Multithreaded Architectures
Lecture 2 of 4

Supercomputing '93 Tutorial

Friday, November 19, 1993

Portland, Oregon

Rishiyur S. Nikhil

Digital Equipment Corporation
Cambridge Research Laboratory

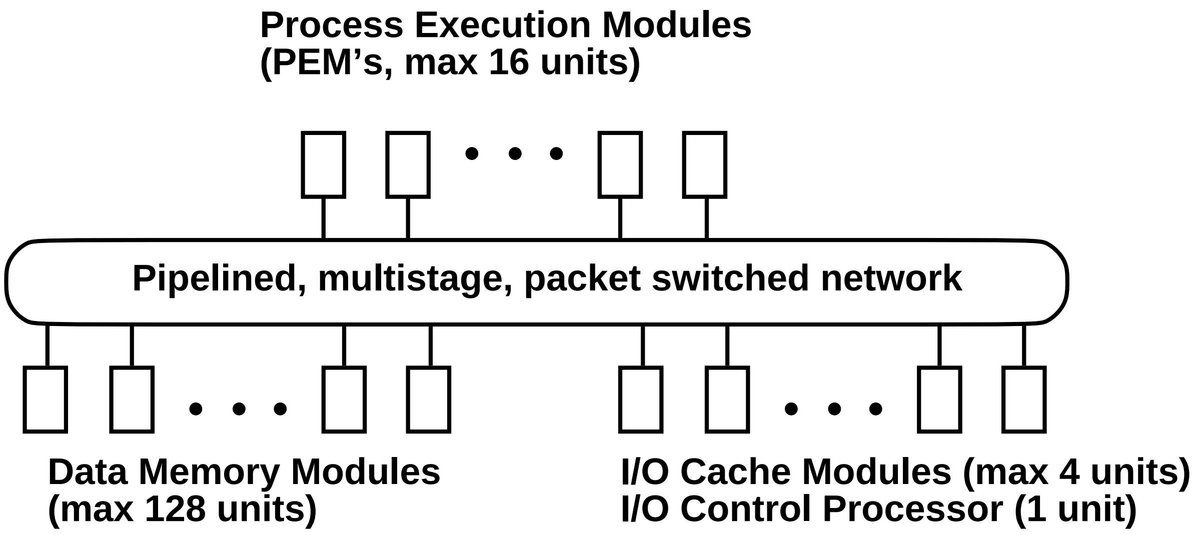
Overview of Lecture 2

Multithreading: A von Neumann Story

- **The Denelcor HEP**
- **The Tera System**
- **The MIT Dataflow/von Neumann Hybrid Architecture, and the IBM Empire**
- **The MIT J−Machine**

The Denelcor HEP

- Late 1970's, early 1980's
- Principal architect: Burton Smith

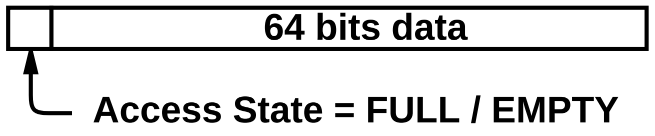

Process Execution Modules
(PEM's, max 16 units)
Pipelined, multistage, packet switched network
Data Memory Modules
(max 128 units)
I/O Cache Modules (max 4 units)
I/O Control Processor (1 unit)

- 10 MHz clock, synchronous
- Max threads*: 2048 (128 threads/PEM)

*(threads are called "processes" in HEP terminology)

The Denelcor HEP Data Memory Modules

- Max 128 modules, each with 1 MW (8 MB) memory
- Bandwidth of each module: 10 M accesses/sec
- Each word:

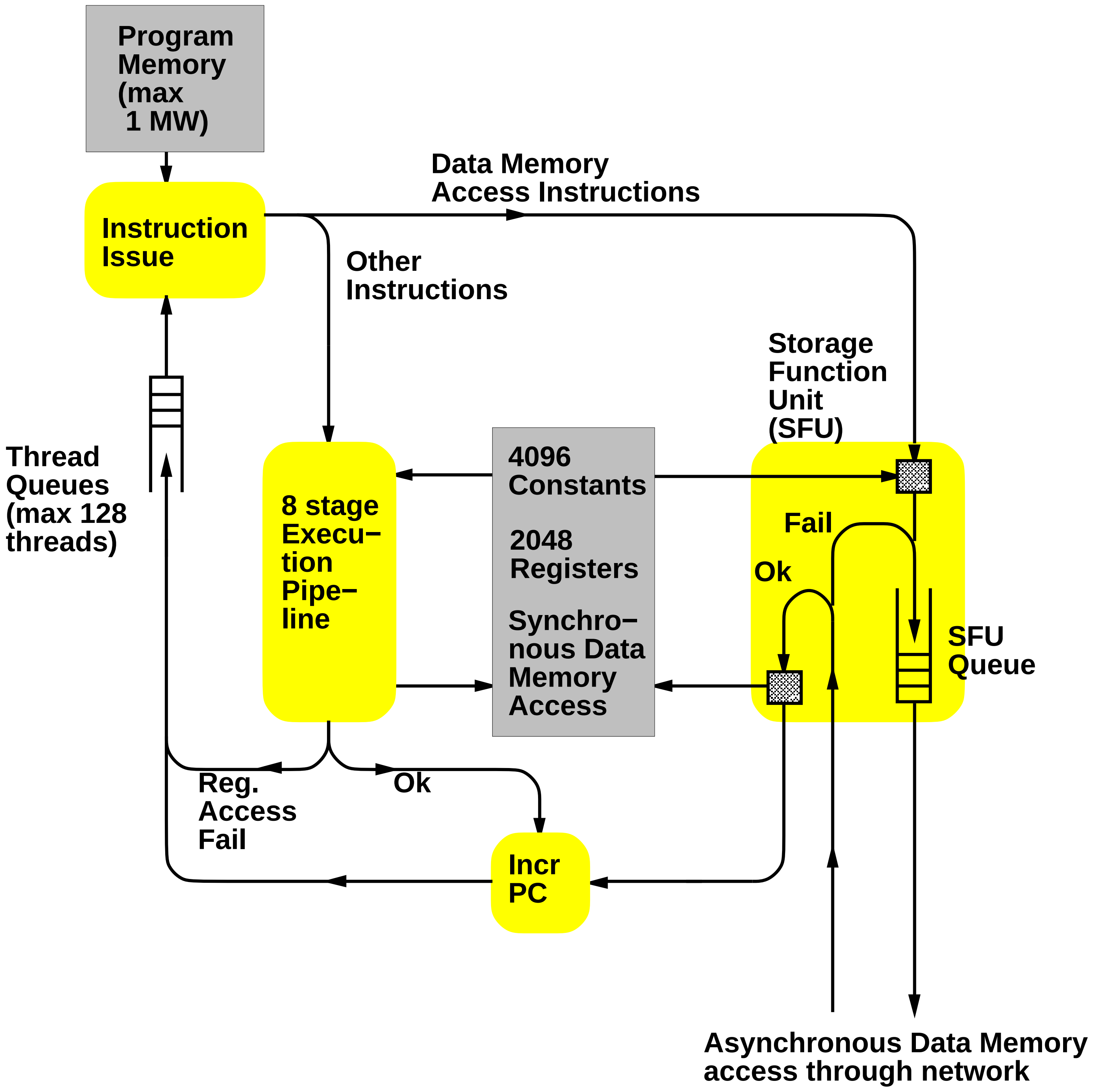


- Conventional LOAD/STORE transactions, plus:
  - Set/clear full/empty bits
  - Synchronizing LOADs:
    - If FULL return packet with data value, set EMPTY
    - If EMPTY return packet with failure status
  - Synchronizing STOREs:
    - If FULL return packet with failure status
    - If EMPTY store data, set FULL, return packet with ack
  - ...

(thus, each location can be used as 1−word mailbox)

The Denelcor HEP PEM's (Process Execution Modules)

- Max 16 units
- 10 MHz clock, 10 MIPS peak


Program Memory (max 1 MW)
Instruction Issue
Data Memory Access Instructions
Other Instructions
Storage Function Unit (SFU)
Thread Queues (max 128 threads)
8 stage Execu-tion Pipe-line
4096 Constants
2048 Registers
Synchro-nous Data Memory Access
Fail
Ok
SFU Queue
Reg. Access Fail
Ok
Incr PC
Asynchronous Data Memory access through network

The Denelcor HEP PEM Scheduling Hierarchy

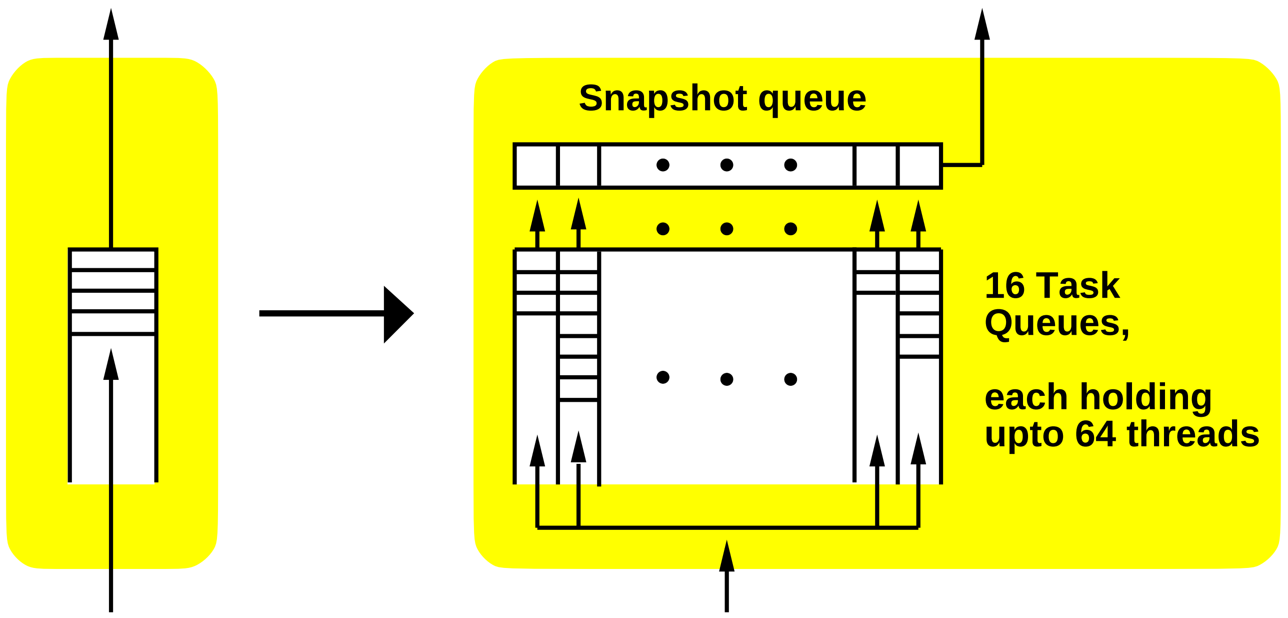
TASK

- = protection domain
- max 16 tasks/PEM (7 User, 8 Supervisor)
- Created with Supervisor call
- Bases and Limits into Constants/−Registers/Program Memory/Data Memory

THREAD (PROCESS)

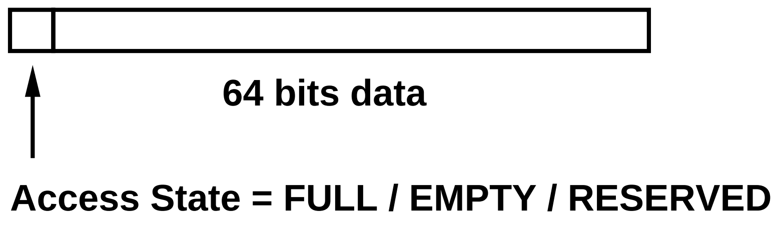
- Max 50 threads/user task
- Max 128 threads/PEM
- Created in single user−level instruction
- Described by PSW (Process Status Word)
  - PC (20 bit)
  - Offsets into Constants/Registers (within Task Base and Limit)
  - trap masks, etc.

Queue scheduling (thread and SFU queues):



The Denelcor HEP Register Synchronization

- **Registers:**



- **FULL/EMPTY states: programmer−visible**

- **Threads (within a task) can communicate through shared registers using synchronizing accesses:**
  - **Wait till EMPTY, store, set FULL**
  - **Wait till FULL, read, set EMPTY**
  - **etc.**

- **RESERVED state used internally:**
  - **When an instruction begins execution, its destination register is marked RESERVED**
  - **When its destination register is written, its state changes to FULL**
  - **Any instruction that finds either a source register or a destination register RESERVED becomes a "no op"; its PC is unchanged, so it will be retried**

The Denelcor HEP PEM Performance

- **Saturated PEM runs at 10 MIPS**

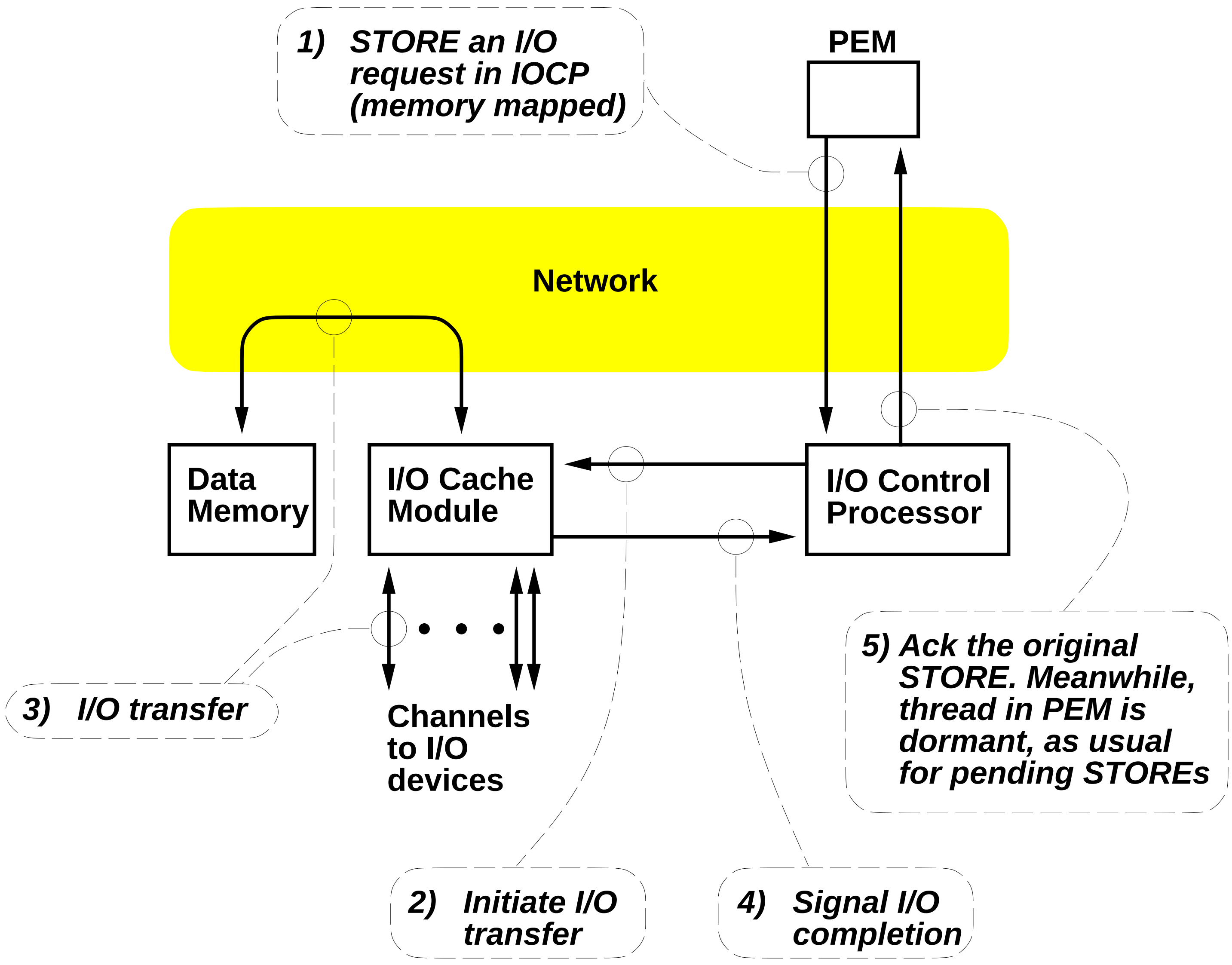
- **However, since a particular thread traverses the 8 tick pipe before its next instruction is ready for issue, a particular thread cannot exceed 1.25 MIPS**
  - **In fact, if more than 8 extant threads, each thread runs at a lesser rate**

- **Typically, about 12 threads needed to saturate a PEM (8 for execution pipe, a few more for memory pipe)**

- **Typical round−trip to Data Memory: 20−30 ticks (on "small, moderately loaded" system)**

The Denelcor HEP Multithreading for I/O

- **No "interrupts"; asynchronous events handled using normal, asynchronous Data Memory operations**



- **In addition, the supervisor initiates concurrent "read−ahead" and "write−behind"**

The Denelcor HEP Miscellaneous

- No caches, no virtual memory
- Instruction lookahead if there is no other work:
  - If thread queues all empty, next instruction from last thread dispatched is examined for independence from previous instruction.

    If so, that instruction is also issued.
  - Sole exception to rule that only one instruction from each thread can execute at a time
  - Single−thread performance improves >= 25 %
- Each PEM also has direct path to one Data Mem module
  - Accesses in usual 8 ticks
  - FULL / EMPTY synchronization disallowed

The Denelcor HEP Assessment

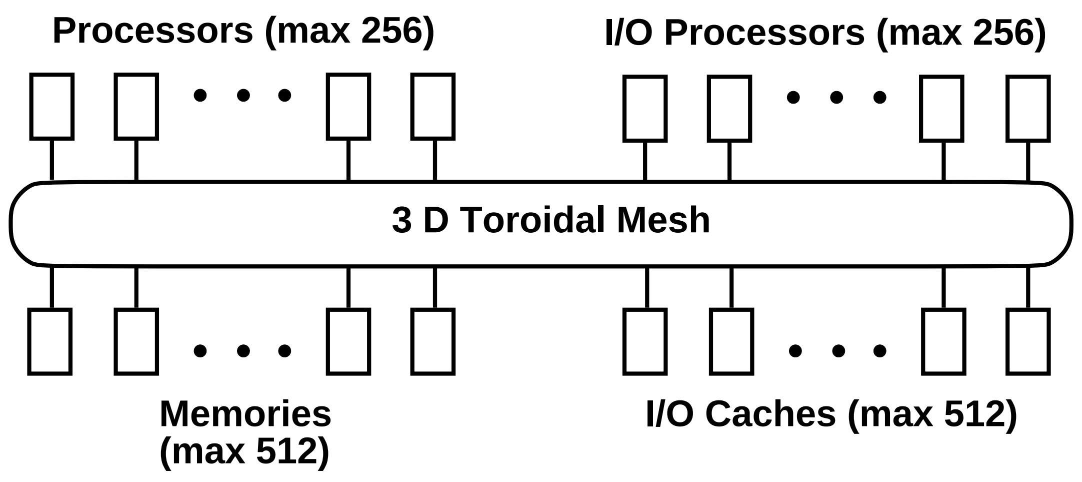
- **Excellent support for "Remote Loads"**
- **For "Synchronizing Loads":**
  - **PEM busy−waits on empty location**
  - **Burns network and memory bandwidth (but not instruction−issue bandwidth)**
  - **So, suitable when this bandwidth waste is tolerable e.g., where expected synchronization wait is small**
  - **For more unpredictably asynchronous situations, messier:**

    No straightforward way to awaken a waiting thread except for the waiting thread to busy−wait.

The Tera Computer System

- **Principal architect: Burton Smith (who also was responsible for the HEP)**
- **Very much a descendant of HEP**
- **Many ideas fleshed out through simulations and software development in a design called "Horizon", at:**
  Supercomputer Research Center, Maryland
- **First machines expected 1994**

The Tera Computer System



- Clock: 400 MHz (HEP: 10 MHz)
- Network: 16x16x16, sparsely populated for bandwidth
- Max 128 threads*/processor (HEP: 128)
- "Super−pipelined" processor: pipe length < 16 ticks (HEP: 8 ticks)
- Memory latency: 70 ticks avg. (HEP: 20−30 ticks)

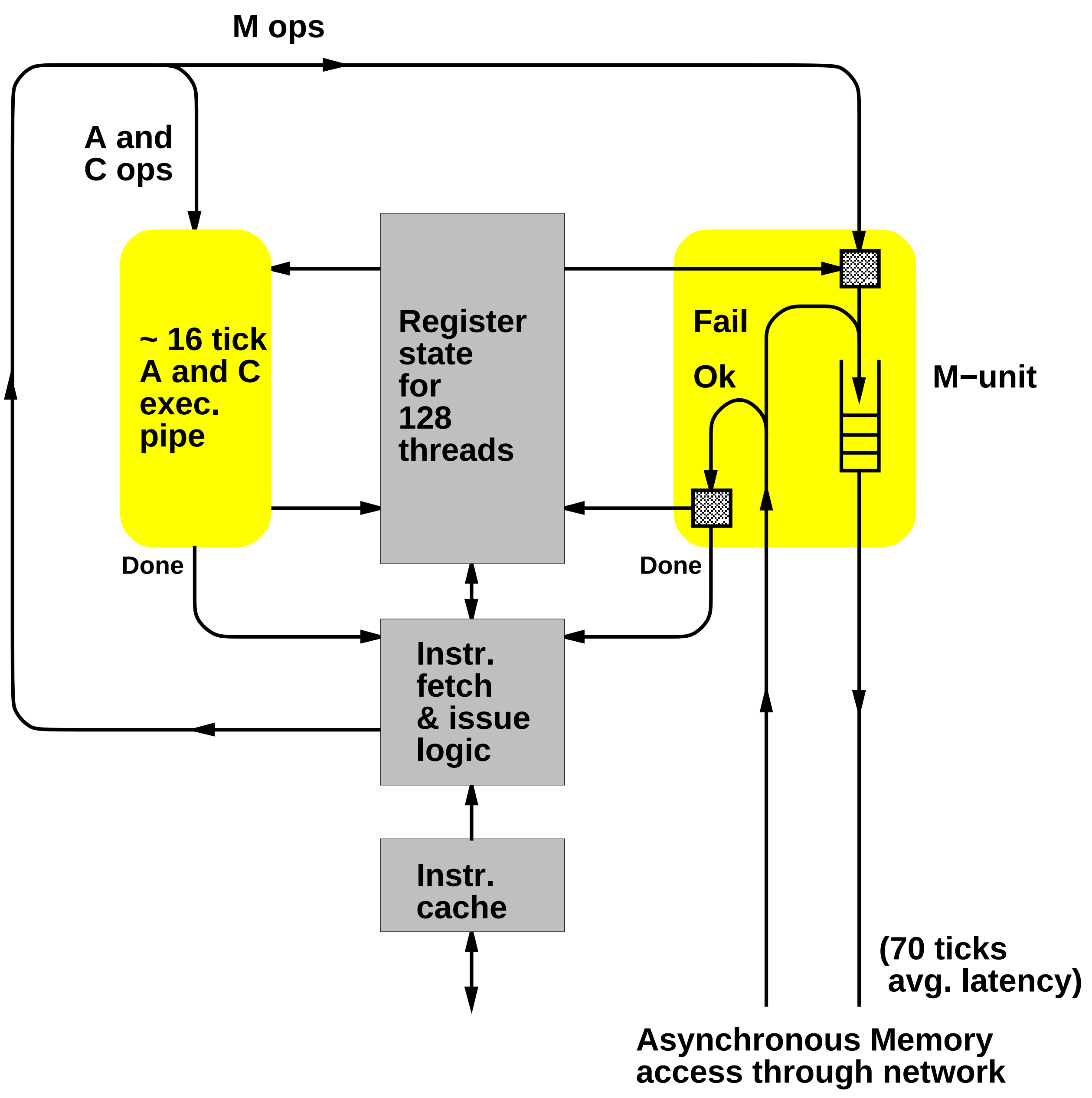
* Threads are called "streams" in the Tera

The Tera Computer System Instructions

- **Wide (moderately): 3 ops/instruction (64b)**
  - **M op (memory access)**
  - **A op (arithmetic)**
  - **C op (control or arithmetic)**

- **Therefore, peak speed (400 MHz):**
  - **1.2 G op/sec** **(HEP: 10 M op/sec)**
  - **However, a particular thread cannot issue faster than about once every 16 ticks**
    So, a thread's peak rate is about 75 M op/sec
    (HEP: 1.25 M op/sec)

The Tera Computer System Processor Organization

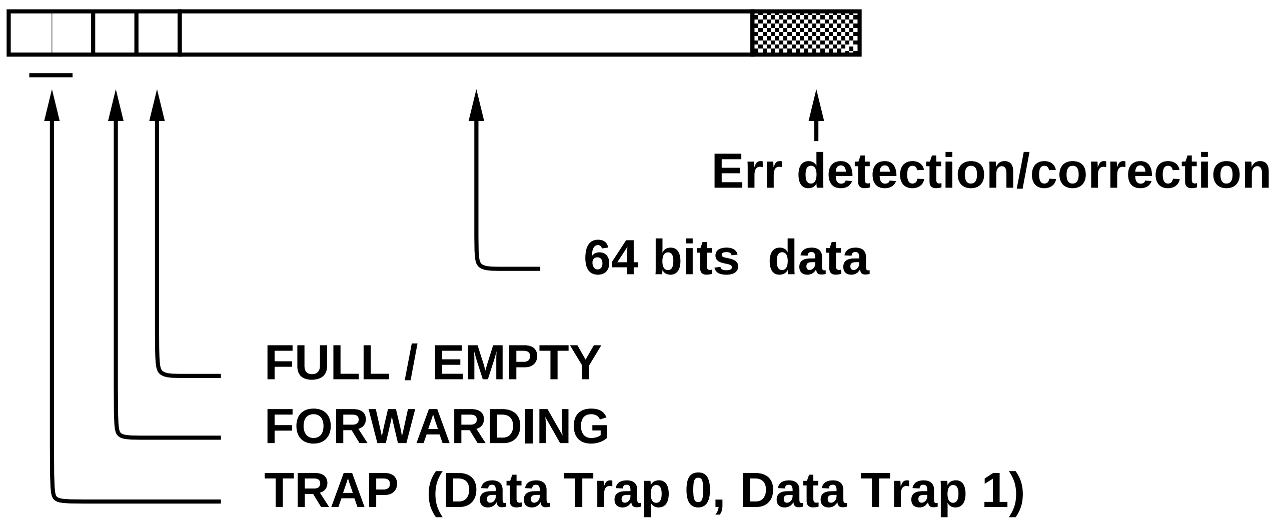

M ops
A and C ops
~ 16 tick A and C exec. pipe
Register state for 128 threads
Fail
Ok
M−unit
Done
Done
Instr. fetch & issue logic
Instr. cache
(70 ticks avg. latency)
Asynchronous Memory access through network

The Tera Computer System Memories

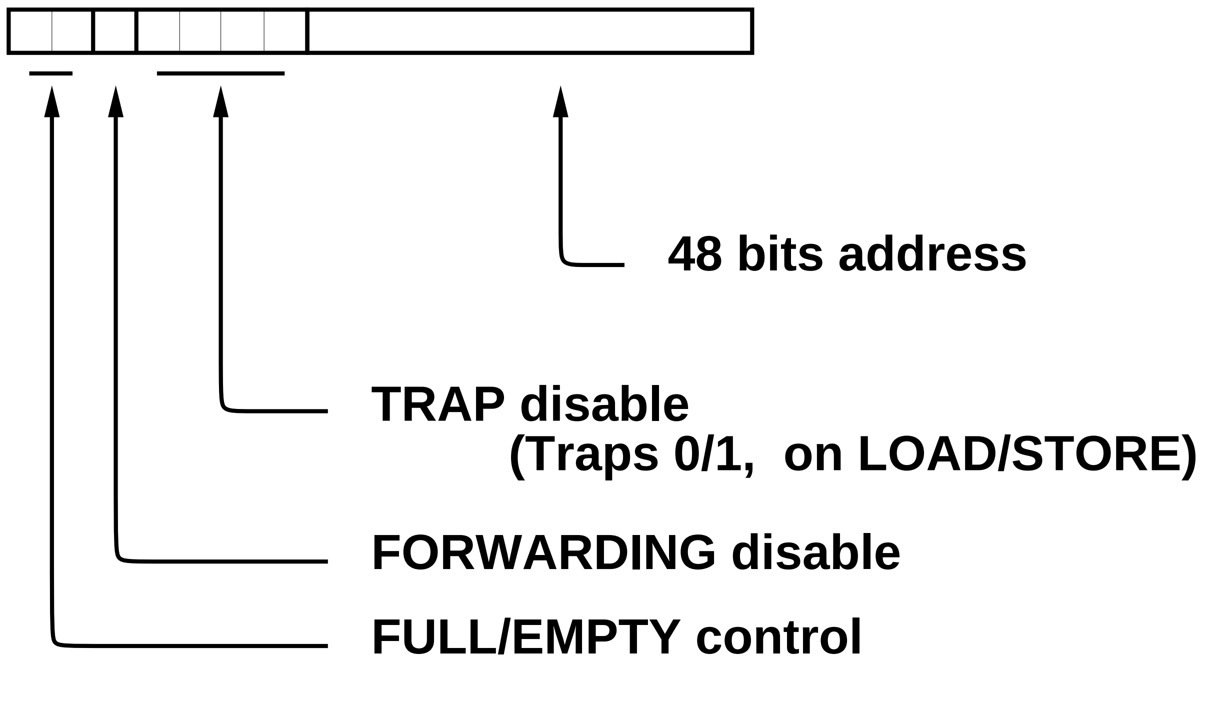
- **Memory Units:**
  - **Each 128 MB max (byte addressable)**
  - **For each processor, two "near" memory units. Used, e.g., for thread stacks**
- **I/O Cache Units:**
  - **Functionally identical to Memory, but slower, denser**
  - **Each 512 MB max**
- **Unlike HEP, no separate processor "Program Memory"**
  - **Use "nearby" I/O Cache (memory mapped)**
  - **Special access path (not through network)**
  - **Normal loads/stores only (no synchronization)**

The Tera Computer System
Memory Access

Mem Locations: Data + Access State bits



Pointers: Address + Access Control



The Tera Computer System
Memory Access

- **Access semantics: combination of**
  - **Access Control on pointer**
  - **Access State on location**
- **An access may *fail***
  - **Response includes status**
- **FULL/EMPTY Access Control**
  - **NORMAL: (ignore Access State)**
    - **STORE: set FULL**
  - **FUTURE: (one producer, multiple consumers)**
    - **LOAD: EMPTY: fail**
      FULL: succeed, leave FULL
    - **STORE: EMPTY: fail**
      FULL: succeed, leave FULL
  - **SYNC: (multiple producers and consumers)**
    - **LOAD: EMPTY: fail**
      FULL: succeed, set EMPTY
    - **STORE: EMPTY: succeed, set FULL**
      FULL: fail

The Tera Computer System Memory Access

- **DATA TRAP Access Control**
  - **Application−specific**
  - **Example uses:**
    - **Catch stack/heap overflow**
    - **Catch use of "poisoned" value (data breakpoint)**
    - **Implementing "Synchronizing Loads" with "I−structure semantics"**
    - **Triggering demand−driven evaluation**

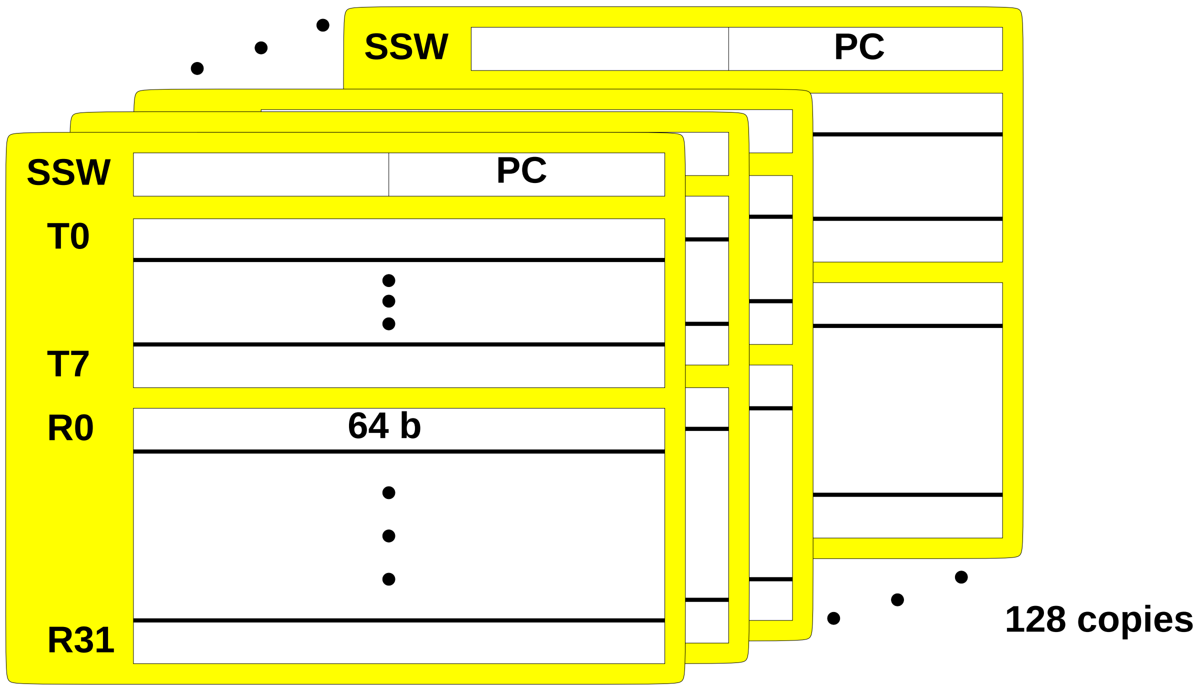
- **FORWARDING Access Control**
  - **Data in this location is pointer to another location. Invisibly forward request there.**
  - **Example uses:**
    - **Garbage collection (moving objects for compaction)**
    - **Relocation for load balancing**
  - **Subtleties:**
    - **FULL/EMPTY priority over FORWARDING**
    - **Infinite forwarding loops: trap after RETRY_LIMIT forwardings.**

The Tera Computer System Threads*

- Threads communicate only through shared memory

  (HEP: also shared registers)

- Thread−private state:



- Stream Status Word (SSW)
  - 32 bit PC (Program Counter)
  - Modes (e.g. rounding, lookahead disable)
  - Trap disable mask (e.g. data alignment, overflow)
  - Condition codes (last four emitted)
- No synchronization bits on R0−R31 (unlike HEP)
- Target Registers (T0−T7) look like PCs

*Threads are called "streams" in the Tera

The Tera Computer System
Branches and Instruction Fetching

- **Branches separated into two actions:**
  - **Load a branch Target Register (e.g., outside loop)**
  - **Branch, specifying a target register**

- **For short forward transfers (e.g., IF−THEN−ELSEs), SKIP instructions available, do not use T0 − T7**

- **Processor pre−fetches instructions from 9 sources:**
  - **PC in SSW ("current PC")**
  - **PCs in 8 Target Registers**

  so, when a Branch instruction is executed, target instructions are available

- **Instruction pre−fetching does not use network bandwidth. Instead, each processor uses special path to nearby I/O Cache Unit.**

The Tera Computer System
Explicit Dependence Lookahead

- **Each instruction has a "lookahead field"**
  - **If instruction J has lookahead = N, then instructions (J+1) ... (J+N) do not depend on the M op (memory operation) of instruction J**
  - **Register dependence handled in hardware.**

- **Normally, in interleaved pipelines, if instruction latency = P, then a particular thread cannot issue at > (1/P x clock rate).**
  - **E.g., in HEP and Monsoon, P >= 8;
    a thread's peak rate is <= 1/8 clock rate**
  - **Tera: lookahead field is 3 bits.**

    So, up to 8 mem ops from a thread may be in flight

    **So, can issue instrs. with memory accesses
    faster than the avg. memory latency (70 ticks)
    in fact, closer to processor pipe depth (< 16 ticks)**

The Tera Computer System
Explicit Dependence Lookahead

- **Lookahead fields are set by the compiler**
  - **Reminiscent of exposed pipes in early RISCs, VLIWs. However, here delay units are implementation independent (instructions, not clock ticks).**
  - **Branches:**
    - **Usually, compiler picks shorter of two lookaheads**
    - **"_OFTEN" and "_SELDOM" qualifiers on BRANCH opcodes permit longer lookaheads down one path**

- **Instruction issue logic takes into account lookahead as well as other factors (e.g., register bank scheduling)**

The Tera Computer System
Support for "Remote Loads"

- **Memory access instructions taken aside into special unit until they complete (like HEP's SFU)**
    - **Do not block other threads from executing**

- **Tera thread: upto 8 memory accesses "in flight"**
    - **HEP thread: only one pending memory access**
    - **"Namespace" for thread identifiers (used to match responses with waiting threads:**

      128 threads x 8 lookahead tags

The Tera Computer System
Support for "Synchronizing Loads"

- **FULL/EMPTY bits on all memory locations (like HEP)**
- **Failing memory accesses retried (like HEP)**
- **TERA thread: has RETRY_LIMIT (unlike HEP)**
  - **After N retries (counting forwardings), traps**
  - **Trap handler can:**
    - **Save thread state in a "CHORE" (software incarnation of a thread)**
    - **Enqueue CHORE on location, setting trap bits.**

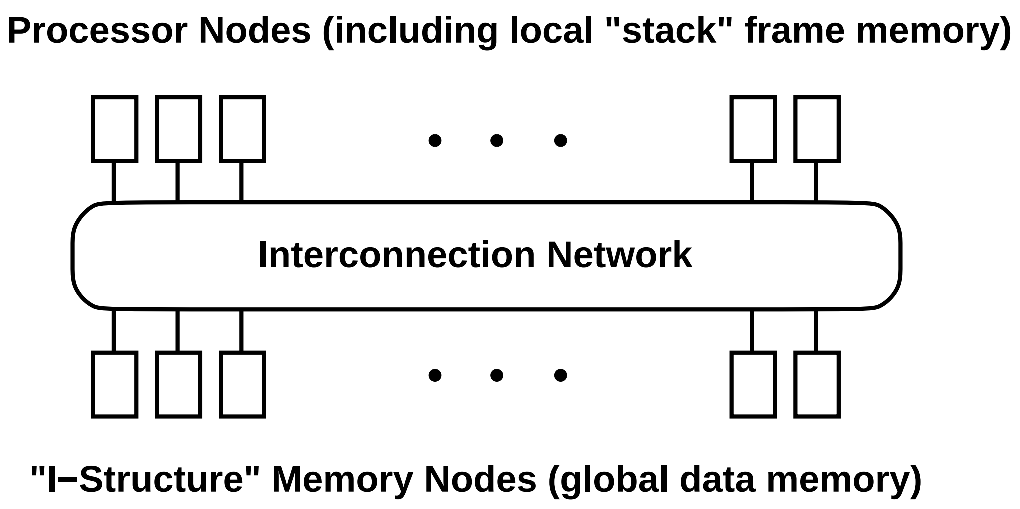
      When the producer writes, it traps;

      That trap handler awakens waiting CHOREs.
- **One RETRY_LIMIT per "Protection Domain" (collection of threads on one processor sharing a memory allocation)**

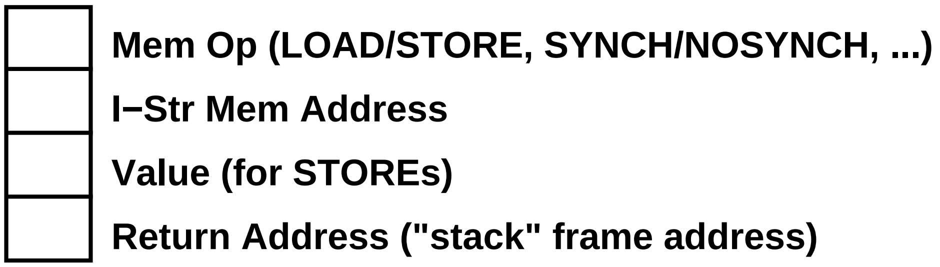
The MIT Dataflow/von Neumann Hybrid

- **Principal Architect: Robert A. Iannucci**
- **Designed, simulated for his Ph.D. thesis (MIT, 1988)**
- **Machine based on these ideas was being designed and constructed at IBM Research under the direction of Iannucci (the Empire project) from 1988 to 1991, when it was abruptly cancelled (for non−technical reasons)**

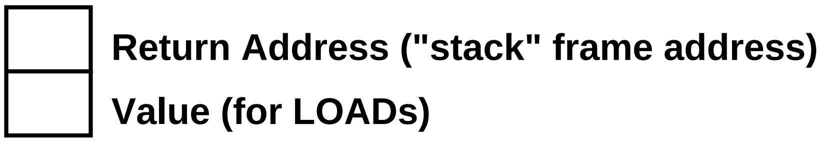
The MIT Dataflow/von Neumann Hybrid Global Organization


Processor Nodes (including local "stack" frame memory)
Interconnection Network
"I−Structure" Memory Nodes (global data memory)

- Processor to I−Structure Memory interaction:
  - Dataflow model (Arvind et. al.)
  - Every I−Str Memory Location has FULL/EMPTY bits
  - Request packet (Processor to I−Str Memory):


Mem Op (LOAD/STORE, SYNCH/NOSYNCH, ...)
I−Str Mem Address
Value (for STOREs)
Return Address ("stack" frame address)

  - Response packet (I−Str Mem to Processor):

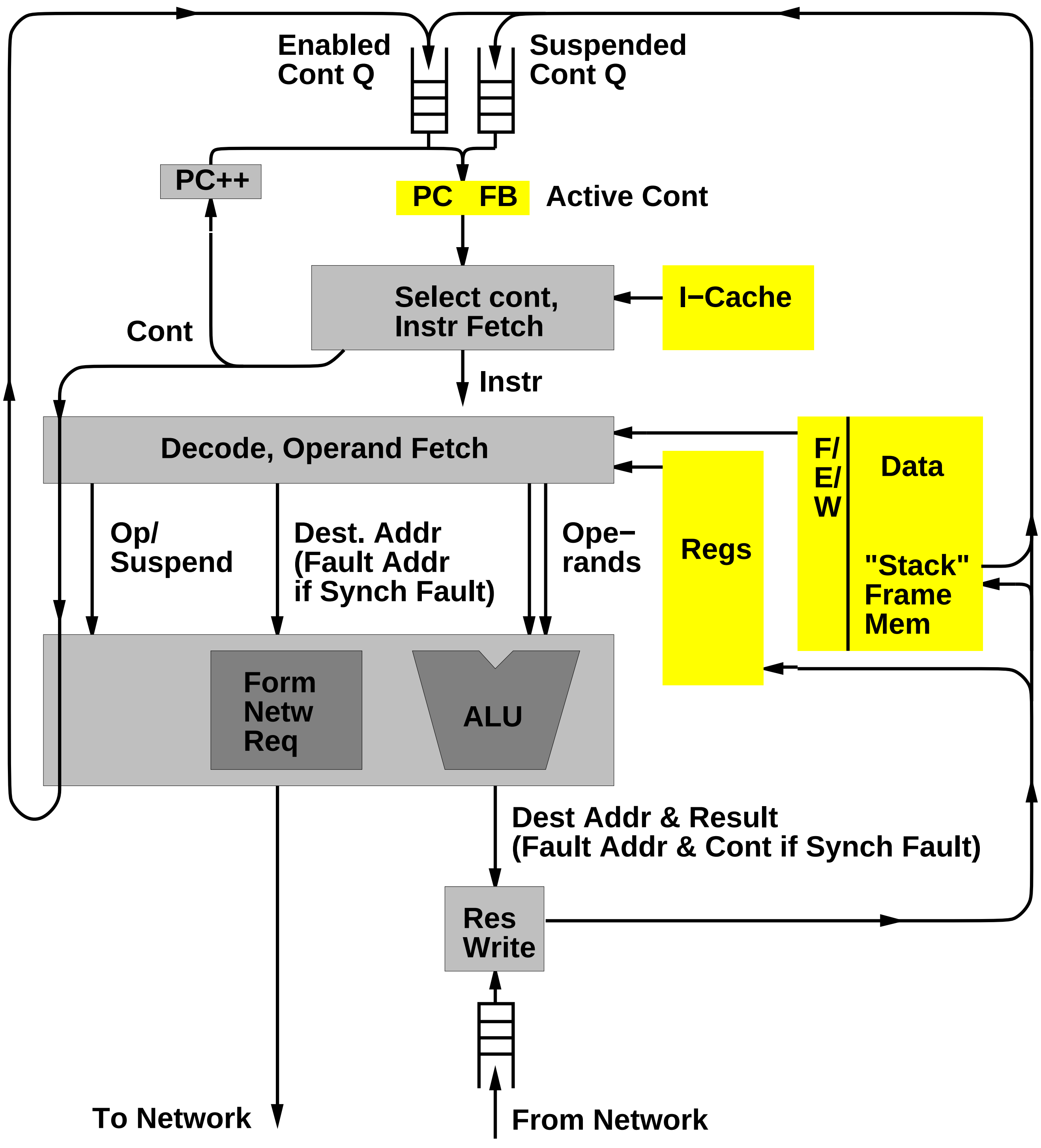


- On synchronizing accesses, I−Structure Memory may hold on to request until satisfiable

The MIT Dataflow/von Neumann Hybrid Processor Organization (simplified)

- "Continuations": Program Counter, Frame Base

  Continuations fit into one word



The MIT Dataflow/von Neumann Hybrid Processor Operation

- **Frame locations have FULL/EMPTY/WAITING bits**
  - **Read with normal or synchronizing semantics**
- **Fetch instruction specified by Active Continuation's PC**

If
  - **Is not a thread termination**
  - **Is not a branch**
  - **Does not involve synchronizing frame read**

Then: **Update Active Cont by incrementing PC**

Else: **Reload Active Cont from Enabled Cont Queue (if available, else Suspended Cont Queue)**

- **Fetch operands from registers/frame location**
  - **If synch fault (reading EMPTY frame location) pass this information downstream**
- **Result Write:**
  - **Normal: write data to register/frame location**

    If frame location is WAITING,
    - **Pop waiting continuation into Enabled Cont Queue**
  - **Synch fault, frame location EMPTY**
    - **Write continuation into frame, mark WAITING**
  - **Synch fault, frame location already WAITING**
    - **Enter continuation into Suspended Cont Queue**

The MIT Dataflow/von Neumann Hybrid Remote and Synchronizing Loads

- **"Remote Loads":**
  - **Initialize frame slot to EMPTY**
  - **Issue remote load (I−structure memory to frame slot)**
  - **Continue execution; if attempt to read frame slot before response arrives, suspend and wait there**
  - **Generalizes naturally to multiple remote loads**

- **"Synchronizing Loads: Handled at I−structure memory**

- **Registers not replicated, not saved on suspension.**
  - **Compiler cannot rely on register values beyond branches or potentially suspensive instructions**

- **May busy−wait (via Suspended Cont Queue) if more than one thread (continuation) has synch failure on same frame slot, because only one continuation can be saved in slot for later awakening by producer**

The MIT J−Machine

- **Principal architect: Bill Dally (MIT)**

- **Constructed in collaboration with Intel**
  Prototype boards running since July 1991

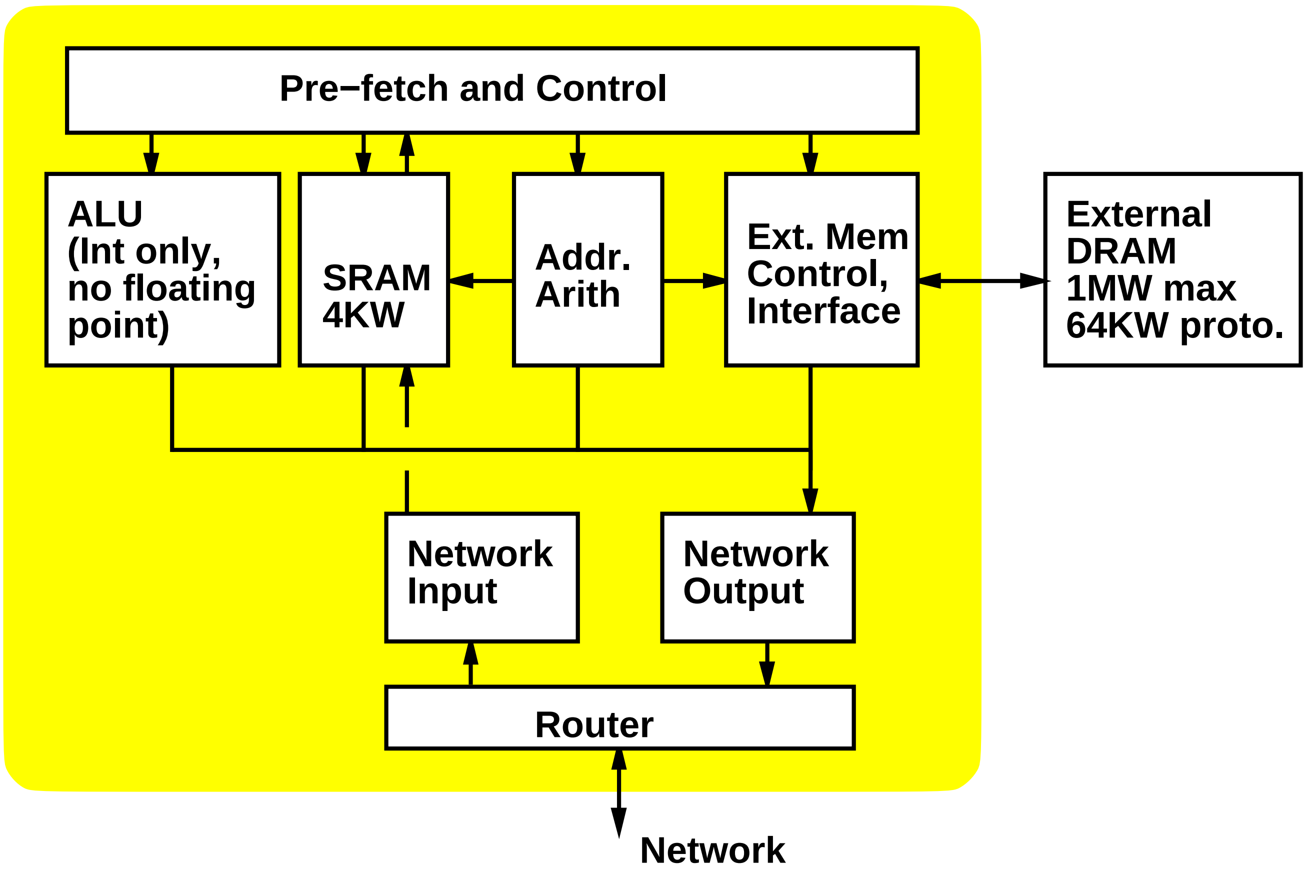
- **3D Mesh of up to 64K identical nodes**
  - **Prototype: up to 4K nodes**

- **Each node: two main components**
  - **an MDP (Message Driven Processor) chip (including part of network, external DRAM controller)**
  - **External DRAM**

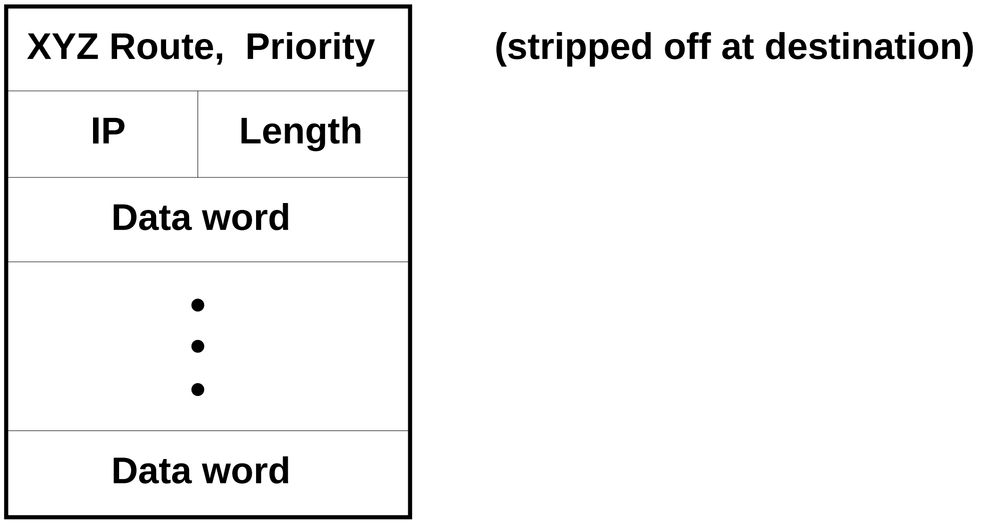
- **J−Machine: so called because nodes are like "jellybeans": cheap, plentiful**

The MIT J−Machine
The MDP (Message Driven Processor)

- 36 bit words (32 bit data + 4 bit type tag)
- Semi−custom (standard cell), 1.1 M transistors
- 20 MHz
- Simplified block diagram:


Pre−fetch and Control
ALU (Int only, no floating point)
SRAM 4KW
Addr. Arith
Ext. Mem Control, Interface
External DRAM 1MW max 64KW proto.
Network Input
Network Output
Router
Network

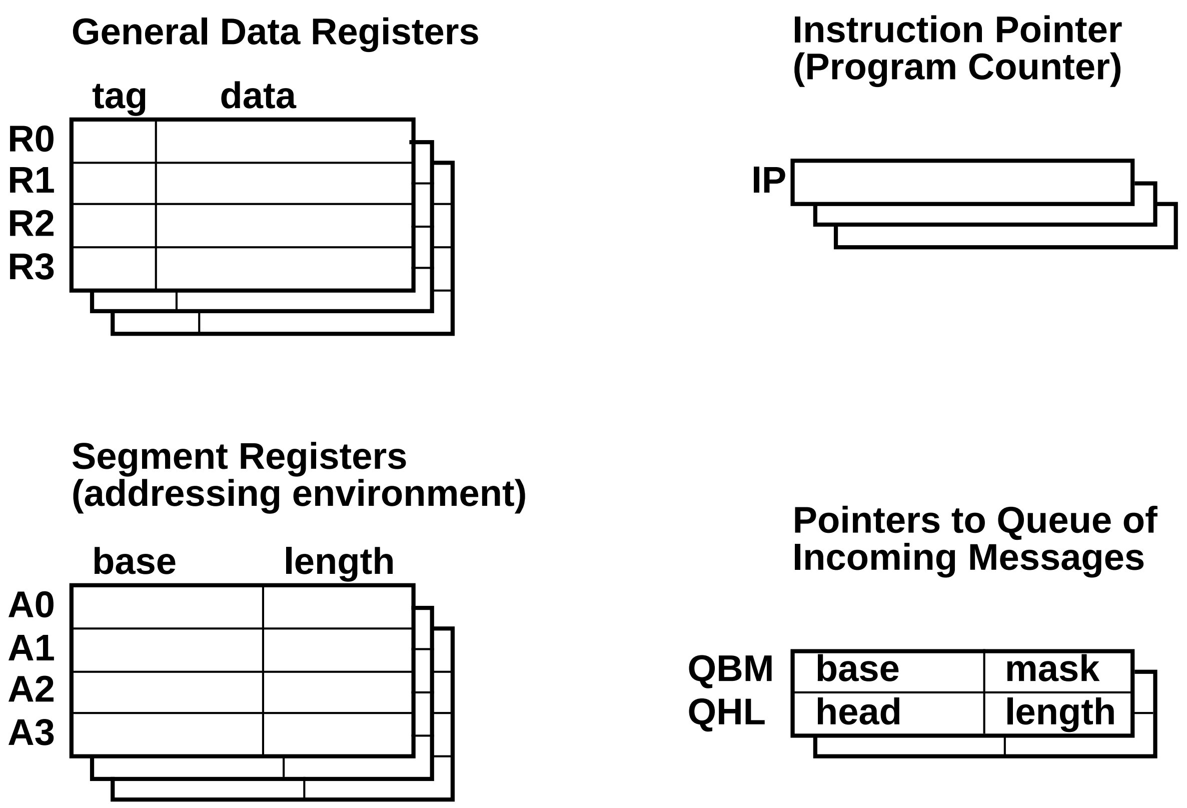
The MIT J−Machine Messages



- Variable length
- Only 1st word (XYZ) interpreted by network
  - Constructed automatically by network interface during message−send, from absolute node number
  - Priority (0 or 1)
    - Specifies one of two MDP queues at destination in which to place the message
    - Message priority independent of priority of sender
- 2nd word: interpreted by destination MDP as

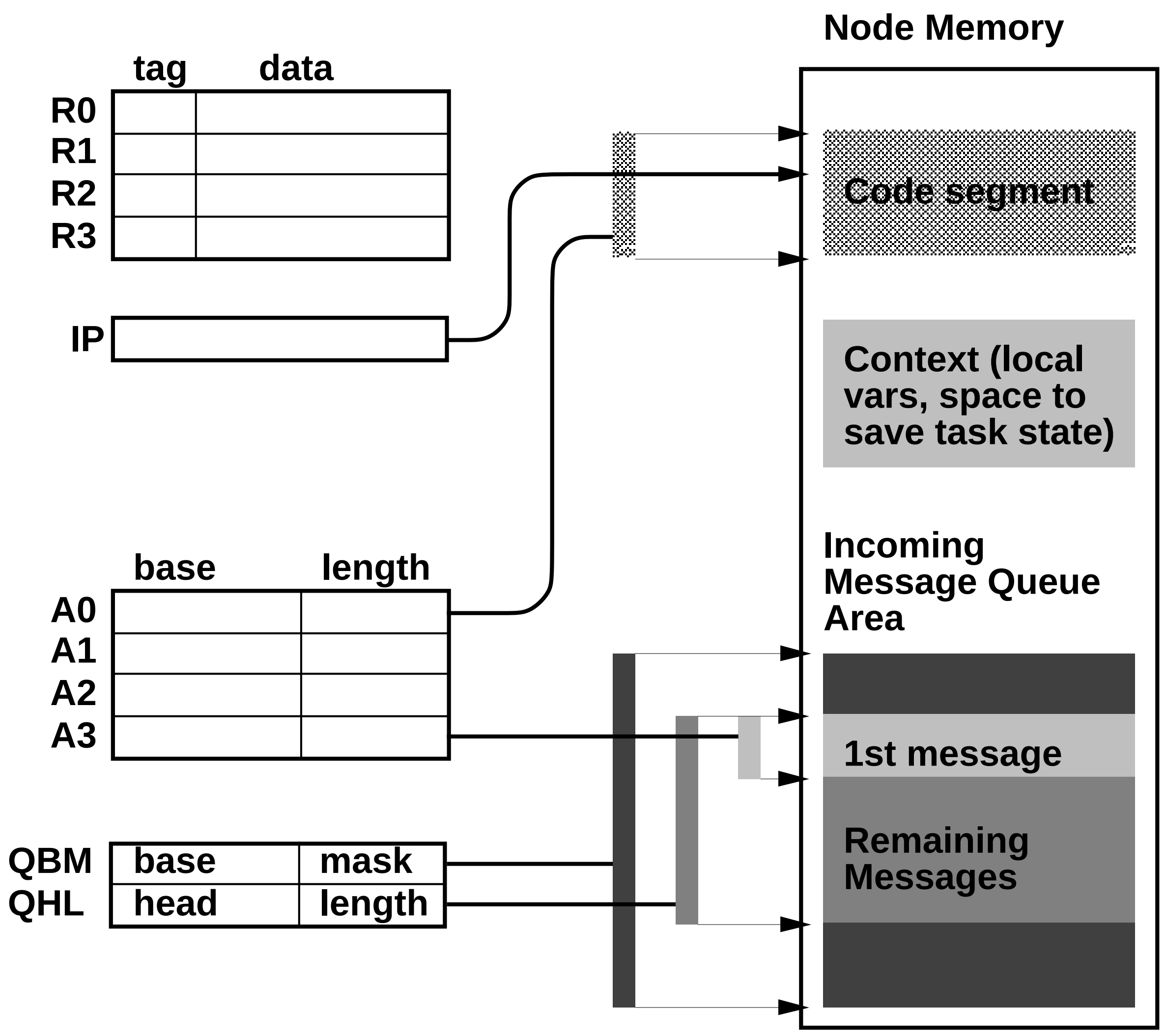
  Instruction Pointer and Length

The MIT J−Machine MDP Processor State (simplified)



- 3 copies of most processor state; one each for:
  - Priority 1, Priority 0, Background
- MDP executes at:
  - Priority 0 or 1, for priority 0 or 1 messages, resply.
  - Background priority, if no pending messages
- Higher priority tasks preempt
  - State of lower priority task need not be saved

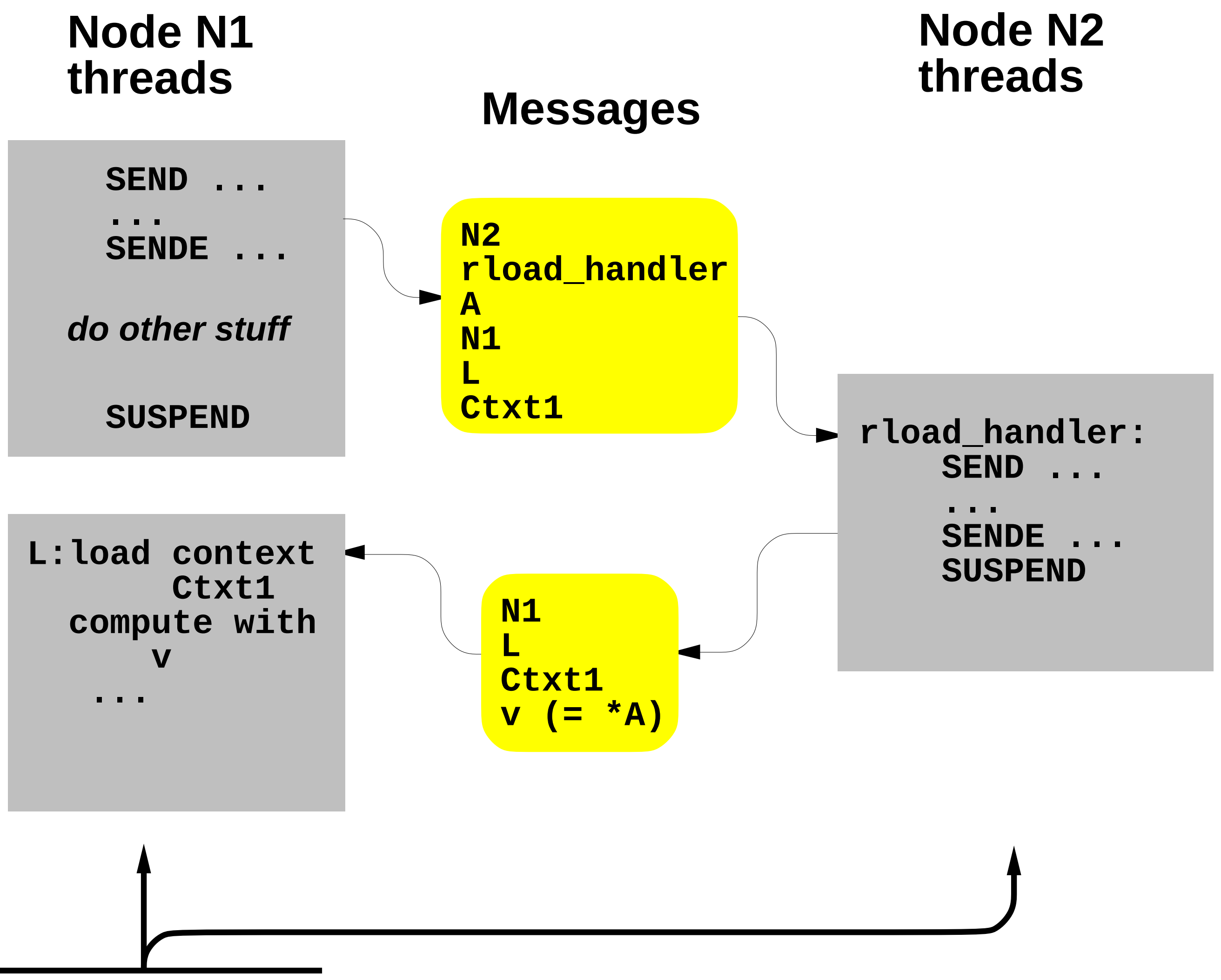
The MIT J−Machine Task State (simplified)



- **Message execution begins when:**
  - **Message arrives at higher priority than current task**
  - **Task executes SUSPEND instruction, and there is a pending message at same priority**
- **In one tick:**
  - **IP loaded from first word of message**
  - **A3 loaded to point at message**

The MIT J−Machine Support for "Remote Loads"

***Solution 1:** (inspired by P−RISC/*T dataflow model; it's not the method normally used)*

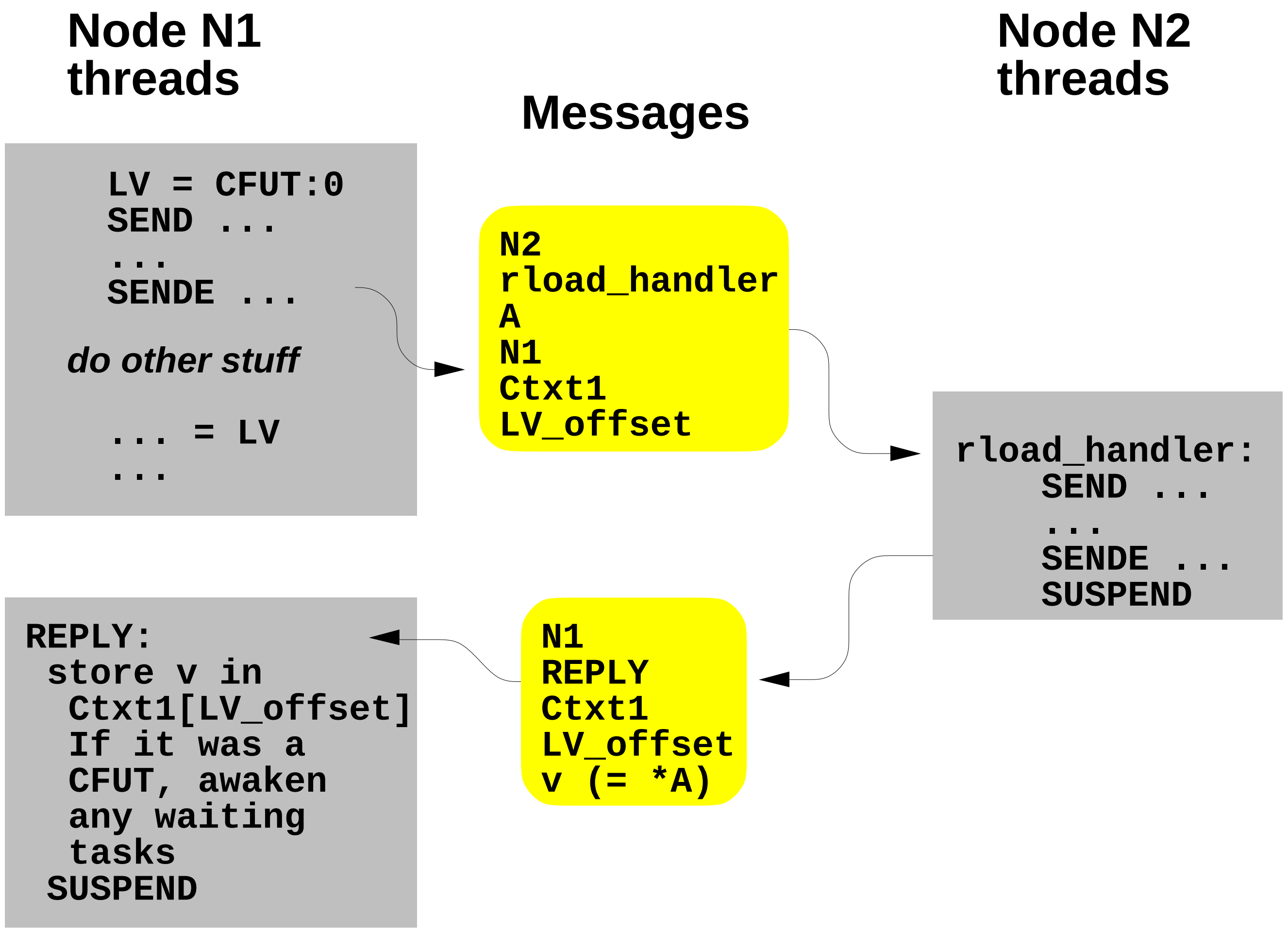


Latency issue:

- **Handlers execute only after all pending messages at same or higher priority**
- **Possible solution: Use Priority 1 messages and handlers for remote loads**
- **However, Priority 1 usually reserved for OS**

The MIT J−Machine
Support for "Remote Loads"

Solution 2: Normal method on the J−Machine



- LV: initialized with CFUT type tag ("context future")
- Attempt to load from LV ("...=LV") causes trap
- Trap handler:
  - saves task state in CTXT1 (registers, message)
  - enqueues CTXT1 at location LV
- The REPLY handler later resumes the task

The MIT J−Machine
Support for "Remote Loads"

Multiple Remote Loads (the "Join Problem")

- **For Solution 1:**
  - **Either do remote loads one at a time**
  - **Or:**
    - **Initialize a "join counter" J to 0**
    - **Issue N remote loads together, each with its own reply label L1, L2, ..., LN**
    - **At each reply Li, increment J and check if = N**
      - **If not, kill the thread**
      - **If yes, continue with sequel**

- **Solution 2 generalizes directly to multiple remote loads**
  - **Initialize all destination locations to CFUTs**
  - **Issue N remote loads together**
  - **Continue with sequel, trapping and suspending as necessary**

The MIT J−Machine
Support for "Synchronizing Loads"

- **CFUT technique can be used at Node N2 to handle Synchronizing Loads, by simulating I−structure semantics**

- **Suppose location A (in Node N2) has been initialized to be a CFUT**

- **When rload_handler attempts to read, it traps and is suspended there**

- **When producer finally writes into A, it checks for suspended tasks and awakens them**

- **rload_handler finally completes, and sends reply**

- **Note: no busy−waiting, no extra messages**

  (like Hybrid, unlike HEP/Tera/Alewife)

The MIT J−Machine
Support for Global Addressing

- **Combination of software and hardware**
- **Memory addresses are node−local**
- **Object, when allocated, is given unique id:**

  <Node #, Object #>
- **Inter−object references use unique ids**
- **An object may move to another node, or be replicated on several nodes**
- **Software maintains something like a "directory−based cache coherence" mechanism:**
  - **Every node maintains associative table mapping object ids to node numbers**
  - **An object's "home node" (where originally allocated) always knows exactly where it is**
  - **Other nodes have "hints" about where it is**
- **Hardware assist:**
  - **A section of MDP on−chip memory supports asso− ciative lookup (used to cache lookup table)**
  - **Instructions to insert/lookup/probe this section**

Multithreading: A von Neumann Story Final Comments

- **HEP, Tera:**
  - **Replicate conventional instruction stream**
  - **Synchronizing loads: hardware (trap) and software**
  - **Interleaved processor pipeline**
- **Hybrid:**
  - **Replicate conventional instruction stream,**
  - **But, do not preserve registers across threads**
  - **Synchronizing loads: entirely in hardware**
  - **Non−interleaved pipe, but branches and global loads break threads.**
- **The MIT J−Machine**
  - **Support for 3 instruction streams (priorities)**
  - **Grew out of message−passing machines (non−shared memory model), but added support for global addressing**
  - **Remote, Synchronizing loads: software convention**

This page intentionally blank

This page intentionally blank

This page intentionally blank