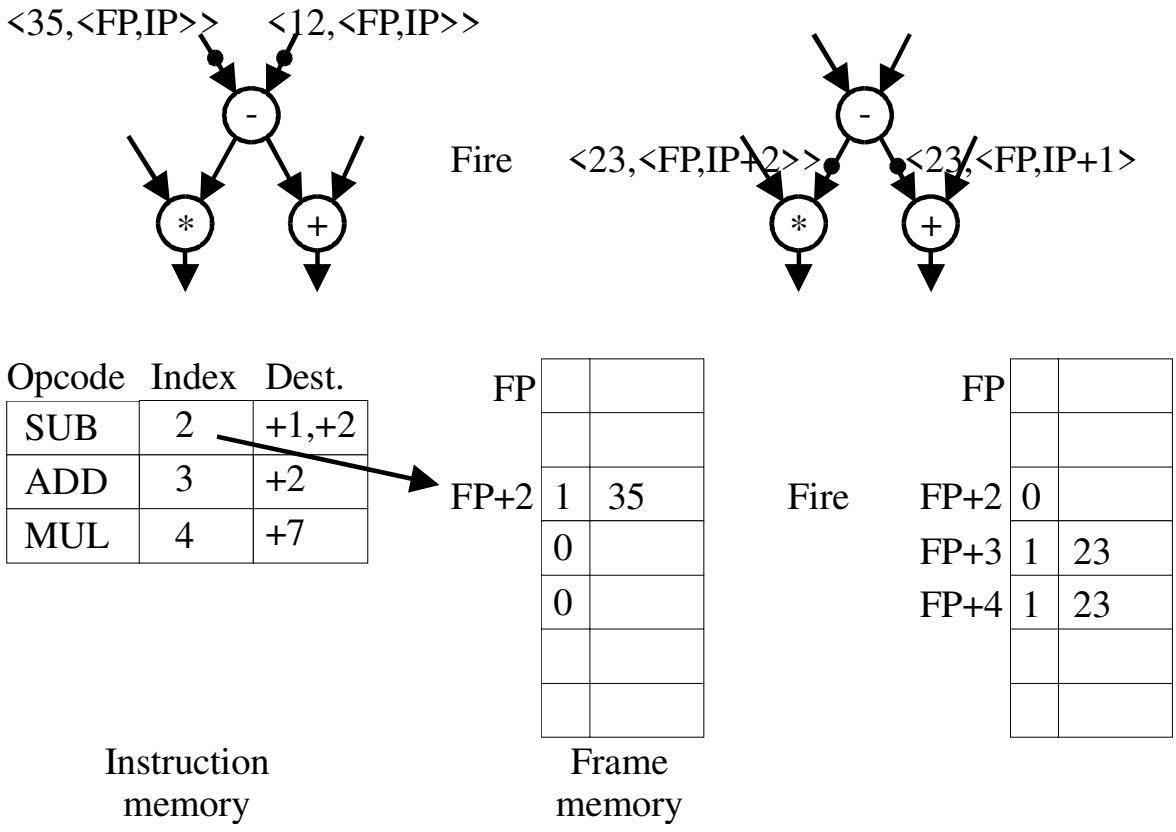


- A node can fire if there exists a colour *i* for which tokens of colour *i* are available on each input arc
- Token: <value,<function, target node, code, index(colour)>, destination arc>
- Support: Matching unit containing tokens which wait for their partner token of the same colour

Multithreaded architectures

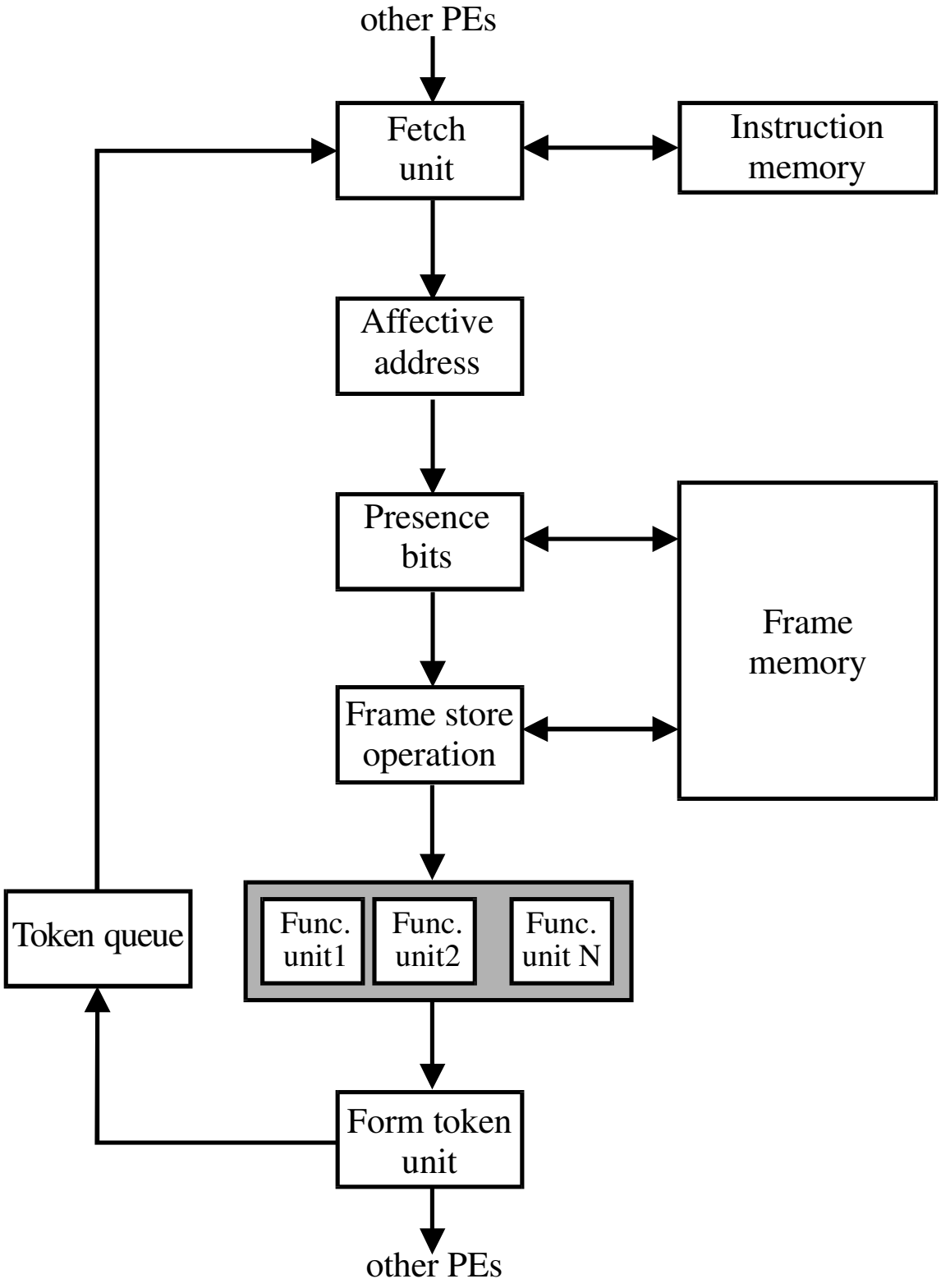
Dataflow architectures - explicit token-store architectures

- Tokens of the same colour belong to the same activation block, and therefore, a separate memory segment can be created for them - Frame memory
- Frame Pointer (FP) = colour, Instruction Pointer (IP) = <function, target node, code, index(colour)>
- Token: <v, <FP, IP>>



Multithreaded architectures

Dataflow architectures - explicit token-store architectures



Multithreaded architectures

Hybrid multithreaded architectures

- The dataflow instruction-level scheduling is superfluous for processing inherently sequential instruction stream
- To reduce overhead, increase granularity and exploit parallelism only among procedures. Inside the procedures the sequential execution scheme of von Neumann machines is applied.
- Hybrid von Neumann/Dataflow architecture

	Dataflow
hybrid architectures	Macro dataflow
	Decoupled hybrid dataflow
	RISC-like hybrid
	von Neumann

Multithreaded architectures

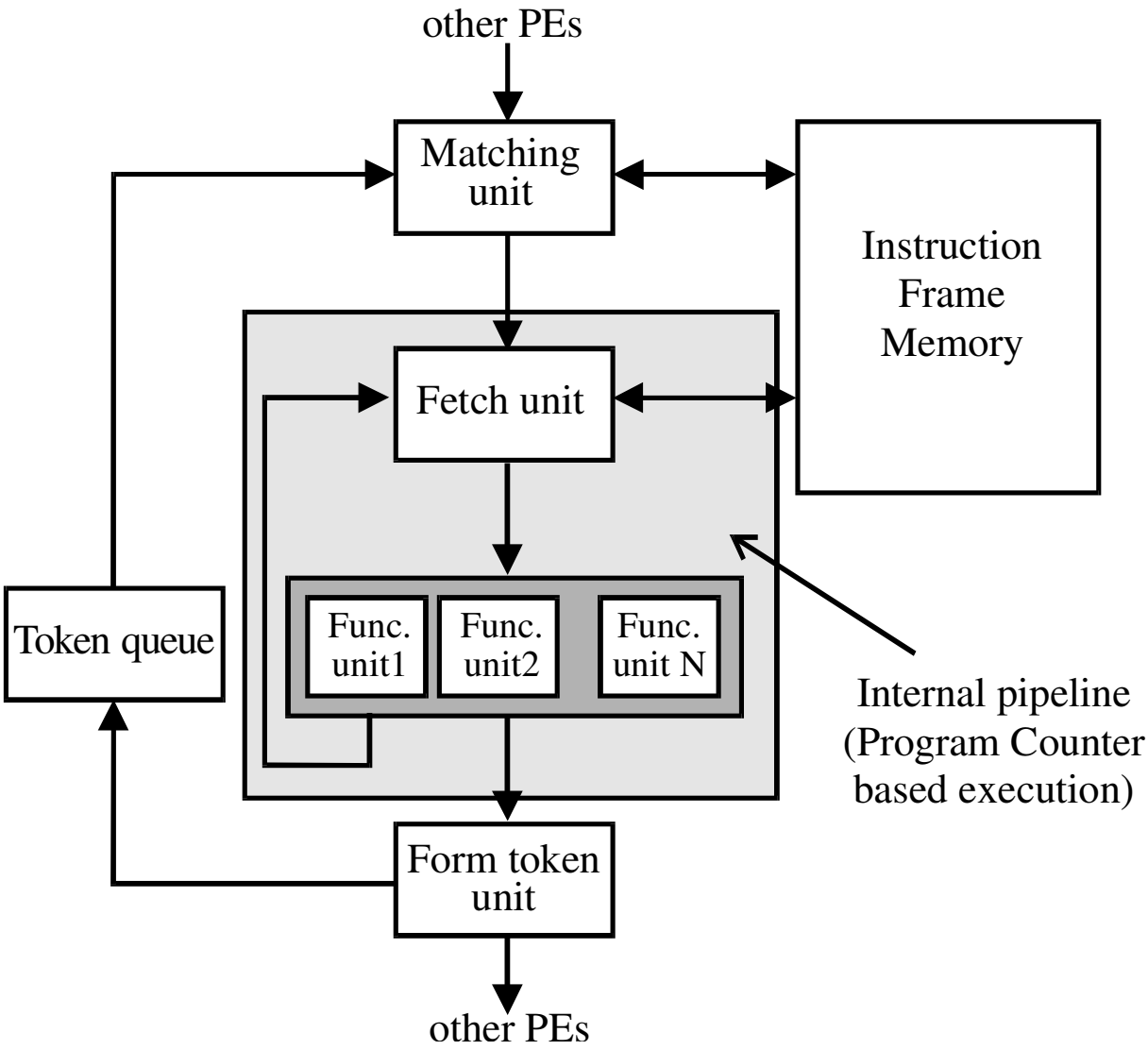
Hybrid multithreaded architectures

	von Neumann architectures	Hybrid architectures	Dataflow architectures
Instruction scheduling flexibility	total sequencing	machine language is able to express partial ordering	sequencing is based on data dependency
Latency toleration	cannot tolerate long latency operations	able for efficient context switch at a low level to hide latency	efficient multi-phase operations called split-transaction to tolerate latency
Synchronisation support	lack of efficient synchronisation at low level	dataflow ideas to be used only at interprocedural level	intra-procedure communication is unnecessarily general

Multithreaded architectures

Macro dataflow architectures

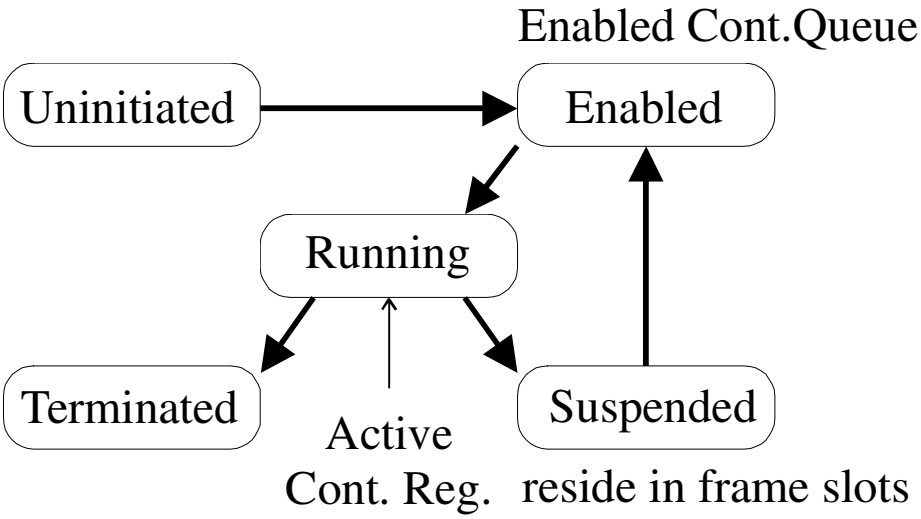
- The first step from conventional dataflow design towards hybrid multithreaded architectures



Multithreaded architectures

MIT Hybrid Machine

- Concept of Shceduling Quantum (SQ): a sequential partition in the dataflow graph according to any criterion of optimality:
  - maximisation of exploitable parallelism
  - maximisation of run length in SQs
  - maximisation of machine utilisation
  - minimisation of explicit synchronisation
  - deadlock avoidance
- An SQ is implemented as a thread and represented as a continuation
- Continuation: <Program Counter, Frame Base Register>
- Continuation states:

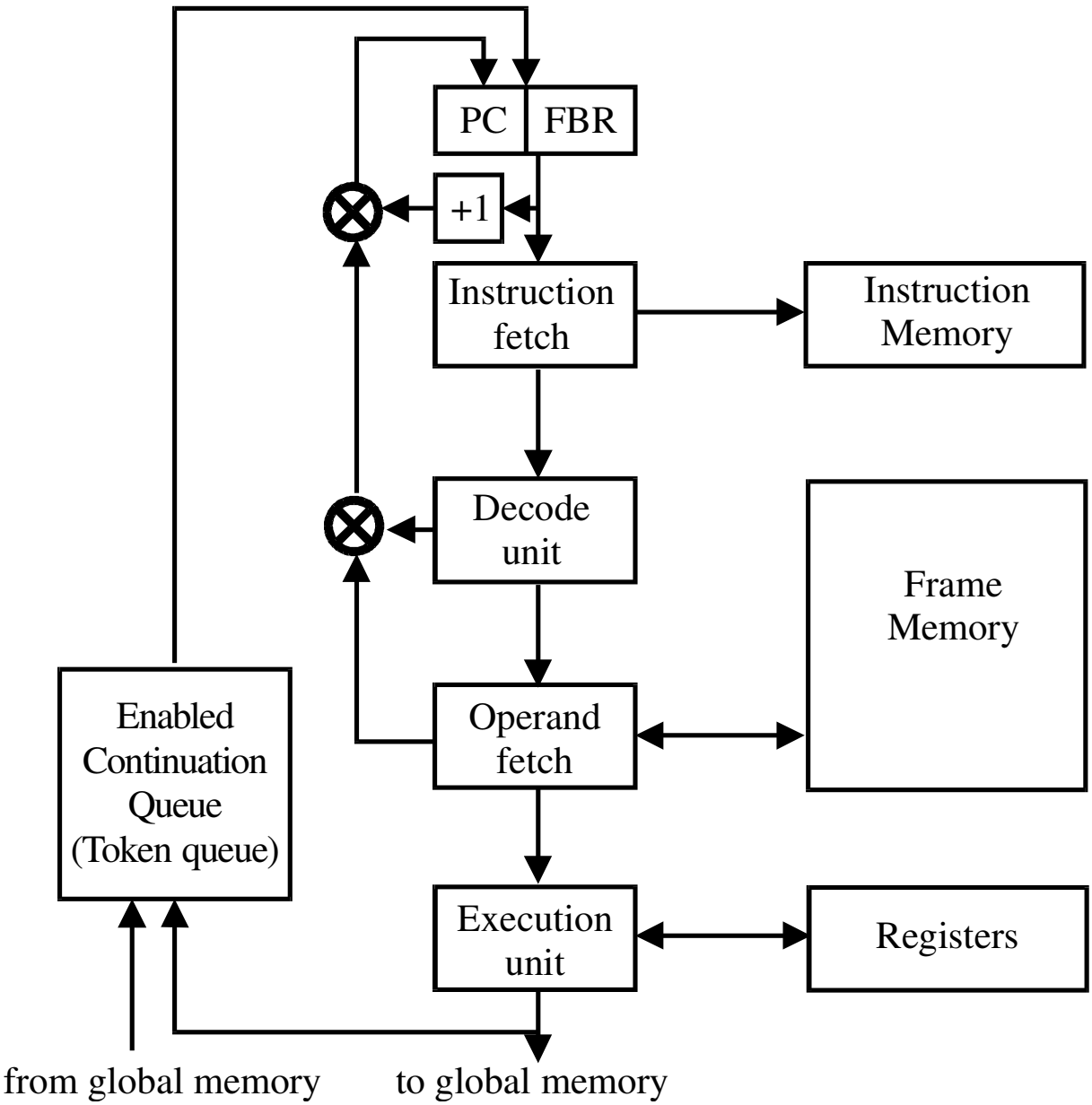


- Encoding of continuation states by location of the continuation

Multithreaded architectures

MIT Hybrid Machine

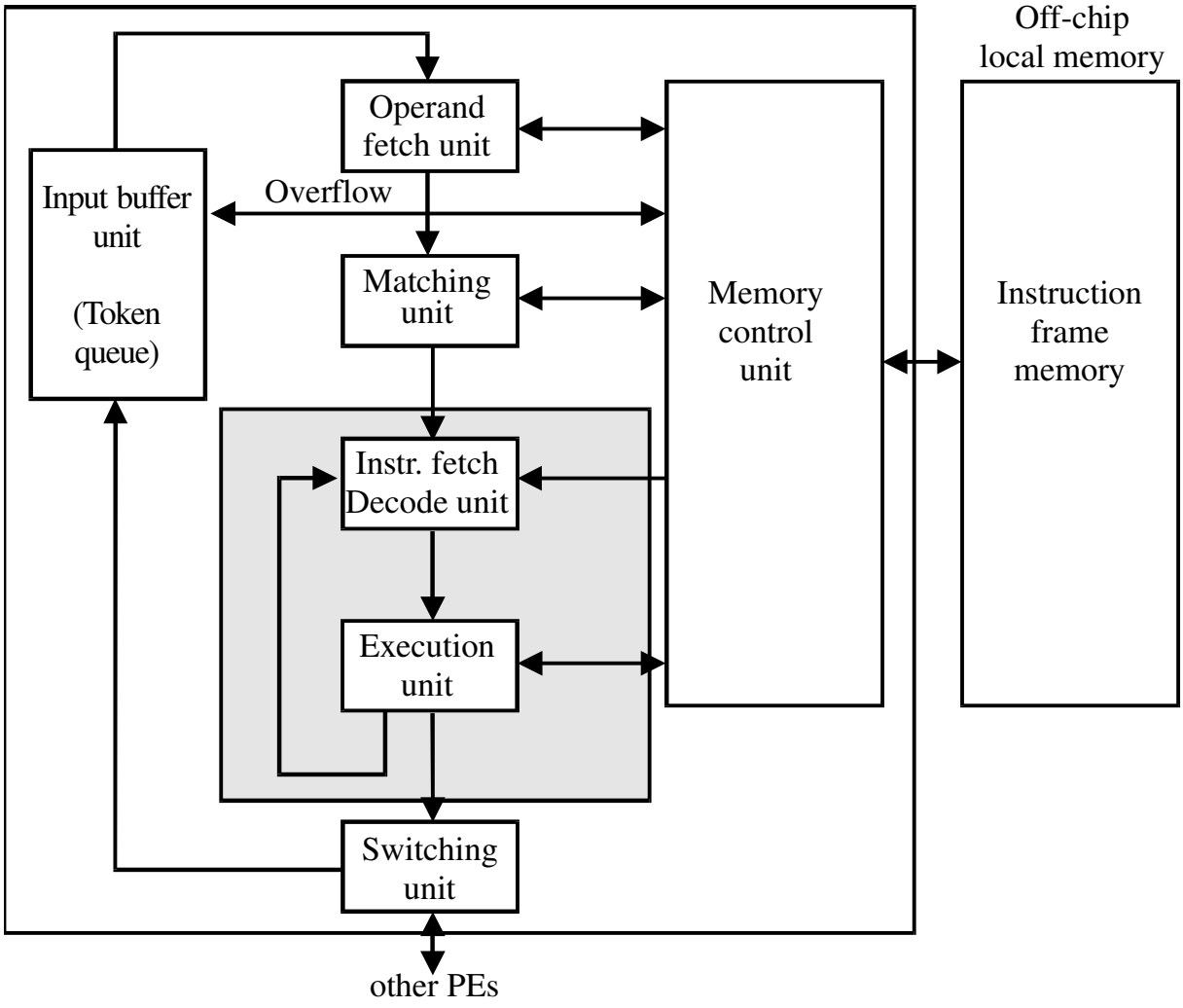
Structure of a PE



Multithreaded architectures

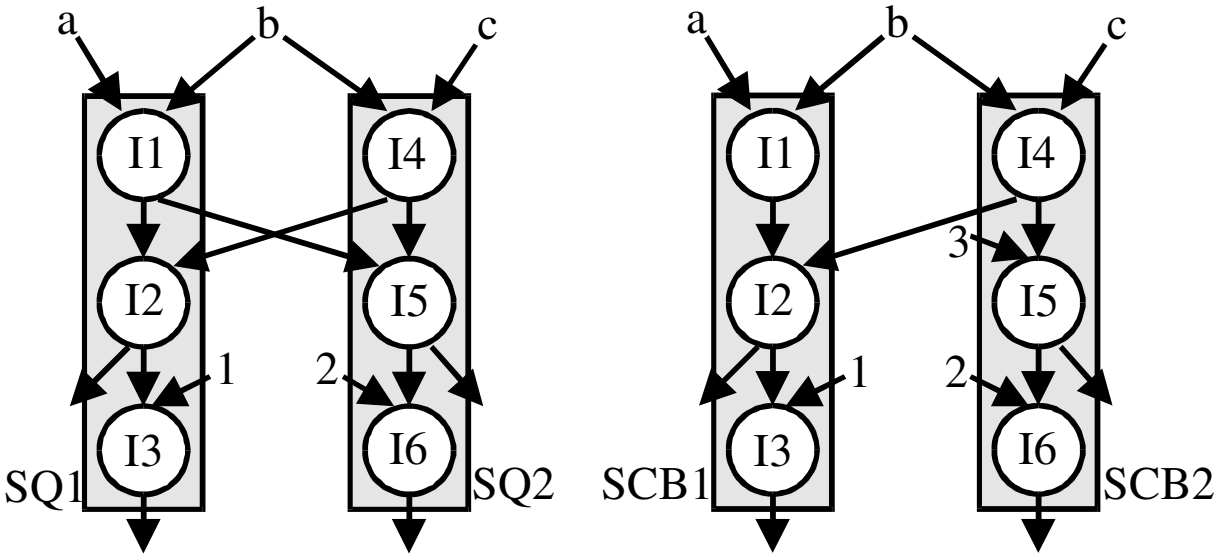
EM-4 macro dataflow machine

- Concept of Strongly Connected Block (macro node). Nodes inside a macro node are connected by strongly connected arcs. Nodes inside an SCB are executed sequentially, an SCB behaves like a thread.
- An SCB can fire when all its input arcs contain a token, i.e. macro nodes are executed based on the dataflow semantics.



Multithreaded architectures

Comparison of SQ and SCB concept



- Execution of instructions in both kinds of macro nodes are sequential
- An SCB can be activated only when all of its input tokens are already available. Afterwards it can be executed without suspension
- Execution of an SQ can be started when its first instruction (node) is enabled. If an instruction inside an SQ later encounters a missing data (token) the whole SQ (thread) execution is suspended on the empty memory slot
- Mutual waiting would cause deadlock in case of SCB . If the mutual dependence is extracted, the SCB concept can be applied, too. First SCB2 can start its work and when I4 is completed, SCB1 becomes enabled, too.

Multithreaded architectures

Decoupled hybrid dataflow architectures

- Decoupled: scheduling and execution units are separated
- Parallel control flow model based on control tokens

Multithreaded architectures

The USC Decoupled Architecture

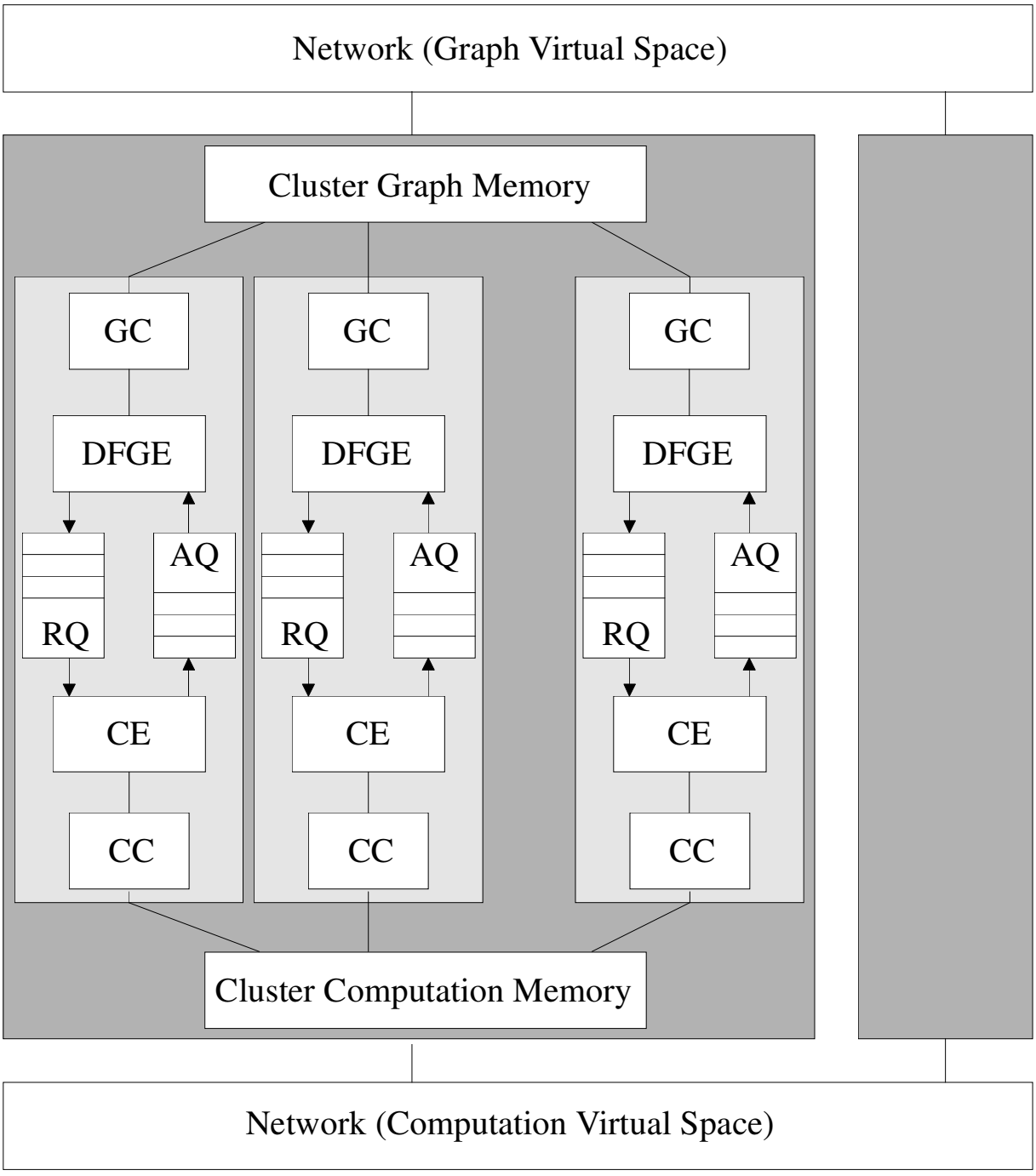
- Multilevel dataflow model. Three types of dataflow nodes:
  1. Scalar node
  2. Vector-macro node (vector operations)
  3. Compound macro node: collection of scalar and/or vector nodes. Scheduled on the same principle as the SCB
- A processing element consists of a Graph Cache, a Data-Flow Graph Engine (scheduling), a Computation Cache and a Computation Engine (execution). They communicate vie the Ready Queue and the Acknowledge Queue.
- Processing elements are clustered

Advantages

1. The USC machine supports both thread-level and instruction-level parallelism
2. The computation engine can be any RISC processor
3. The number of threads per processor is virtually not restricted.

Multithreaded architectures

The USC Decoupled Hybrid Architecture



Multithreaded architectures

The Super-Actor Machine (SAM)

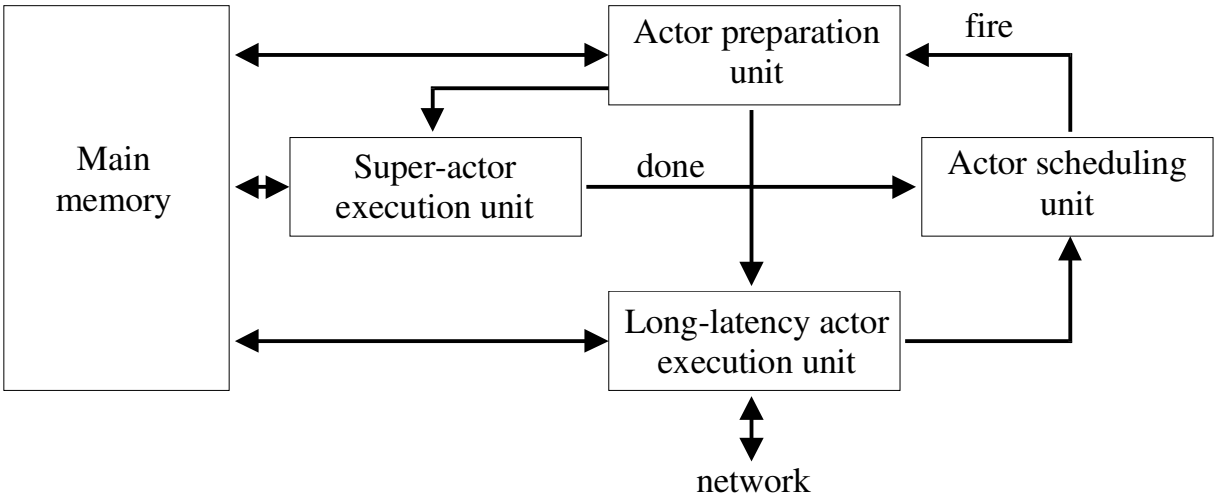
- Thread is a 'super-actor' equivalent to a macro node of the dataflow graph
- Sequential and parallel super-actors are distinguished by data dependency (or independency)
- A super-actor can be enabled if
    1. Firing rule of static dataflow models: all the input arcs should contain a token and its output arcs should be empty
    2. Space locality condition: Its input data are physically residing in the register-cache and space is reserved there to store the results.

Remote load latency does not occur due to the space locality condition

Synchronisation with other super-actors take place in the Actor Scheduling Unit which processes the control tokens as well

Multithreaded architectures

Structure of a node in SAM



- The ASU implements the firing rule condition and the APU realises the space locality condition

Advantages

1. The SAM supports both thread-level and instruction-level parallelism based on compile-time decision.
2. The SEU is an optimised pipelined RISC processor enhanced with the register-cache.
3. The execution of a thread is atomic without any long-latency operation

Multithreaded architectures

RISC-like hybrid multithreaded architectures

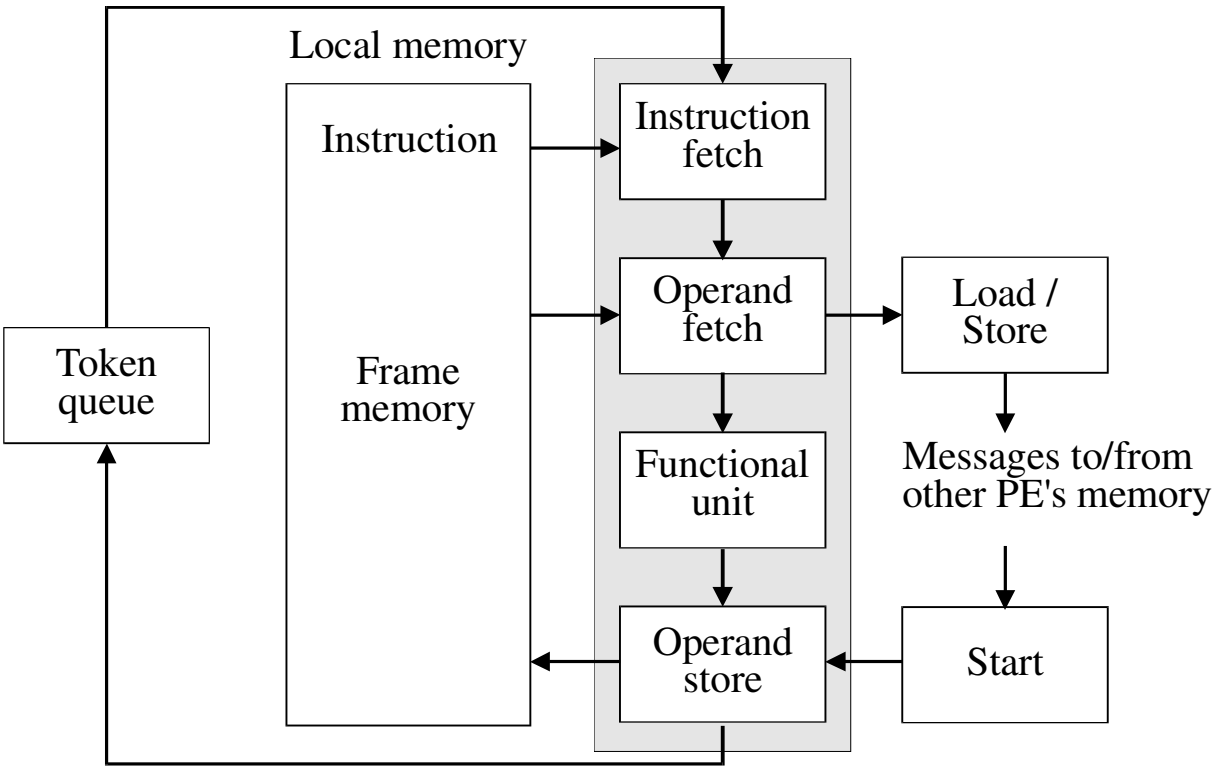
- The closest architecture type to the von Neumann machines
- Parallel control operators (FORK, JOIN) based computational model
- Synchronistaion on reusable memory cells is necessary which represents overhaed and requires special care at user level
- The main advantage of the hybrid multithreaded architectures is that they are able to solve the synchronisation in a fine-grain way with low overhead as a consequence of inheriting latency hiding techniques of dataflow machines
- Examples

    1. P-RISC developed at MIT
    2. *T - (star)T - developed at MIT and Motorola

Multithreaded architectures

P-RISC

- Tokens (continuation, context) : <FP.IP> (like macro dataflow machines)
- In each clock cycle a new thread enters the pipeline (like HEP)



- After executing the token <FP.IP>, a new token is placed into the queue: <FP.IP+1> (incrementing the PC)

Multithreaded architectures

P-RISC

- Remote load: split-phase operation: the request and the answer are separated
- **load a x** is transformed into a message **<READ, [FP+a], FP.IP+1, x>** which is sent via the interconnection network
- the reply to a read request is **<START, v, FP.IP+1, x>**, the value v is placed into [FP+x] and the continuation <FP.IP+1> is placed into the queue

Advantages:

1. Any number of loads can be performed in the distributed system.
2. Responses can return in any order without disturbing the original semantics of the program.
3. The processor is not blocked during load operations (unlike conventional RISC processors).
4. Meanwhile a thread is travelling in the network other threads from the token queue can keep the processor busy.

Multithreaded architectures

P-RISC

- Store operation: based on message passing mechanism
- **store x a** is tranformed into a message **<WRITE, [FP+a], [FP+x]>**, which is sent via the interconection network
- The updated <FP.IP+1> is placed into the queue immediatelly (no need to wait for the response)
- Thread synchronisation on memory locations is realised by means of I-structure memories

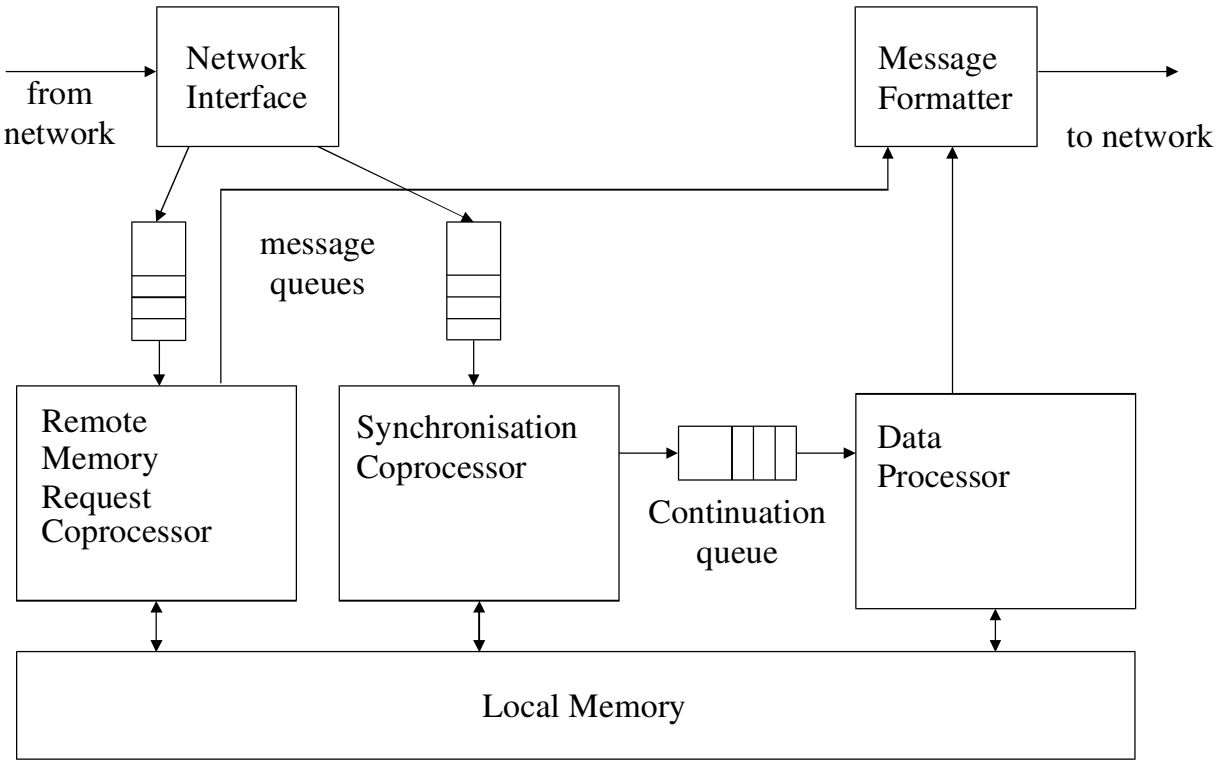
Support for fine-grain thread-level parallelism

- Creating threads: FORK IPt: insertion of two continuations, <FP.IP+1>, <FP.IPt> into the token queue
- Removing threads: JOIN x. Memory cell FP+x is checked. If it contains 0, the thread is removed and [FP+x] is set to 1. Otherwise the thread can continue (the updated continuation is put into the queue

Multithreaded architectures

*T

- Creating a commercial multithreaded chip based on the principles of P-RISC
- Two functional coprocessors were added to support remote memory access



Data processor: contains the execution pipeline and operates on local instructions

Remote Memory Request Coprocessor: handles incoming rload and rstore requests

Synchronisation Coprocessor: handles returning rload responses (<START, v, FP.IP+1, x>)

Multithreaded architectures

Sumary of multithreaded architectures

Machine	Computational model	Granularity	Memory organization	Thread / processor	Year
Denelcor HEP	von Neumann	fine	physically shared	max. 64	1978
Tera	von Neumann	fine	distributed shared + I/O cache	max. 128	1990
MIT Alwife	von Neumann	coarse	cache-coherent distributed shared	max. 3	1990
EM-4	macro dataflow	coarse	distributed shared	no limit	1990
MIT Hybrid Arch.	macro dataflow	coarse	physically shared + local	no limit	1988
SAM	control token	coarse	distributed shared + register cache	no limit	1991
USC Decoupled Arch.	control token	coarse	shared + cache	no limit	1990
P-RISC	control operators	fine	distributed shared	no limit	1989
*T	control operators	fine	distributed shared	no limit	1992

Multithreaded architectures

von Neumann		hybrid von Neumann / dataflow	
fine-grain thread interleaving	coarse-grain thread interleaving	fine-grain thread interleaving	fine-grain thread interleaving
		macro decoupled node arch	RISC-like
• large number of registers to enable as many threads as possible	• coherent cache support to reduce the frequency of thread switches	• sophisticated compiler support is needed	• loads are split-phase transactions • any number of loads can be simultaneously executed • responses can return in any order • the processor is not blocked during load
HEP, Tera	MIT Alewife	MIT Hybrid Arch., EM-4, USC Decoupled Arch., SAM	P-RISC, *T