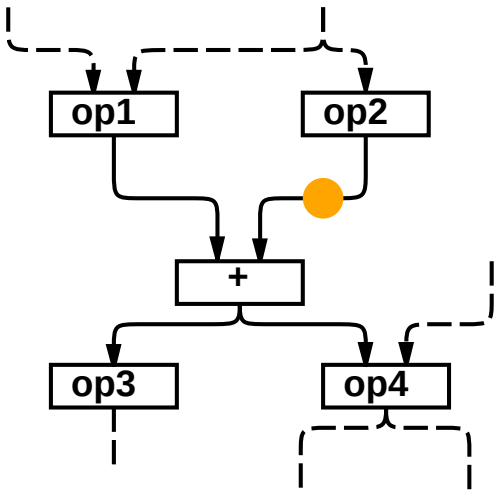
Encoding Dataflow Graphs and Tokens

Conceptual



Encoding of graph

Program memory:

	Op−code	Destination(s)
109	op1	120L
113	op2	120R
120	+	141, 159L
141	op3	...
159	op4	... , ...

Encoding of token:

A "packet" containing:

120R	Destination instruction address, Left/Right port
6.847	Value

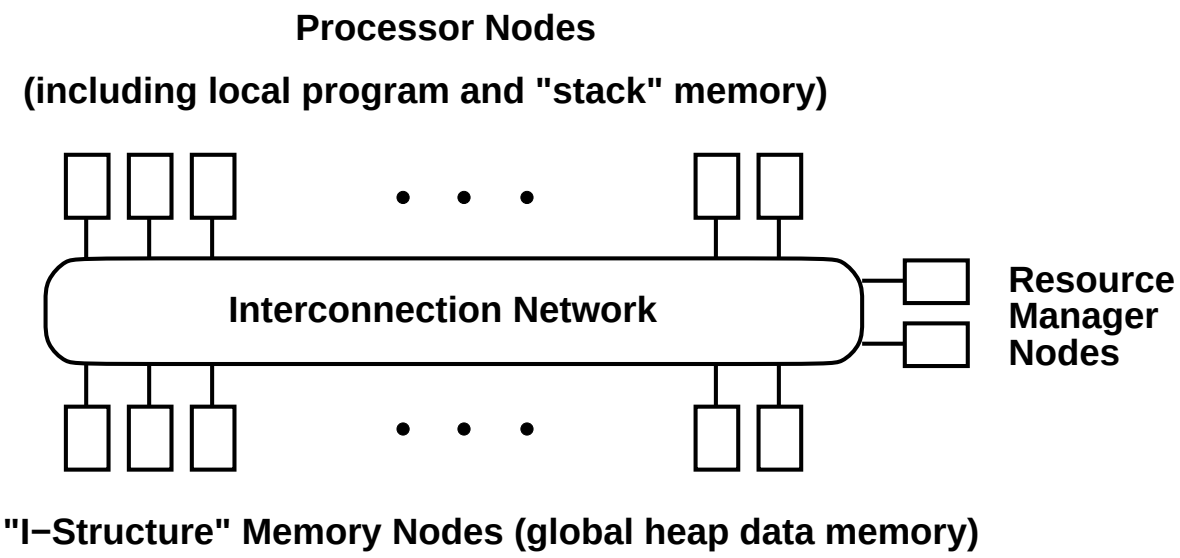
Re−entrancy ("dynamic" dataflow):

- Each invocation of a function or loop iteration gets its own, unique, "Context"
- Tokens destined for same instruction in different invocations are distinguished by a context identifier

120R	Destination instruction address, Left/Right port
Ctxt	Context Identifier
6.847	Value

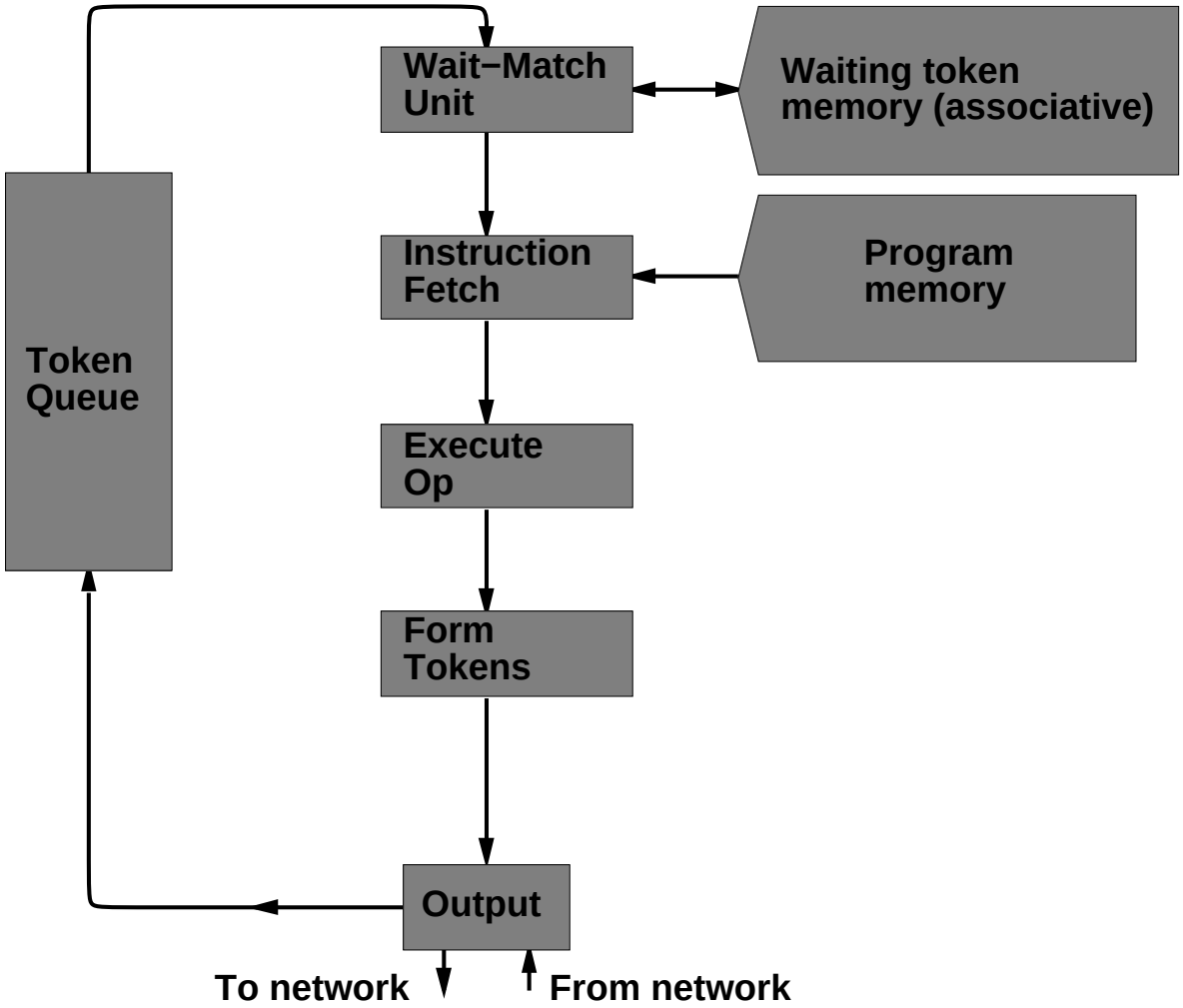
MIT Tagged Token Dataflow Architecture

- Designed by Arvind et. al., MIT, early 1980's
- Simulated; never built
- Global view:



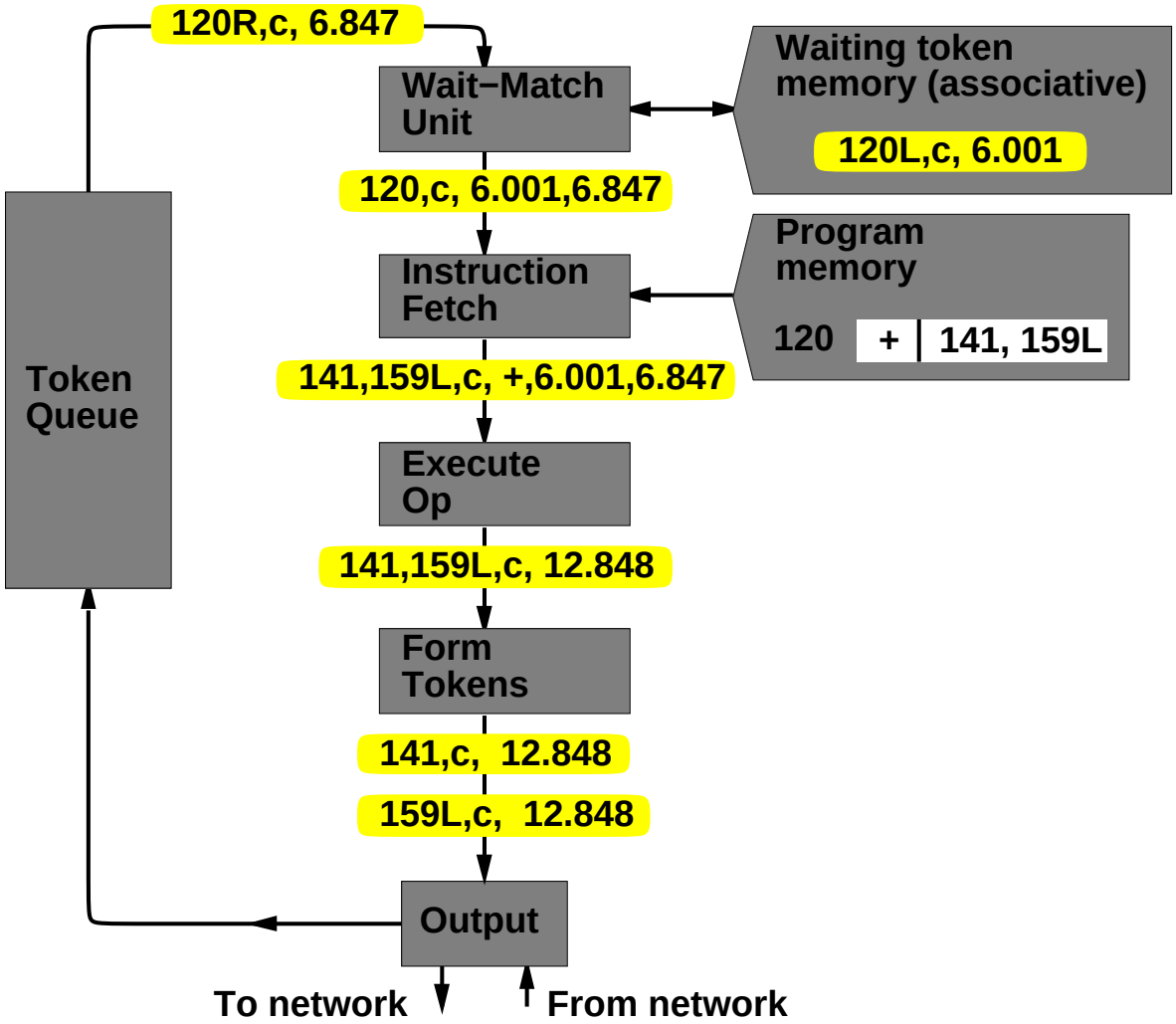
- Resource Manager Nodes responsible for
  - Function allocation (allocation of context identifiers)
  - Heap allocation
  - etc.
- Stack memory and heap memory: globally addressed

MIT Tagged Token Dataflow Architecture Processor



- **Wait−Match Unit:**
  - **Tokens for unary ops go straight through**
  - **Tokens for binary ops: try to match incoming token and a waiting token with same instruction address and context id**
    - **Success: Both tokens forwarded**
    - **Fail: Incoming token −−> Waiting Token Mem, Bubble (no−op) forwarded**

MIT Tagged Token Dataflow Architecture Processor Operation

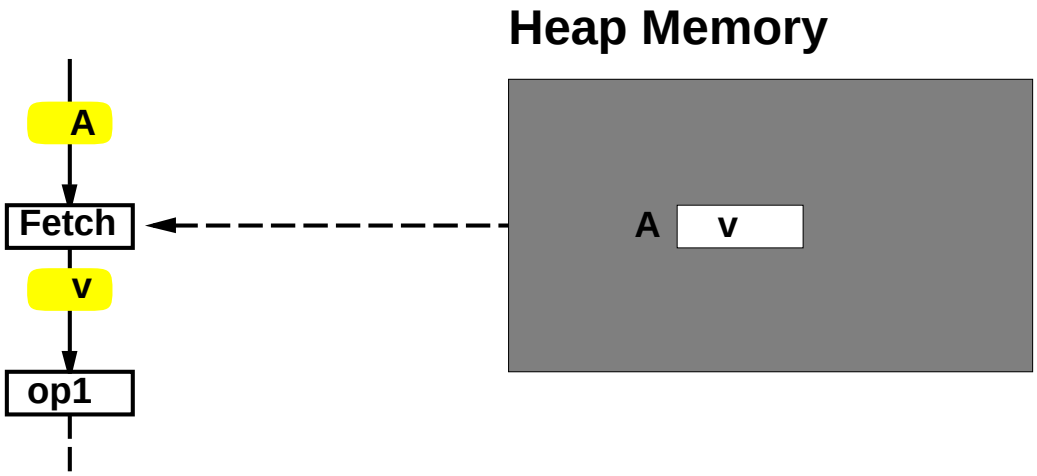


- Output Unit routes tokens:
  - Back to local Token Queue
  - To another Processor
  - To heap memory

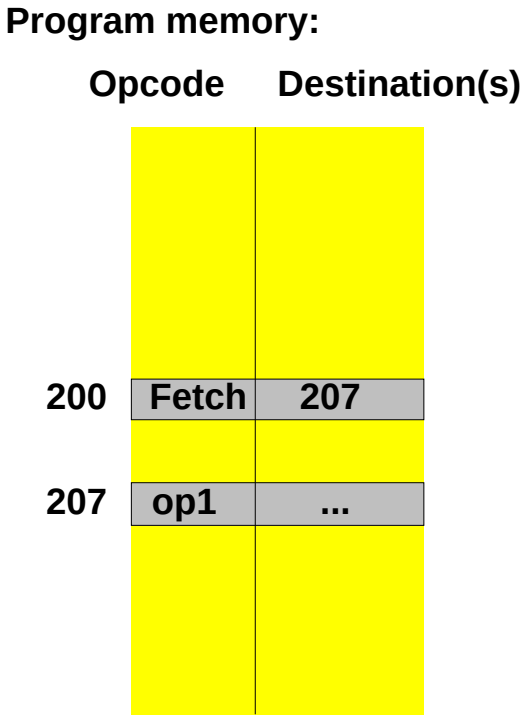
  based on the addresses on the token
- Tokens from network are placed in Token Queue

MIT Tagged Token Dataflow Architecture Support for "Remote Loads"

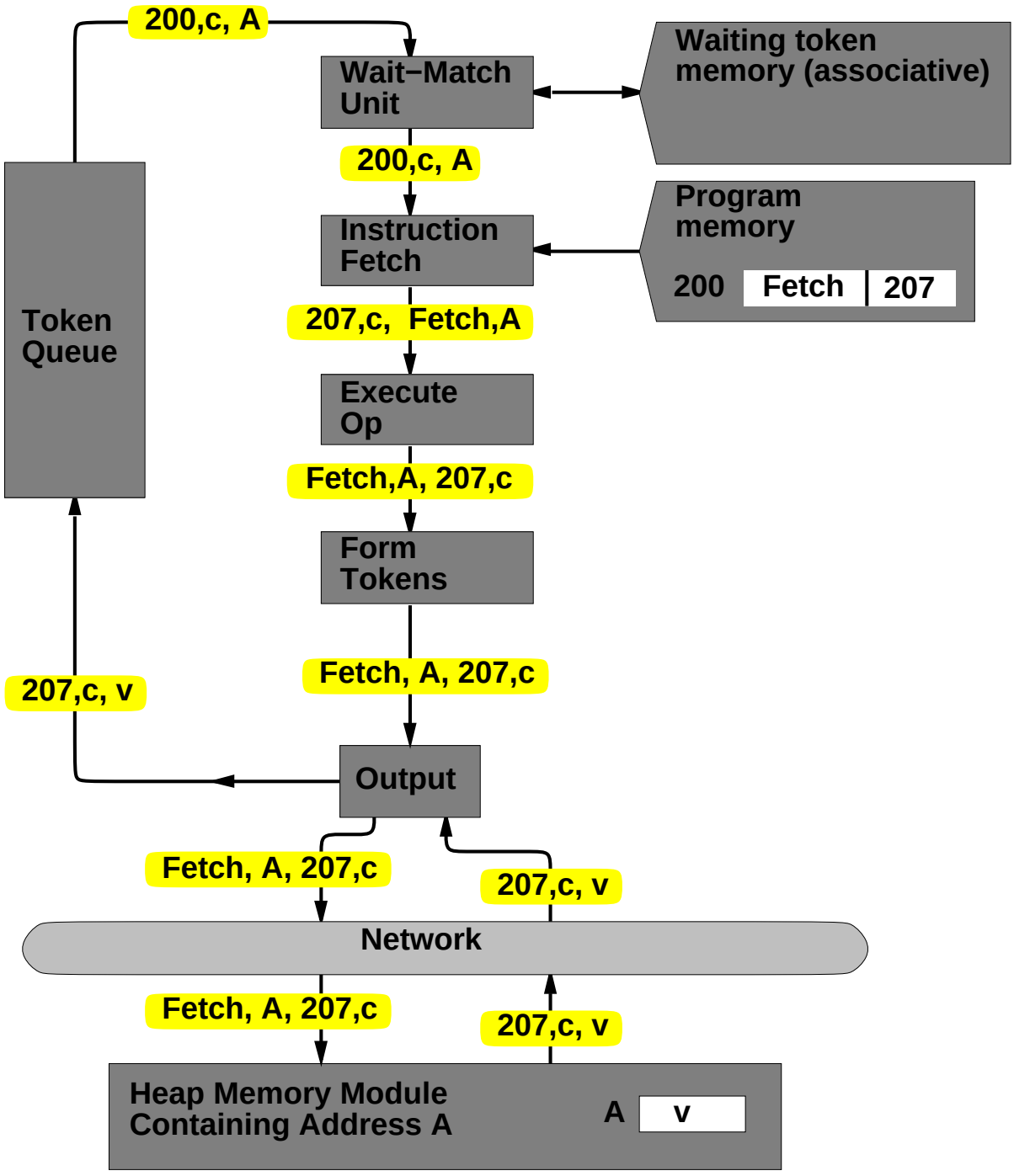
Conceptual:



Encoding of graph:



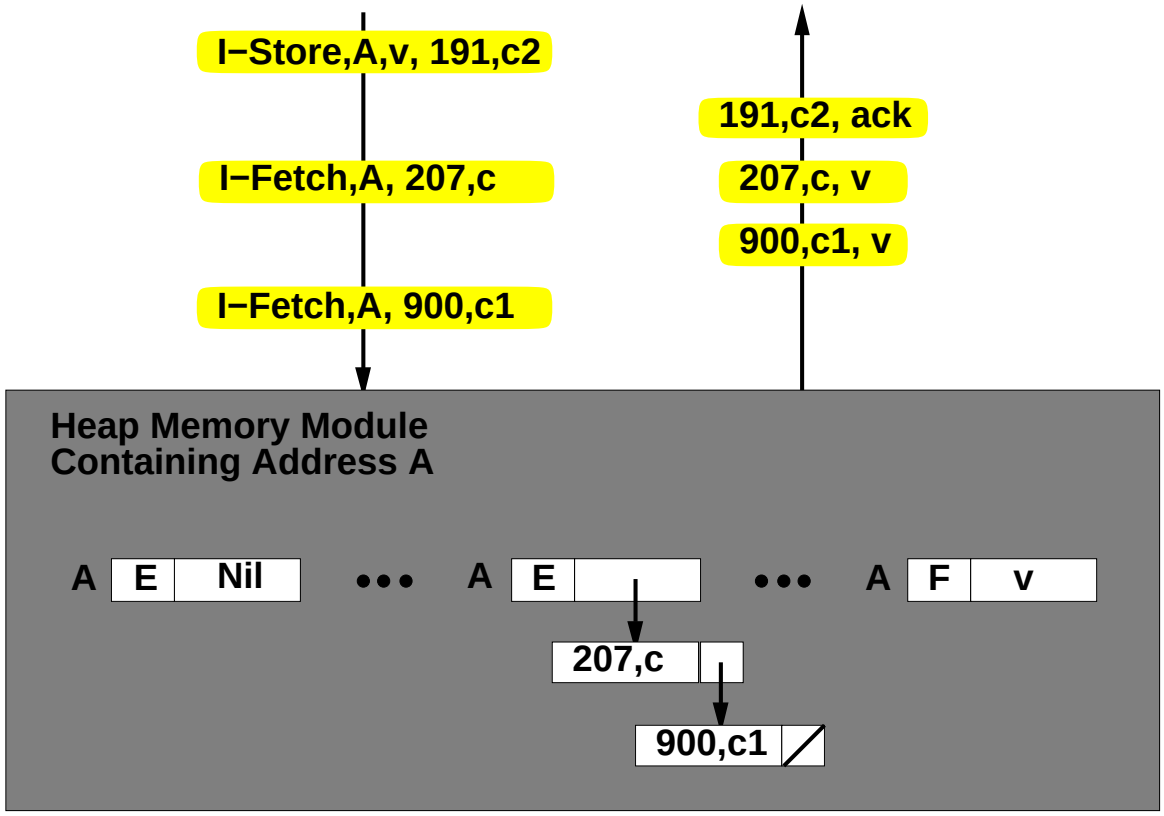
MIT Tagged Token Dataflow Architecture Support for "Remote Loads"



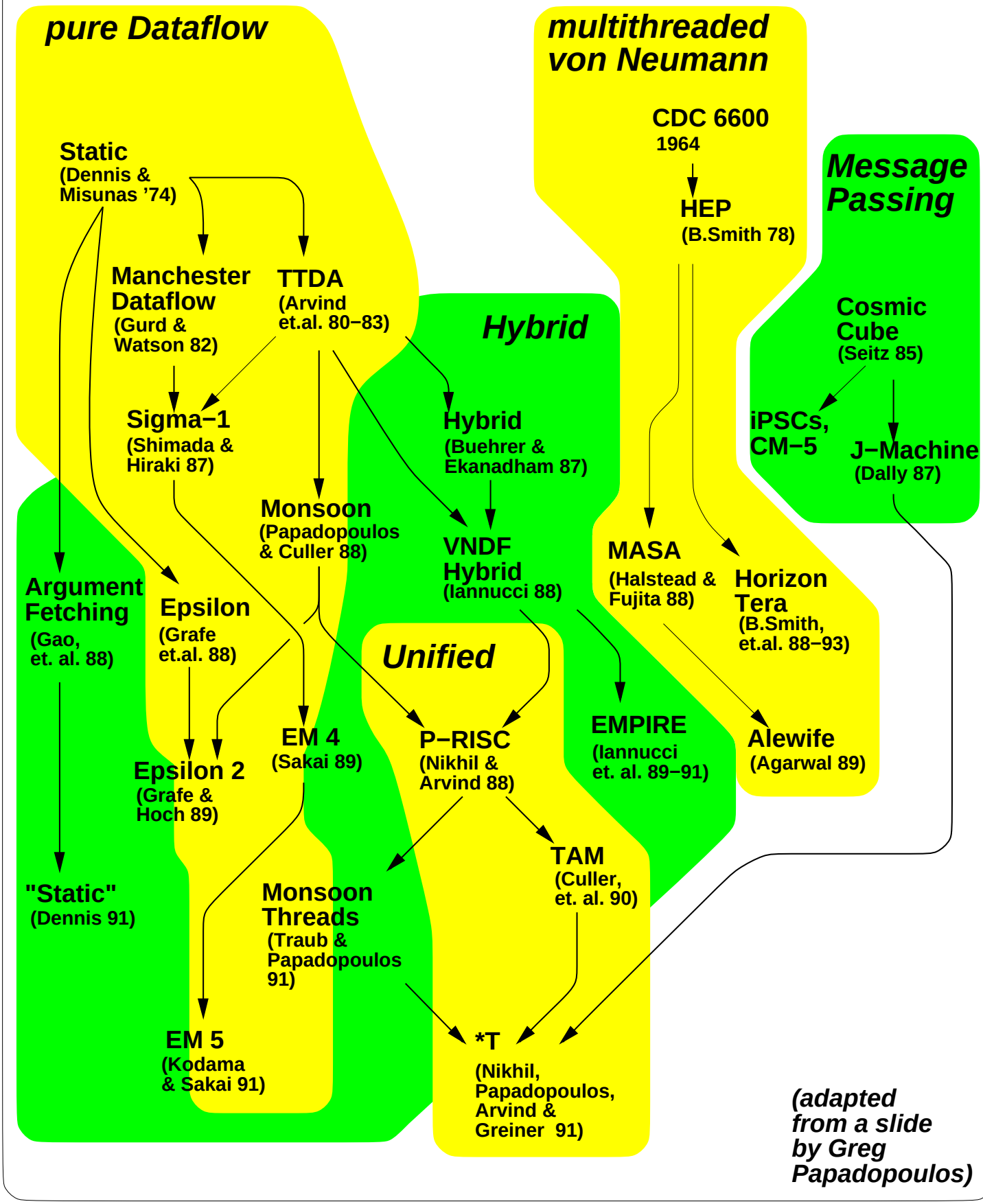
- Multiple remote loads are no problem:
  - Can be issued in parallel
  - "Join" of responses is implicit in Wait−Match

MIT Tagged Token Dataflow Architecture Support for "Synchronizing Loads"

- Heap memory locations have FULL/EMPTY bits
- When "I−Fetch" request arrives (instead of "Fetch"), if the heap location is EMPTY, heap memory module queues request at that location
- Later, when "I−Store" arrives, pending requests are discharged
- "I−structure semantics"
  Note: no busy waiting, no extra messages



Unification of Dataflow & von Neumann Designs



(adapted from a slide by Greg Papadopoulos)