Dataflow: Passing the Token

Arvind
Computer Science and Artificial Intelligence Lab
M.I.T.

ISCA 2005, Madison, WI
June 6, 2005

Inspiration: Jack Dennis

General purpose parallel machines based on a dataflow graph model of computation

Inspired all the major players in dataflow during seventies and eighties, including Kim Gostelow and I @ UC Irvine

EM4: single-chip dataflow micro

Da

Sigma-1: The largest

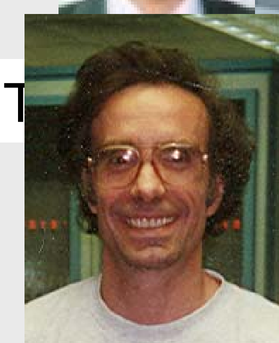
flow m n

St

Dy

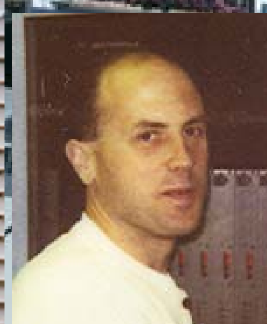
T

Greg Papadopoulos



Andy Boughton

Chris Joerg



Jack Costanza

Monsoon

Software Influences

- Parallel Compilers
  - Intermediate representations: DFG, CDFG (SSA, ϕ ,...)
  - Software pipelining

  Keshav Pingali, G. Gao, Bob Rao, ..
- Functional Languages and their compilers
- Active Messages

  David Culler
- Compiling for FPGAs, ...

  Wim Bohm, Seth Goldstein...
- Synchronous dataflow
  - Lustre, Signal

  Ed Lee @ Berkeley

This talk is mostly about MIT work

- Dataflow graphs
  - A clean model of parallel computation
- Static Dataflow Machines
  - Not general-purpose enough
- Dynamic Dataflow Machines
  - As easy to build as a simple pipelined processor
- The software view
  - The memory model: I-structures
- Monsoon and its performance

- Musings

Dataflow Graphs

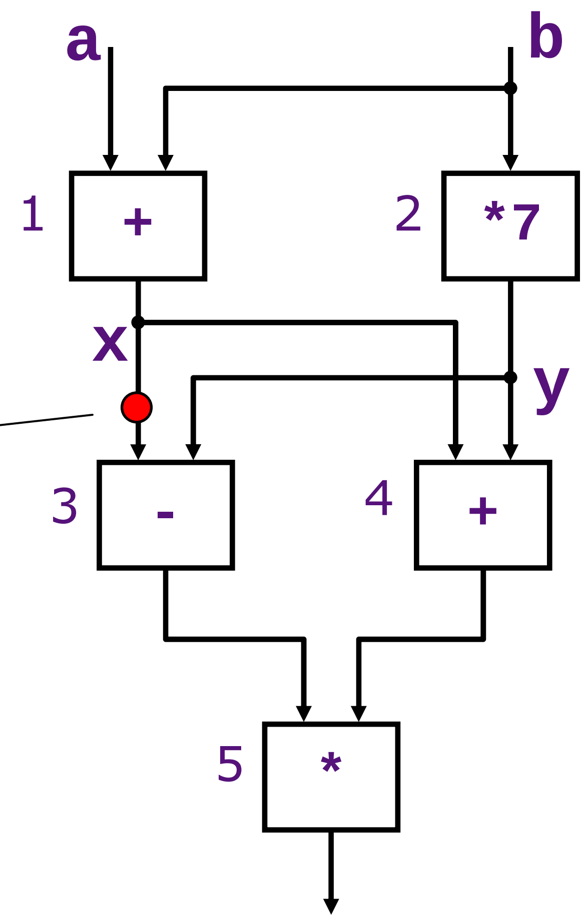
```
{x = a + b;
 y = b * 7
in
    (x-y) * (x+y)}
```

- Values in dataflow graphs are represented as tokens

token < ip , p , v >

instruction ptr — port — data

ip = 3
p = L

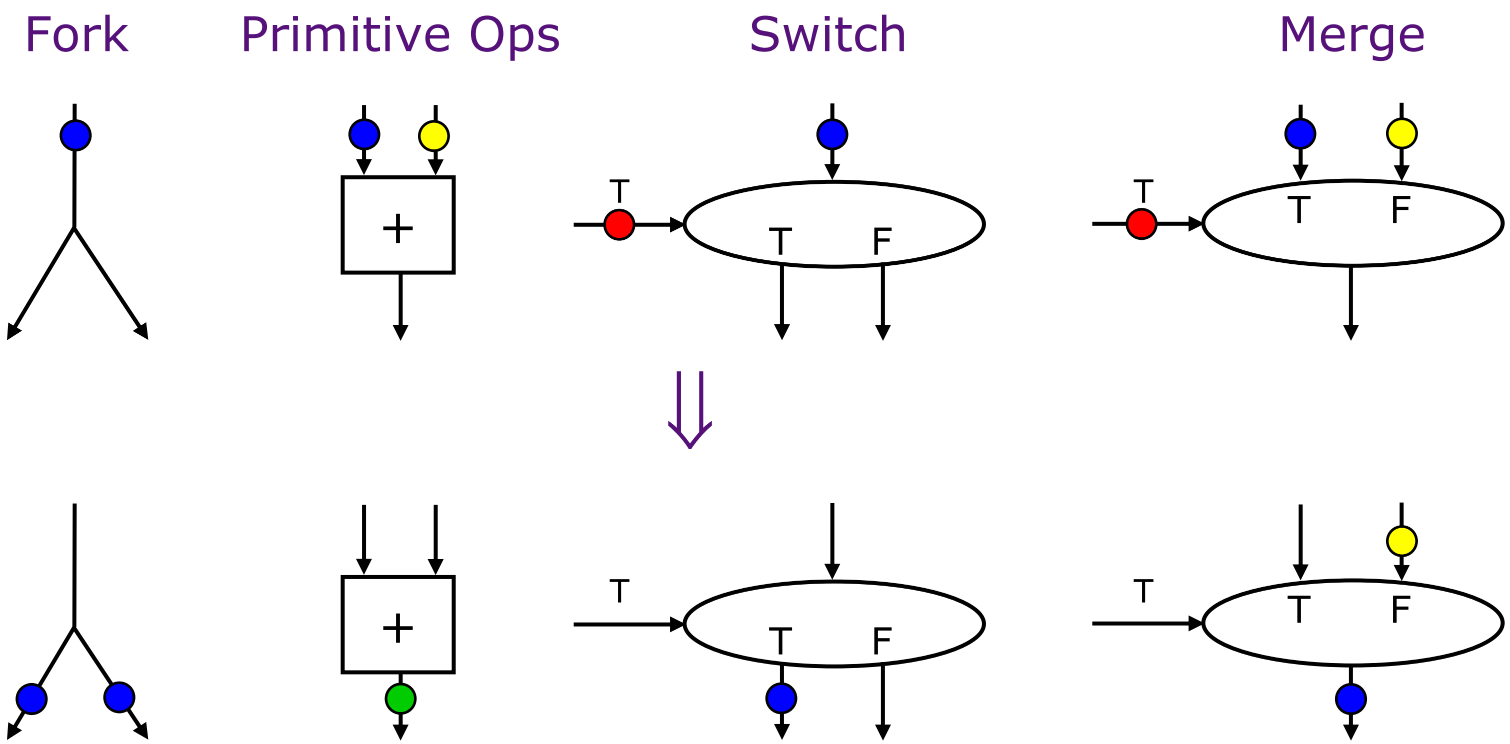


- An operator executes when all its input tokens are present; copies of the result token are distributed to the destination operators

no separate control flow

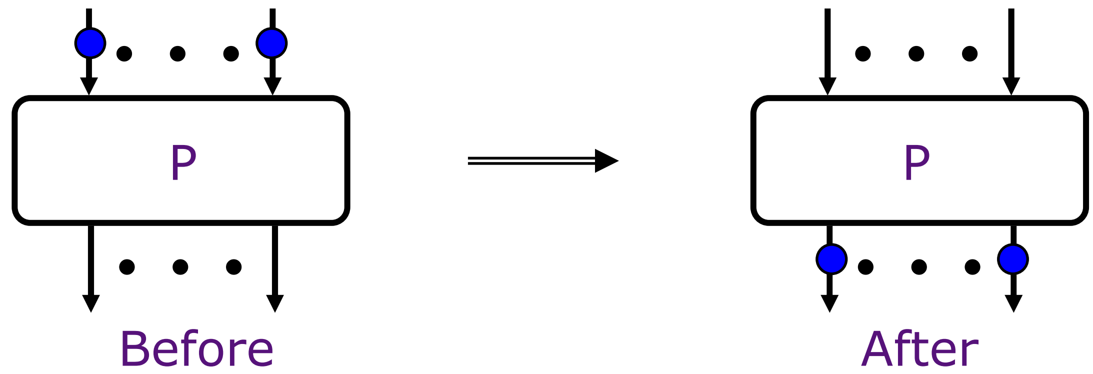
Dataflow Operators

- A small set of dataflow operators can be used to define a general programming language

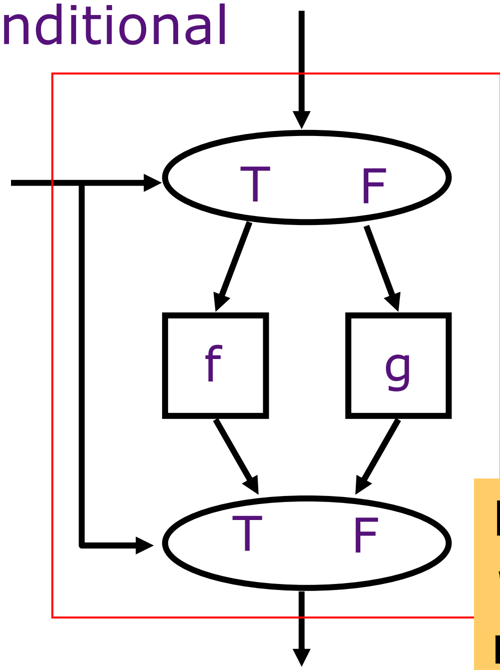


Well Behaved Schemas

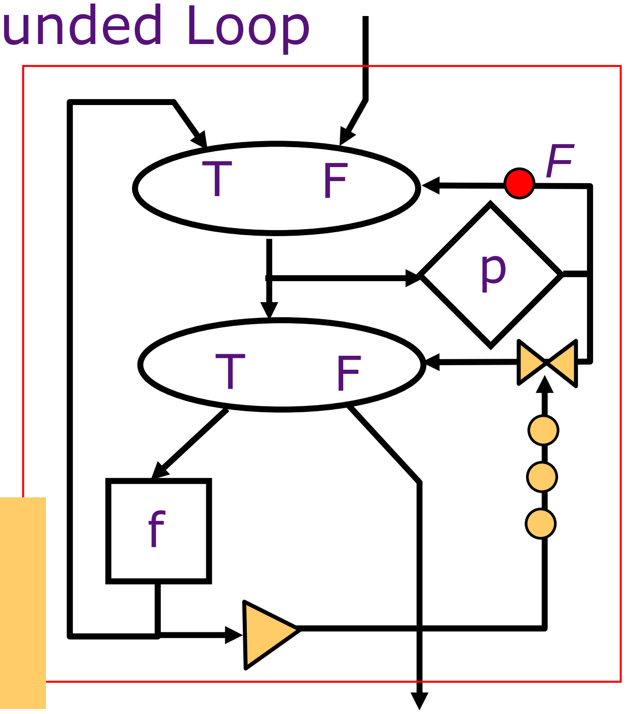
one-in-one-out
& self cleaning



Conditional



Bounded Loop



Needed for resource management

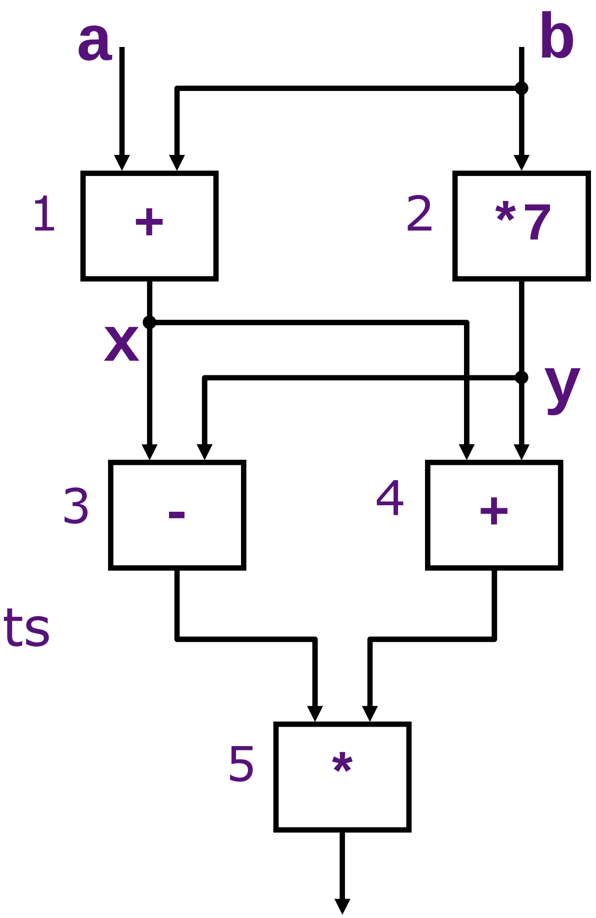
Outline

- Dataflow graphs √
  - A clean model of parallel computation
- Static Dataflow Machines ←
  - Not general-purpose enough
- Dynamic Dataflow Machines
  - As easy to build as a simple pipelined processor
- The software view
  - The memory model: I-structures
- Monsoon and its performance
- Musings

Static Dataflow Machine:
Instruction Templates

	Opcode	Destination 1	Destination 2	Operand 1	Operand 2
1	+	3L	4L		
2	*	3R	4R		
3	-	5L			
4	+	5R			
5	*	out			

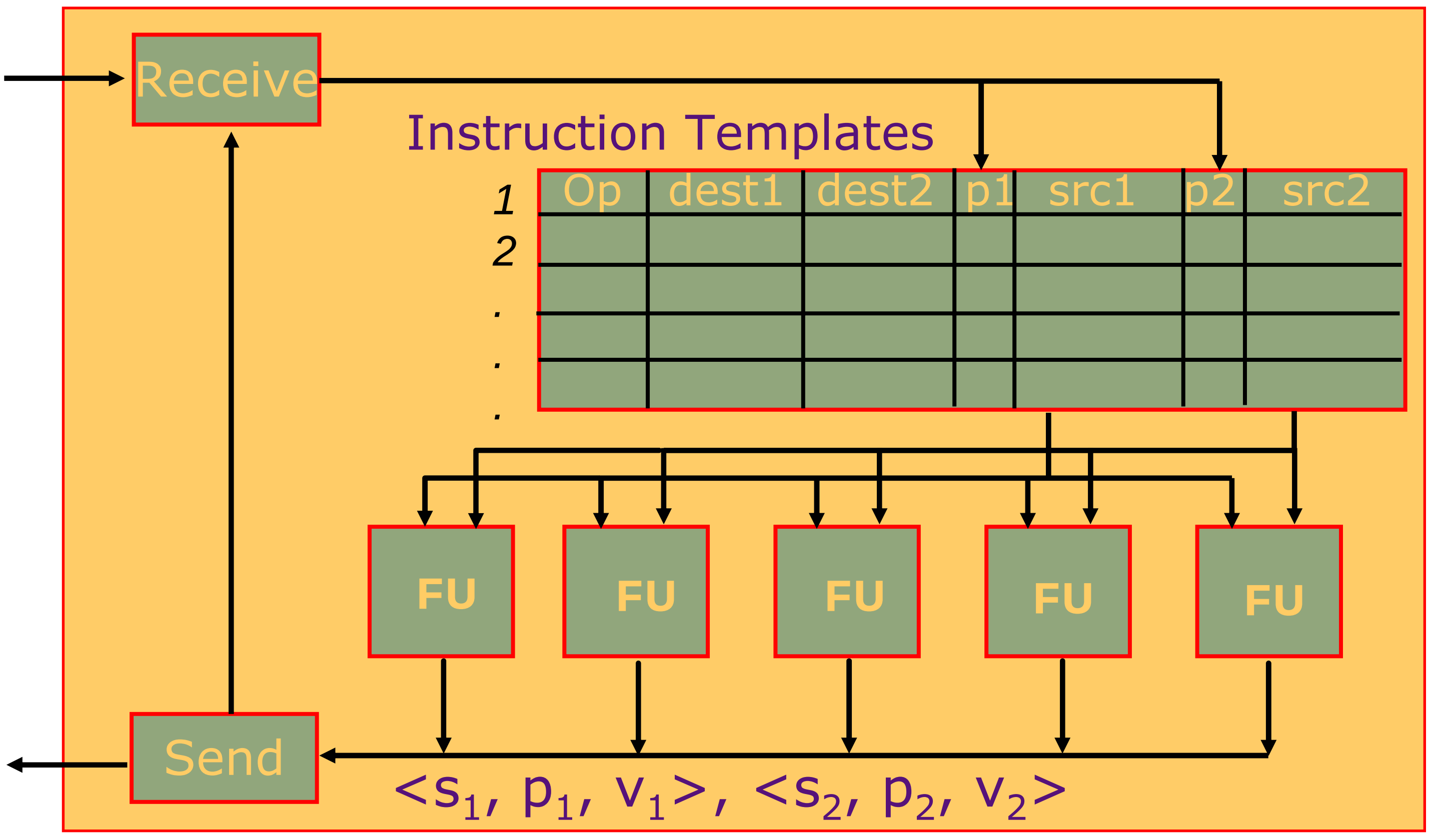
Presence bits



Each arc in the graph has a operand slot in the program

Static Dataflow Machine

Jack Dennis, 1973



- Many such processors can be connected together
- Programs can be statically divided among the processor

Static Dataflow: Problems/Limitations

- Mismatch between the model and the implementation
  - The model requires *unbounded FIFO token queues* per arc but the architecture provides storage for one token per arc
  - The architecture *does not ensure FIFO* order in the reuse of an operand slot
  - The *merge* operator has a unique firing rule

- The static model *does not support*
  - Function calls
  - Data Structures

- No easy solution in the static framework
- Dynamic dataflow provided a framework for solutions

Outline

- Dataflow graphs √
  - A clean model of parallel computation
- Static Dataflow Machines √
  - Not general-purpose enough
- Dynamic Dataflow Machines ←
  - As easy to build as a simple pipelined processor
- The software view
  - The memory model: I-structures
- Monsoon and its performance

- Musings

Dynamic Dataflow Architectures

- Allocate instruction templates, i.e., a frame, dynamically to support each loop iteration and procedure call
  - termination detection needed to deallocate frames

- The code can be shared if we separate the code and the operand storage

a token <fp, ip, port, data>

frame pointer (fp)

instruction pointer (ip)

A Frame in Dynamic Dataflow

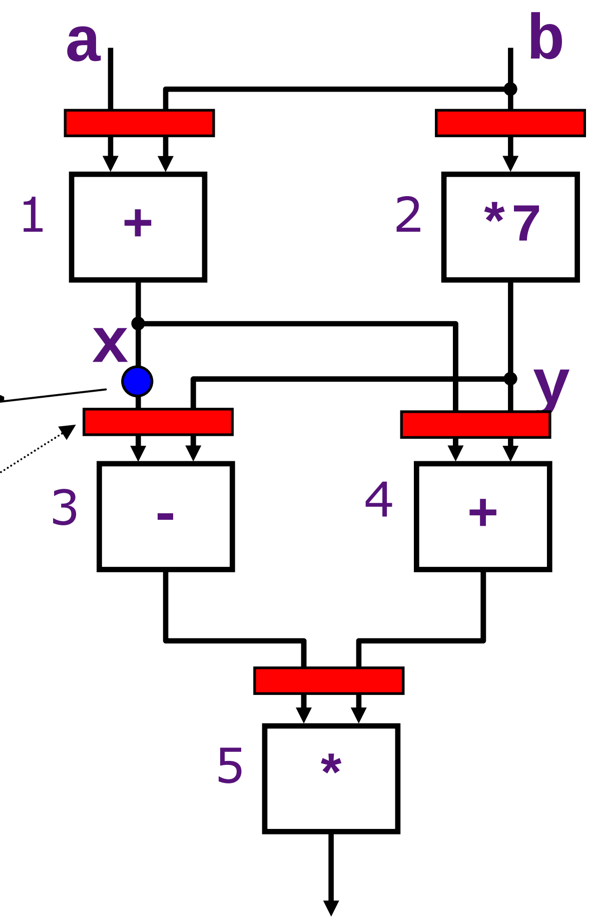
Program

1	+	*1*	3L, 4L
2	*	*2*	3R, 4R
3	-	*3*	5L
4	+	*4*	5R
5	*	*5*	out

<fp, ip, p , v>

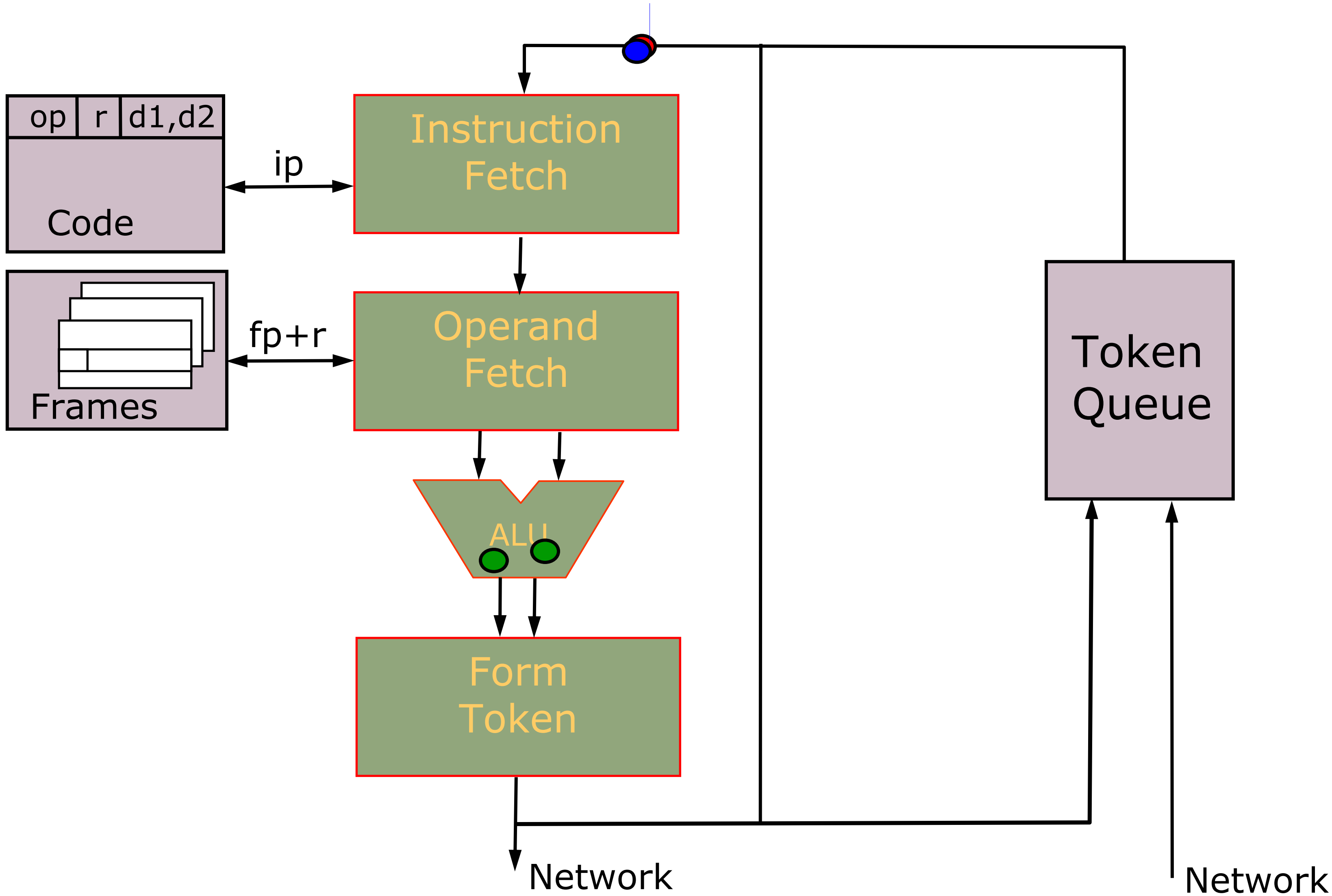
Frame

1		
2		7
3		
4		
5		



Need to provide storage for only one operand/operator

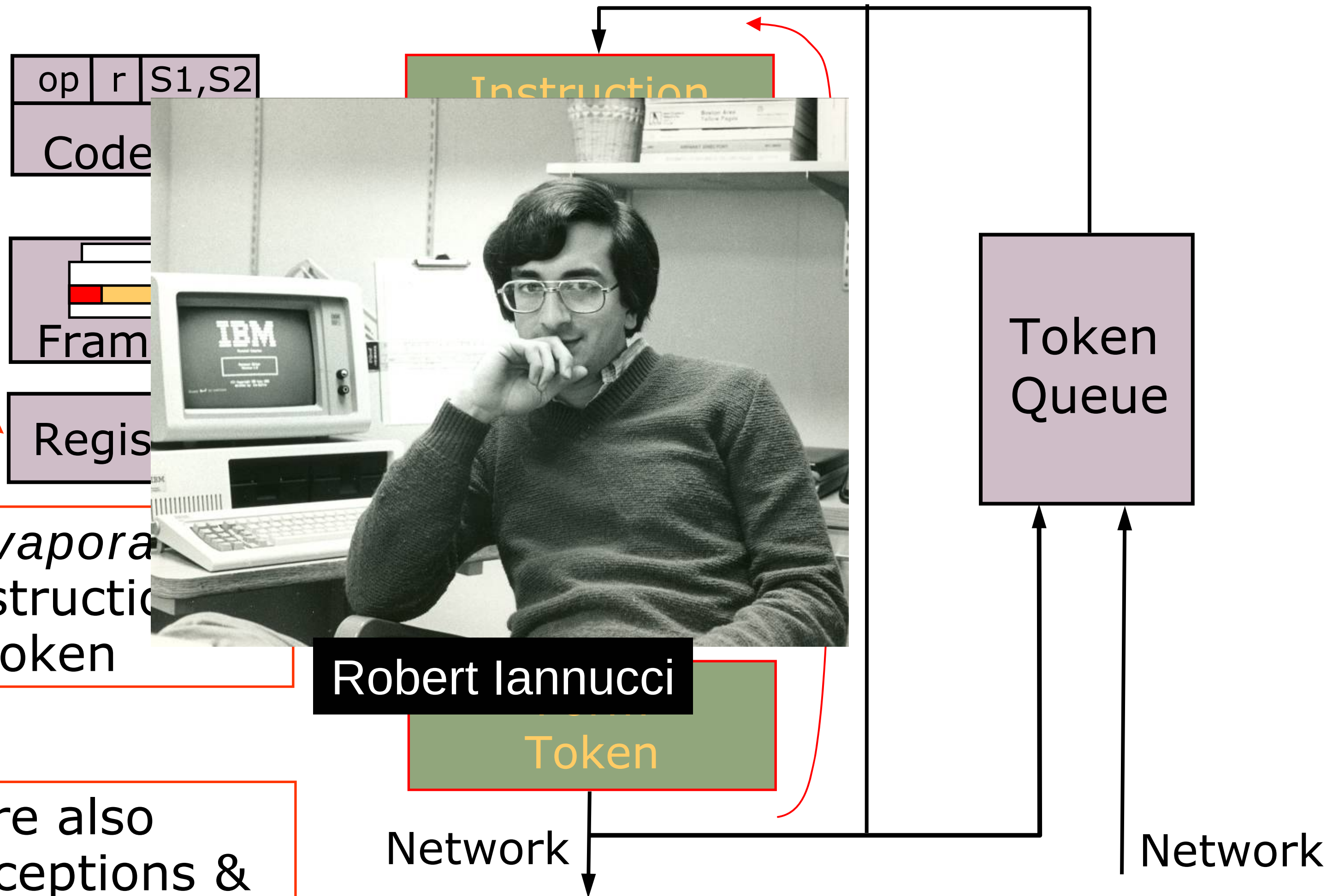
Monsoon Processor
Greg Papadopoulos



Temporary Registers & Threads

Robert Iannucci

n sets of registers (n = pipeline depth)

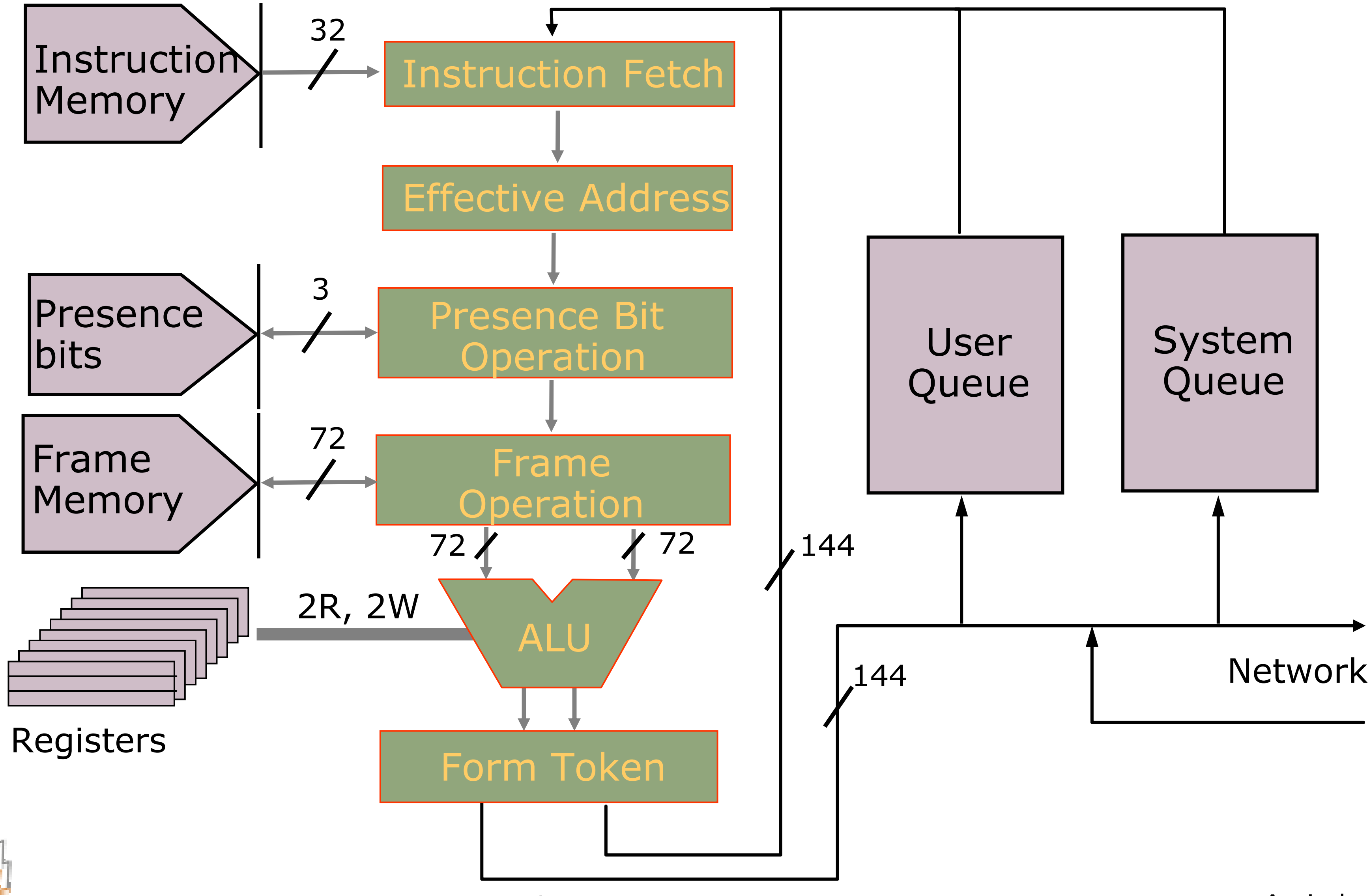


Robert Iannucci

Registers *evapora*
when an instructi
thread is broken

Registers are also used for exceptions & interrupts

Actual Monsoon Pipeline:
Eight Stages



Instructions directly control the pipeline

The opcode specifies an operation for each pipeline stage:

opcode   r   dest1   [dest2]

EA WM RegOp ALU FormToken

Easy to implement; no hazard detection

EA - effective address

FP + r: *frame relative*
r: *absolute*
IP + r: *code relative* (not supported)

WM - waiting matching

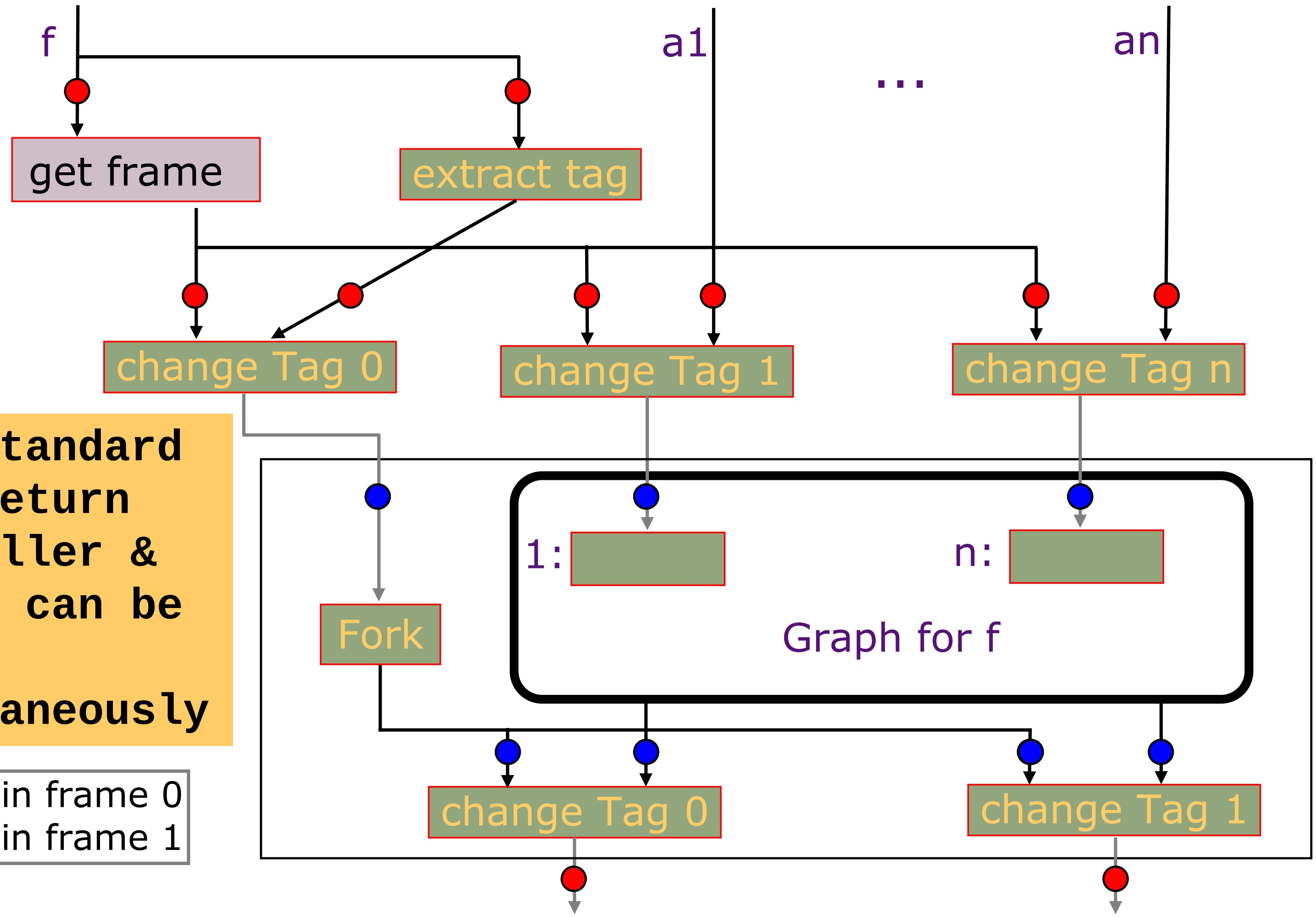
Unary; Normal; Sticky; Exchange; Imperative
PBs X port $\rightarrow$ PBs X Frame op X ALU inhibit

Register ops:

ALU: V_L X V_R $\rightarrow$ V'_L X V'_R , CC

Form token: V_L X V_R X Tag_1 X Tag_2 X CC $\rightarrow$ $Token_1$ X $Token_2$

Procedure Linkage Operators



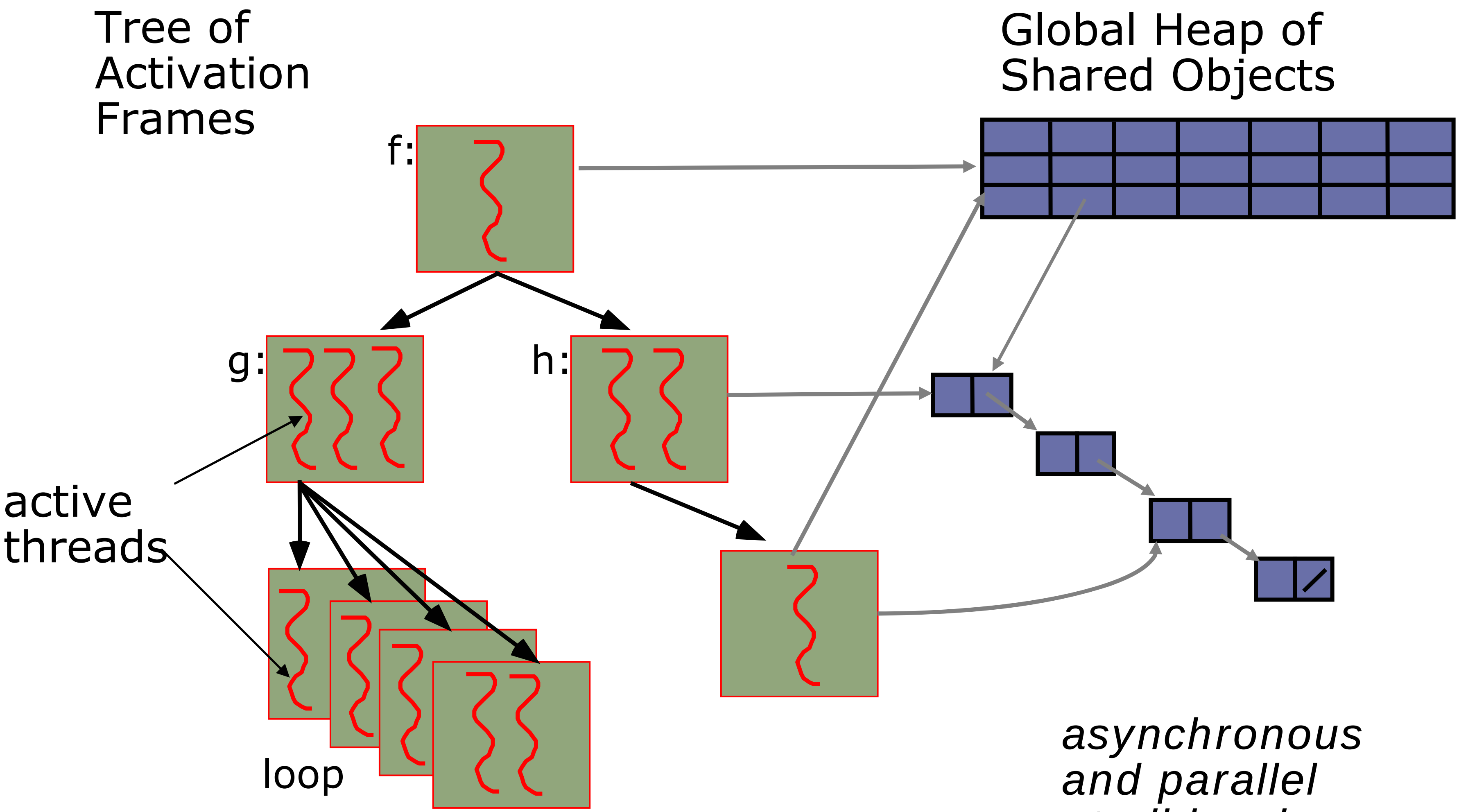
Like standard call/return but caller & callee can be active simultaneously

- token in frame 0
- token in frame 1

Outline

- Dataflow graphs √
  - A clean model of parallel computation
- Static Dataflow Machines √
  - Not general-purpose enough
- Dynamic Dataflow Machines √
  - As easy to build as a simple pipelined processor
- The software view ←
  - The memory model: I-structures
- Monsoon and its performance
- Musings

Parallel Language Model



asynchronous
and parallel
at all levels

Id World
implicit parallelism

Id

Dataflow Graphs + I-Structures + . . .

TTDA

Monsoon

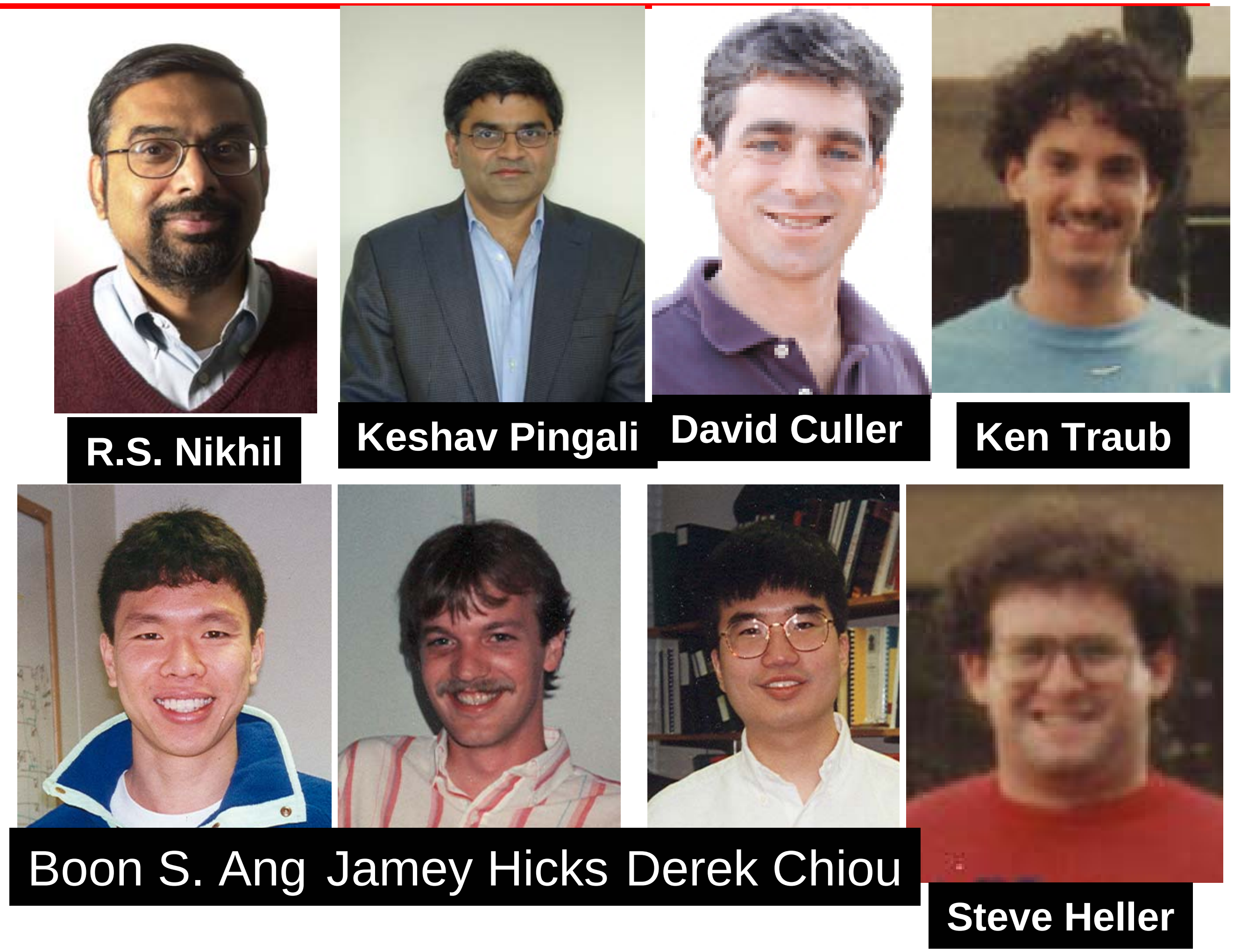
*T

*T-Voyager

Id World people

- Rishiyur Nikhil,
- Keshav Pingali,
- Vinod Kathail,
- David Culler
- Ken Traub
- Steve Heller,
- Richard Soley,
- Dinart Mores
- Jamey Hicks,
- Alex Caro,
- Andy Shaw,
- Boon Ang
- Shail Anditya
- R Paul Johnson
- Paul Barth
- Jan Maessen
- Christine Flood
- Jonathan Young
- Derek Chiou
- Arun Iyangar
- Zena Ariola
- Mike Bekerle

- K. Eknadham (IBM)
- Wim Bohm (Colorado)
- Joe Stoy (Oxford)
- ...

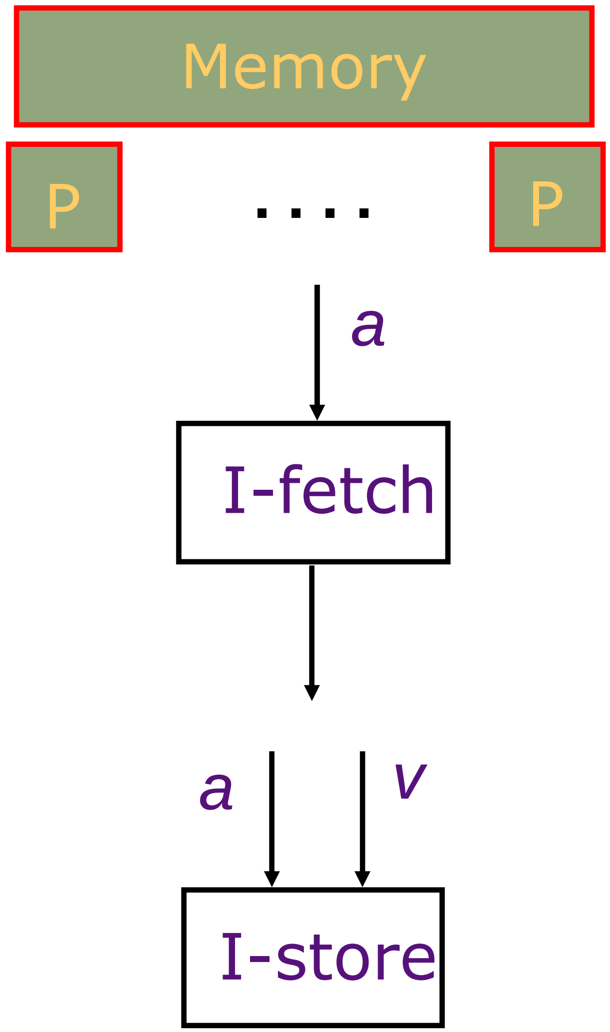


Data Structures in Dataflow

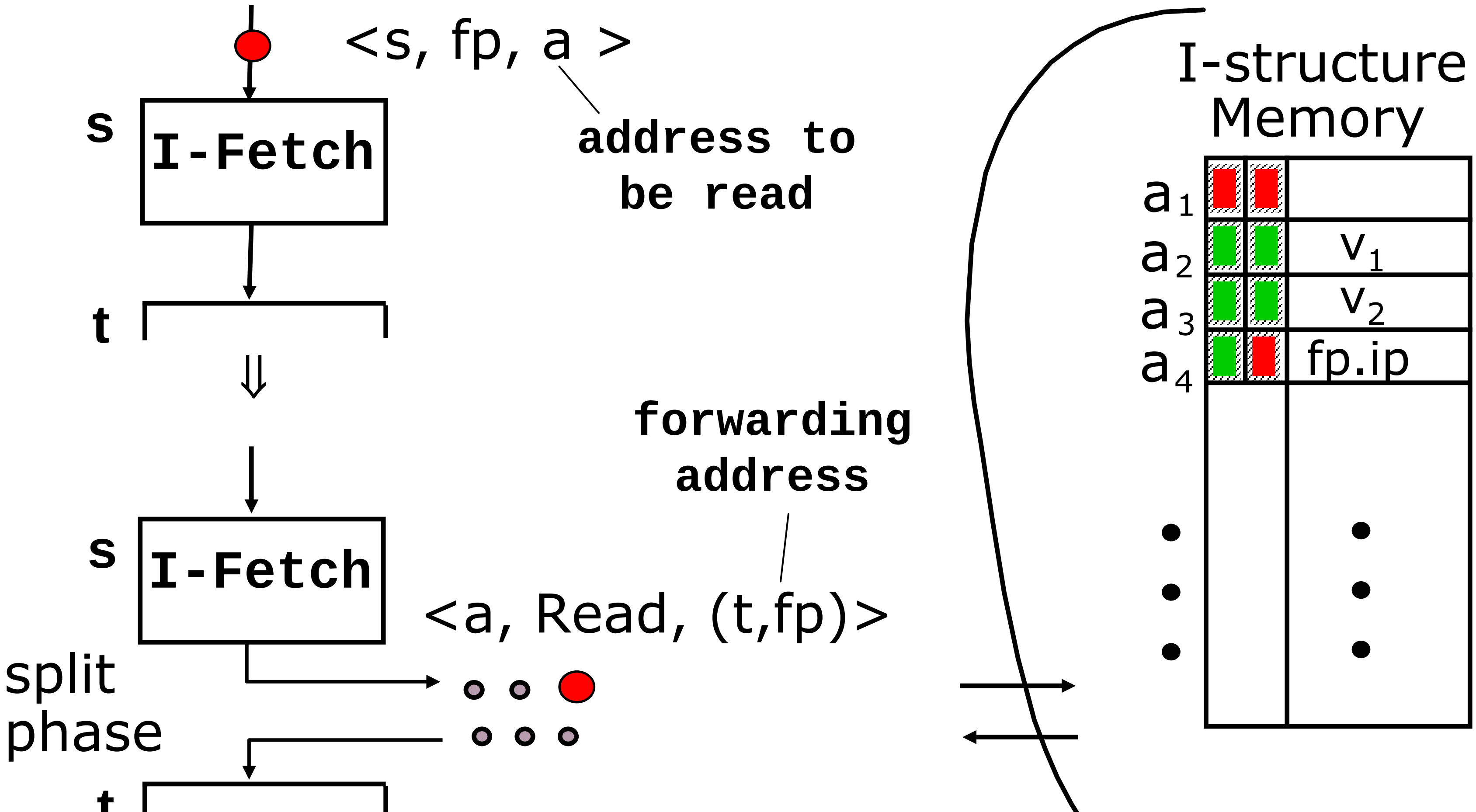
- Data structures reside in a structure store
  - ⇒ tokens carry pointers

- I-structures: Write-once, Read multiple times *or*
  - allocate, write, read, ..., read, deallocate

  ⇒ No problem if a reader arrives before the writer at the memory location



I-Structure Storage: Split-phase operations & Presence bits



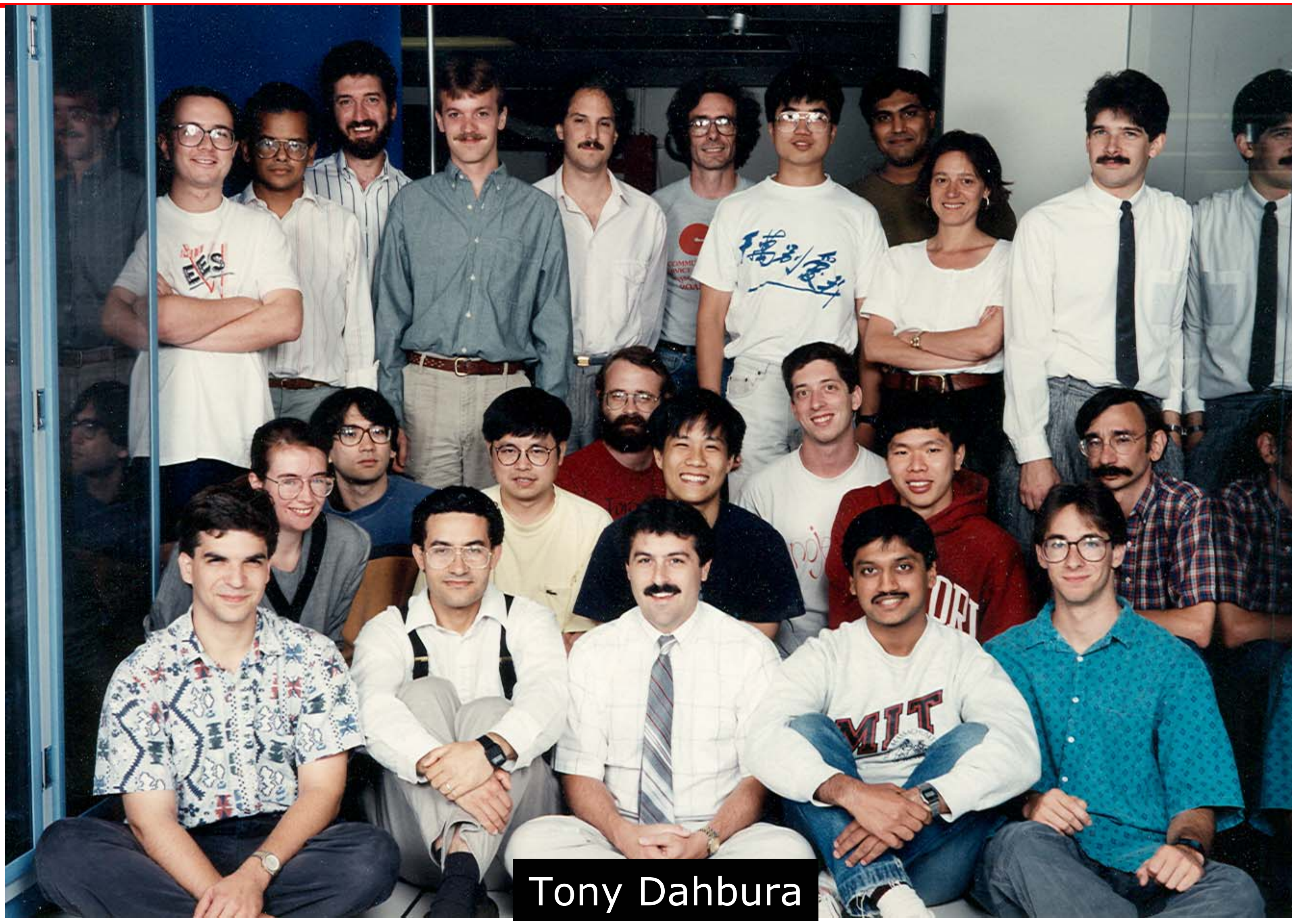
- *Need to deal with multiple deferred reads*
- *other operations:*

  fetch/store, take/put, clear

Outline

- Dataflow graphs √
  - A clean parallel model of computation
- Static Dataflow Machines √
  - Not general-purpose enough
- Dynamic Dataflow Machines √
  - As easy to build as a simple pipelined processor
- The software view √
  - The memory model: I-structures
- Monsoon and its performance ←
- Musings

The Monsoon Project
Motorola Cambridge Research Center + MIT



rds

Id Applications on Monsoon @ MIT

- Numerical
  - Hydrodynamics - SIMPLE
  - Global Circulation Model - GCM
  - Photon-Neutron Transport code -GAMTEB
  - N-body problem
- Symbolic
  - Combinatorics - free tree matching,Paraffins
  - Id-in-Id compiler
- System
  - I/O Library
  - Heap Storage Allocator on Monsoon
- Fun and Games
  - Breakout
  - Life
  - Spreadsheet

Id Run Time System (RTS) on Monsoon

- *Frame Manager:* Allocates frame memory on processors for procedure and loop activations

  Derek Chiou

- *Heap Manager:* Allocates storage in I-Structure memory or in Processor memory for heap objects.

  Arun Iyengar

Single Processor Monsoon Performance Evolution

One 64-bit processor (10 MHz) + 4M 64-bit I-structure

	Feb. 91	*Aug. 91*	*Mar. 92*	*Sep. 92*
Matrix Multiply 500x500	4:04	3:58	3:55	1:46
Wavefront 500x500, 144 iters.	5:00	5:00	3:48	
Paraffins				
n = 19	:50	:31		:02.4
n = 22				:32
GAMTEB-9C				
40K particles	17:20	10:42	5:36	5:36
1M particles	7:13:20	4:17:14	2:36:00	2:22:00
SIMPLE-100				
1 iterations	:19	:15	:10	:06
1K iterations	4:48:00			1:19:49

Need a real machine to do this

hours:minutes:seconds

Monsoon Speed Up Results

Boon Ang, Derek Chiou, Jamey Hicks

	speed up				critical path (millions of cycles)			
	1pe	2pe	4pe	8pe	1pe	2pe	4pe	8pe
Matrix Multiply 500 x 500	1.00	1.99	3.90	7.74	1057	531	271	137
Paraffins n=22	1.00	1.99	3.92	7.25	322	162	82	44
GAMTEB-2C 40 K particles	1.00	1.95	3.81	7.35	590	303	155	80
SIMPLE-100 100 iters	1.00	1.86	3.45	6.27	4681	2518	1355	747

September, 1992

Could not have asked for more

Base Performance?
Id on Monsoon vs. C / F77 on R3000

	MIPS (R3000) (x 10e6 cycles)	Monsoon (1pe) (x 10e6 cycles)
Matrix Multiply 500 x 500	954 +	1058
Paraffins n=22	102 +	322
GAMTEB-9C 40 K particles	265 *	590
SIMPLE-100 100 iters	1787 *	4682

MIPS codes won't run on a parallel machine without recompilation/recoding

8-way superscalar? Unlikely to give 7 fold speedup

R3000 cycles collected via Pixie
* Fortran 77, fully optimized + MIPS C, O = 3
64-bit floating point used in Matrix-Multiply, GAMTEB and SIMPLE

The Monsoon Experience

- Performance of implicitly parallel Id programs scaled effortlessly.

- Id programs on a single-processor Monsoon took 2 to 3 times as many cycles as Fortran/C on a modern workstation.
  - Can certainly be improved

- Effort to develop the *invisible software* (loaders, simulators, I/O libraries,....) *dominated* the effort to develop the *visible software* (compilers...)

Outline

- Dataflow graphs √
  - A clean parallel model of computation
- Static Dataflow Machines √
  - Not general-purpose enough
- Dynamic Dataflow Machines √
  - As easy to build as a simple pipelined processor
- The software view √
  - The memory model: I-structures
- Monsoon and its performance √

- Musings ←

What would we have done differently - 1

- Technically: Very little
  - Simple, high performance design, easily exploits fine-grain parallelism, tolerates latencies efficiently
  - Id preserves fine-grain parallelism which is abundant
  - Robust compilation schemes; DFGs provide easy compilation target
- Of course, there is room for improvement
  - Functionally several different types of memories (frames, queues, heap); all are not full at the same time
  - Software has no direct control over large parts of the memory, e.g., token queue
  - Poor single-thread performance and it hurts when single thread latency is on a critical path.

What would we have done differently - 2

- Non technical but perhaps even more important
  - It is difficult enough to cause one revolution but two? Wake up?
  - Cannot ignore market forces for too long – may affect acceptance even by the research community
  - Should the machine have been built a few years earlier (in lieu of simulation and compiler work)?
    Perhaps it would have had more impact
    (had it worked)
  - The follow on project should have been about:
    1. Running conventional software on DF machines, or
    2. About making minimum modifications to commercial microprocessors

    (We chose 2 but perhaps 1 would have been better)

Imperative Programs and Multi-Cores

- Deep pointer analysis is required to extract parallelism from sequential codes
  - otherwise, extreme speculation is the only solution
- A multithreaded/dataflow model is needed to present the found parallelism to the underlying hardware
- Exploiting fine-grain parallelism is necessary for many situations, e.g., producer-consumer parallelism

Locality and Parallelism: Dual problems?

- Good performance requires exploiting both
- Dataflow model gives you parallelism for free, but requires analysis to get locality
- C (mostly) provides locality for free but one must do analysis to get parallelism
  - Tough problems are tough independent of representation

Parting thoughts

- Dataflow research as conceived by most researchers achieved its goals
  - The model of computation is beautiful and will be resurrected whenever people want to exploit fine-grain parallelism

- But installed software base has a different model of computation which provides different challenges for parallel computing
  - Maybe possible to implement this model effectively on dataflow machines – we did not investigate this but is **absolutely worth investigating further**
  - Current efforts on more standard hardware are having lots of their own problems
  - Still an open question on what will work in the end

Thank You!

and thanks to

R.S.Nikhil, Dan Rosenband,
James Hoe, Derek Chiou,
Larry Rudolph, Martin Rinard,
Keshav Pingali

for helping with this talk

DFGs vs CDFGs

- Both Dataflow Graphs and Control DFGs had the goal of structured, well-formed, compositional, executable graphs
- CDFG research (70s, 80s) approached this goal starting with original sequential control-flow graphs (“flowcharts”) and data-dependency arcs, and gradually adding structure (e.g., φ-functions)
- Dataflow graphs approached this goal directly, *by construction*
  - Schemata for basic blocks, conditionals, loops, procedures
- CDFGs is an Intermediate representation for compilers and, unlike DFGs, not a language.