Dynamic Dataflow

Arvind
Computer Science & Artificial Intelligence Lab
Massachusetts Institute of Technology

D

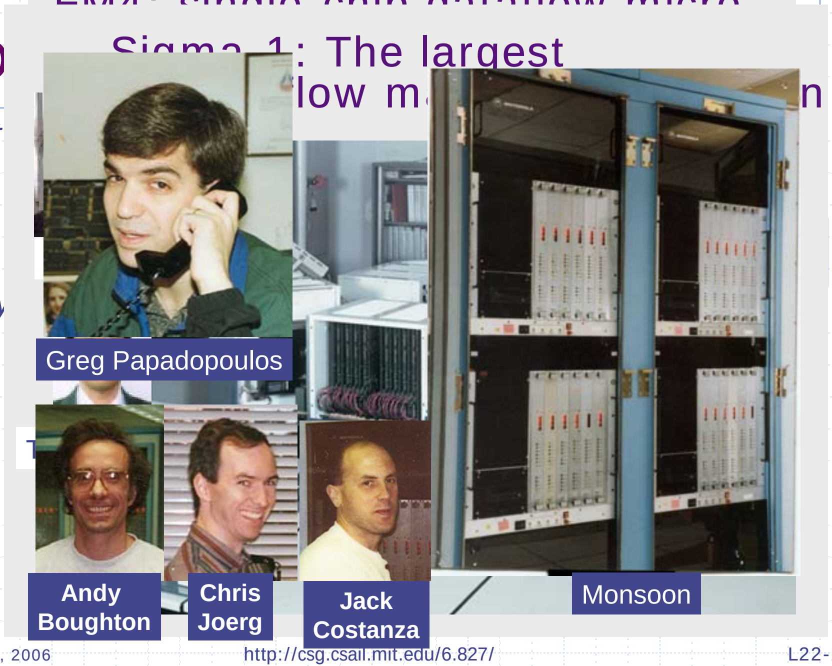
St

Dy

T

EM4: single chip dataflow micro

Sigma-1: The largest
low m n



Outline

- 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

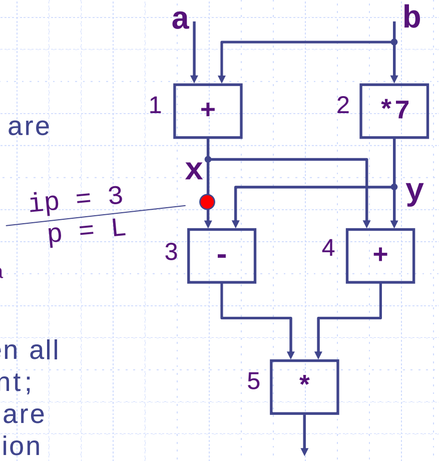
Dataflow Graphs

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

- Values in dataflow graphs are represented as tokens

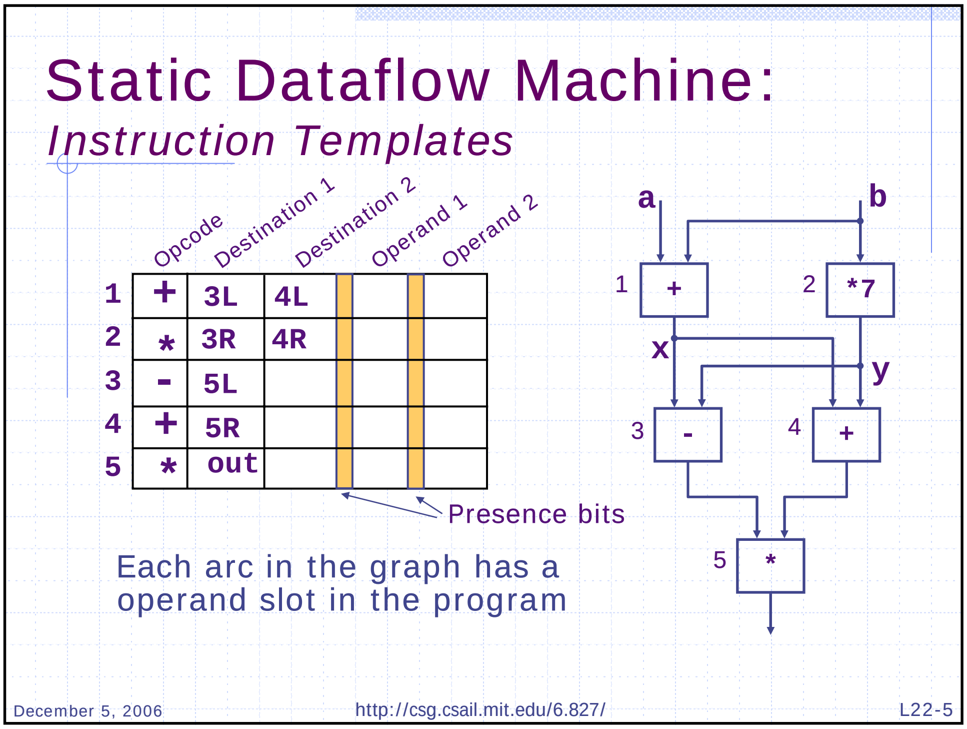
token < ip , p , v >

instruction ptr port data



- 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



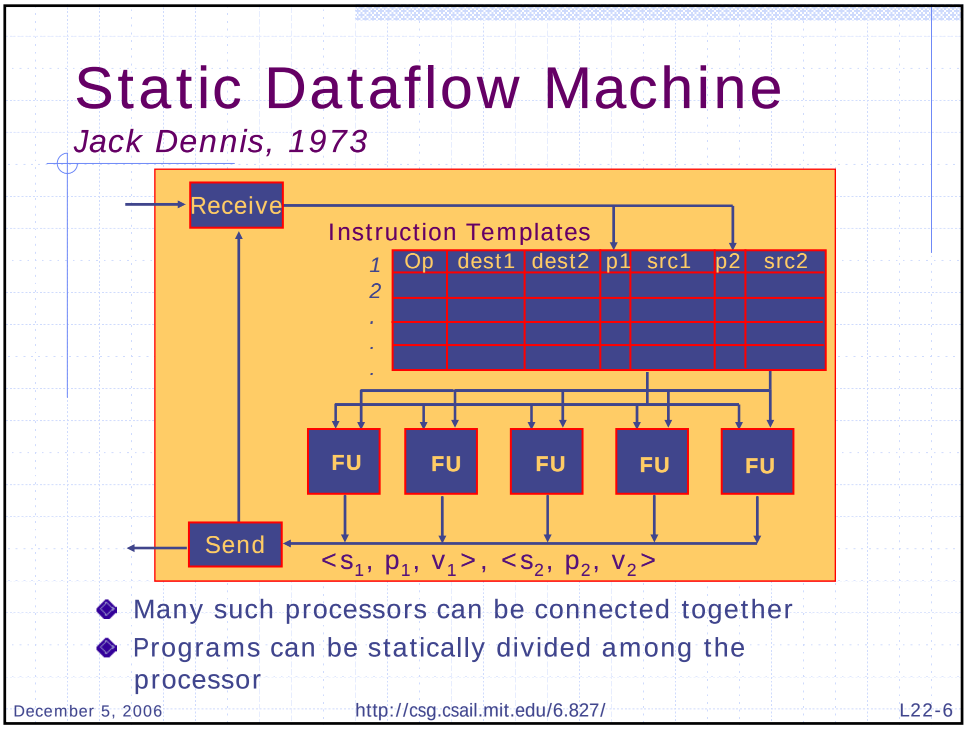
Static Dataflow Machine:

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



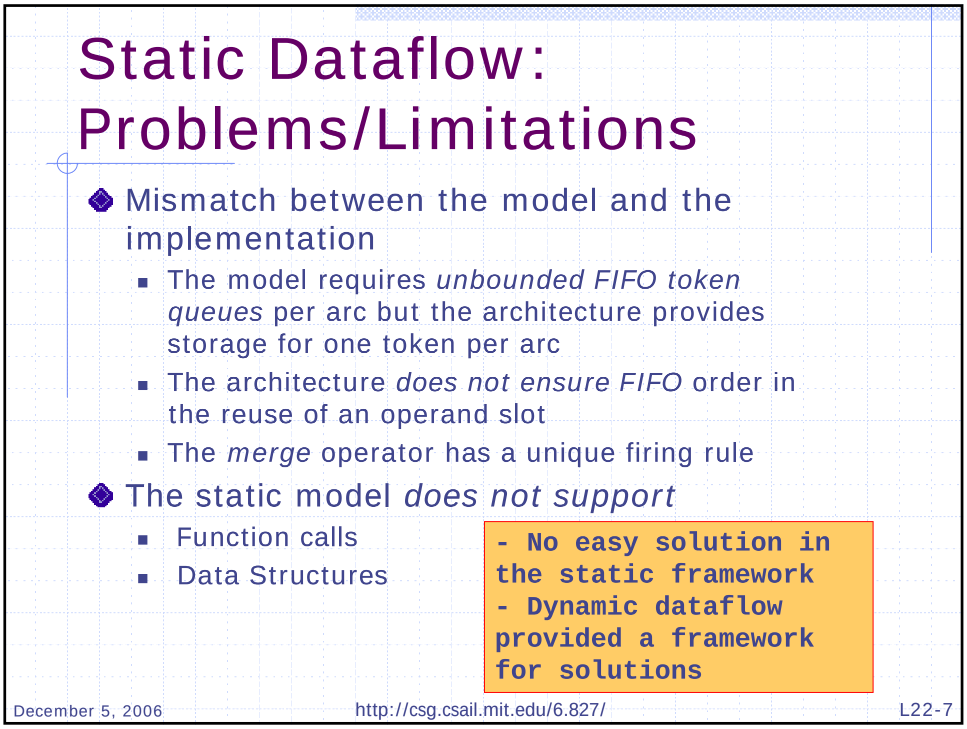
Static Dataflow Machine

Jack Dennis, 1973

	Op	dest1	dest2	p1	src1	p2	src2
1							
2							
.							
.							
.							

$<s_1, p_1, v_1>, <s_2, p_2, v_2>$

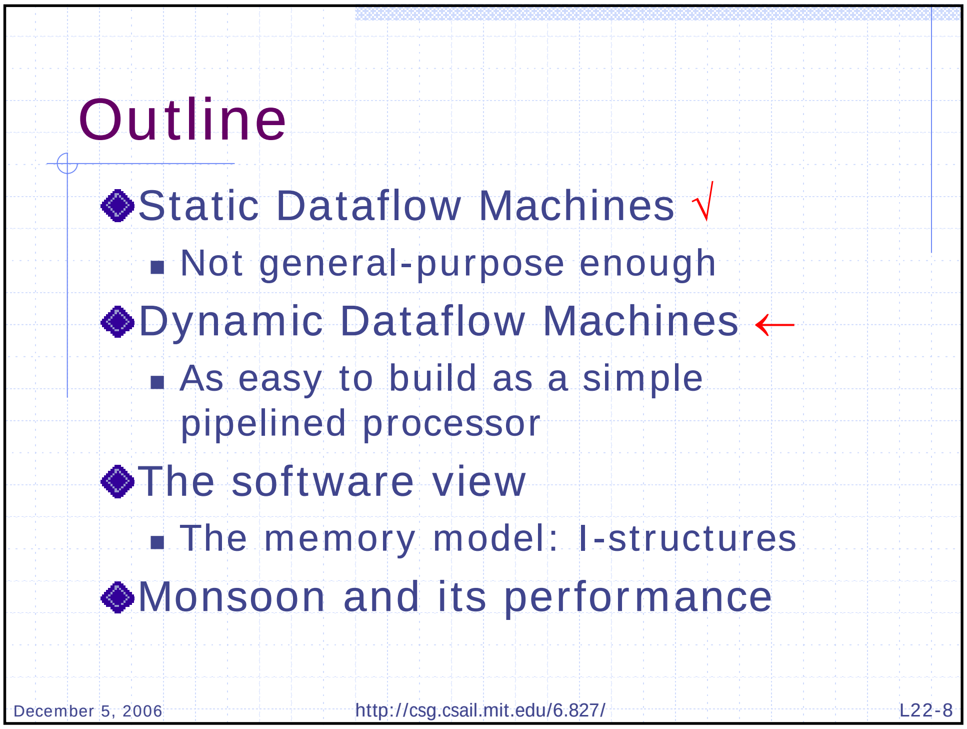
- Many such processors can be connected together
- Programs can be statically divided among the processor



Static Dataflow: Problems/Limitations

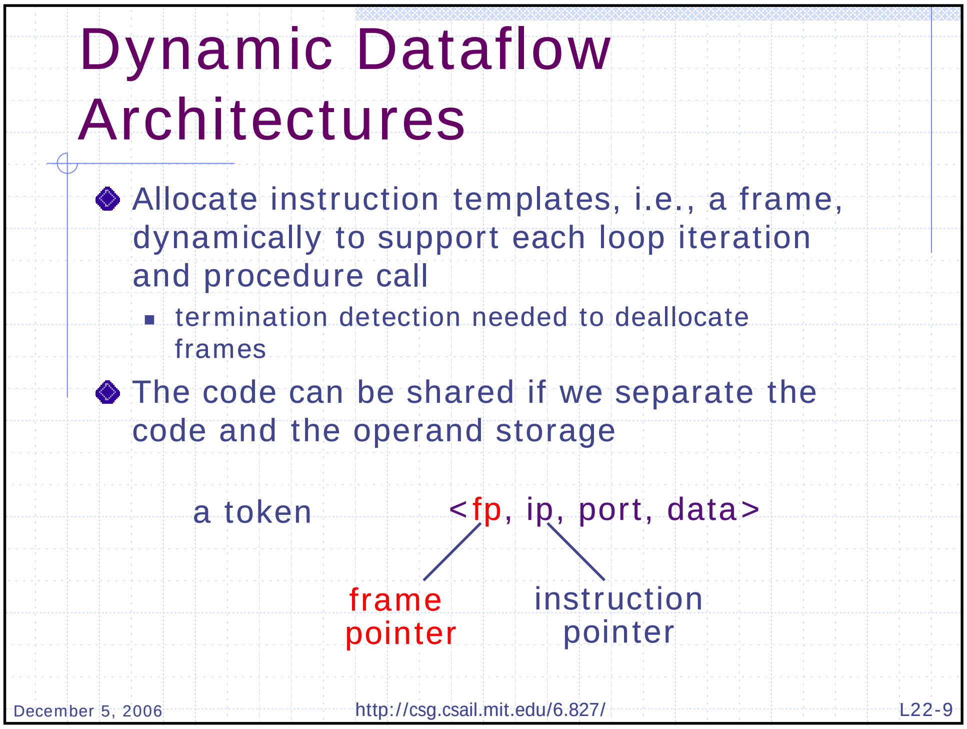
- Mismatch between the model and the implementation
  - The model requires *unbounded FIFO token queues* per arc but the architecture provides storage for one token per arc
  - The architecture *does not ensure FIFO* order in the reuse of an operand slot
  - The *merge* operator has a unique firing rule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

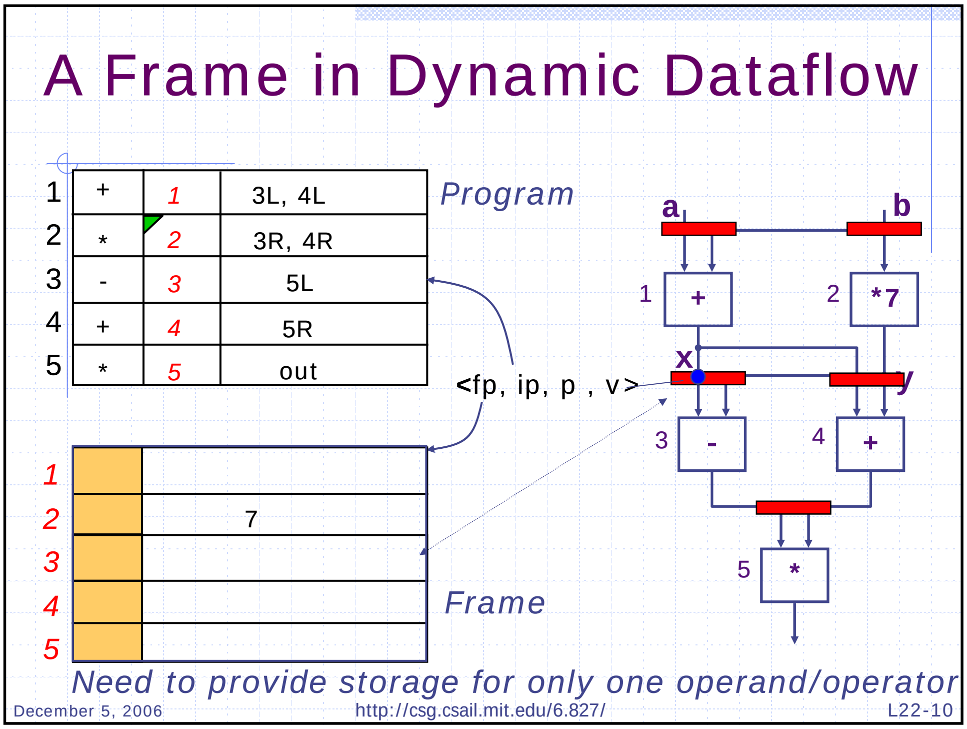
- Static Dataflow Machines √
  - Not general-purpose enough
- Dynamic Dataflow Machines ←
  - As easy to build as a simple pipelined processor
- The software view
  - The memory model: I-structures
- Monsoon and its performance



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
instruction pointer



A Frame in Dynamic Dataflow

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

Program

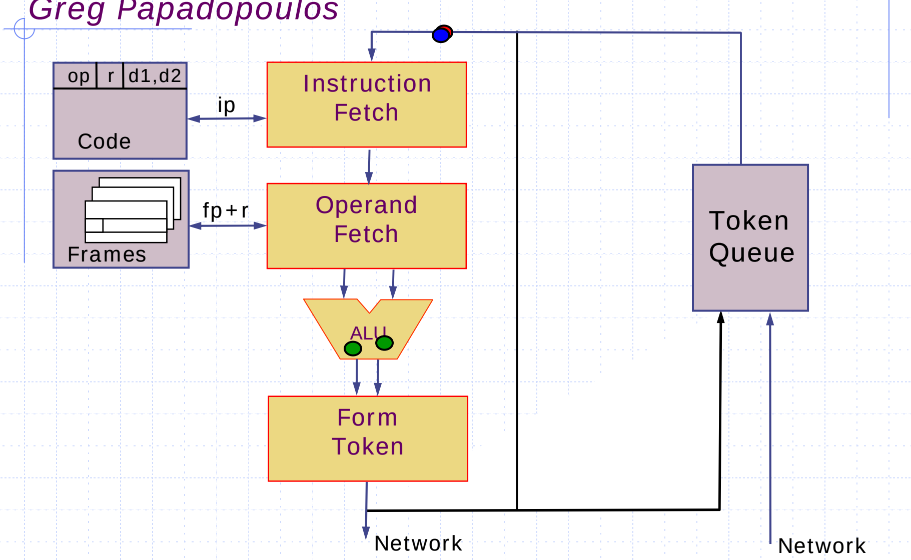
1	🟧	
2	🟧	7
3	🟧	
4	🟧	
5	🟧	

Frame

Need to provide storage for only one operand/operator

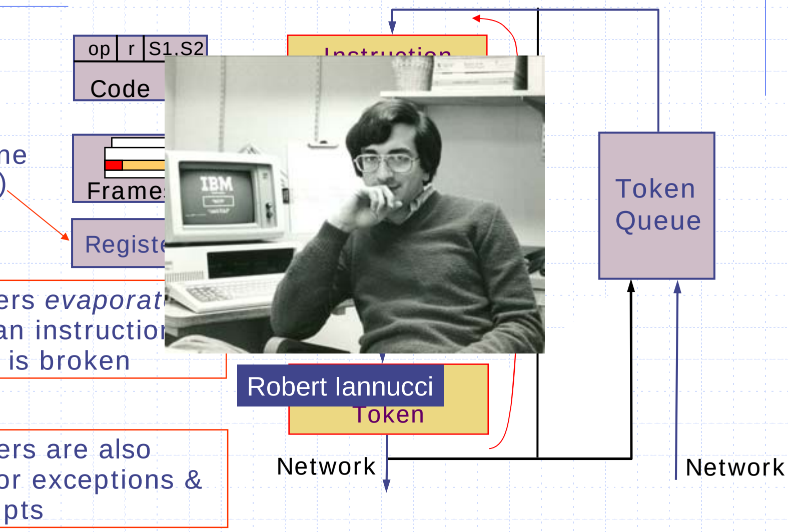
Monsoon Processor

Greg Papadopoulos



Temporary Registers & Threads *Robert Iannucci*

n sets of registers (n = pipeline depth)

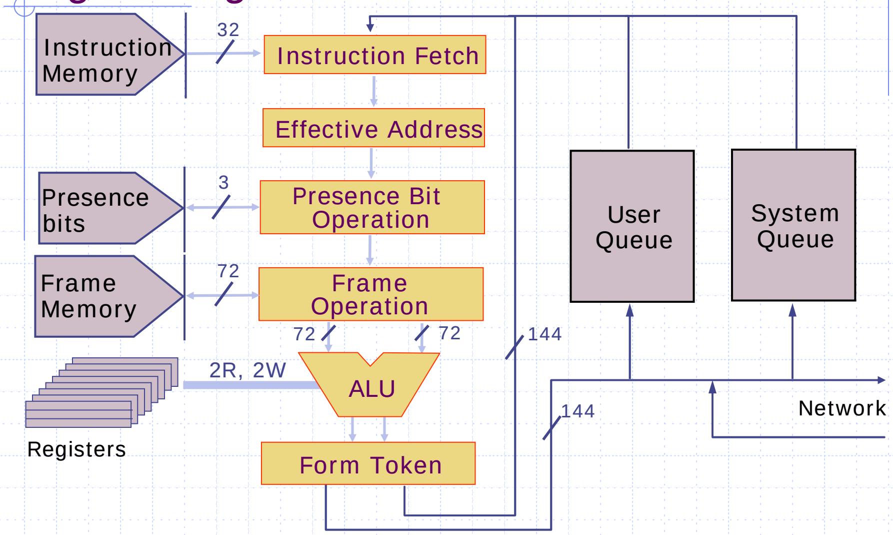


Registers *evaporate* when an instruction thread is broken

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)

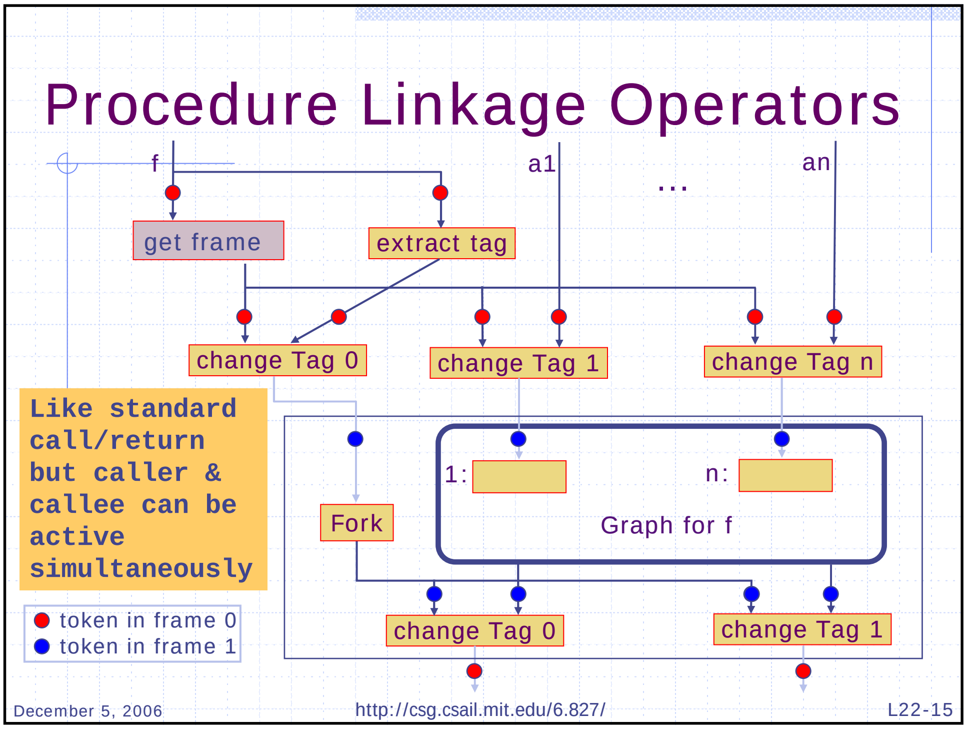
WM - waiting matching

Unary; Normal; Sticky; Exchange; Imperative
PBs X port → PBs X Frame op X ALU inhibit

Register ops:

ALU: V_L X V_R → V'_L X V'_R , CC

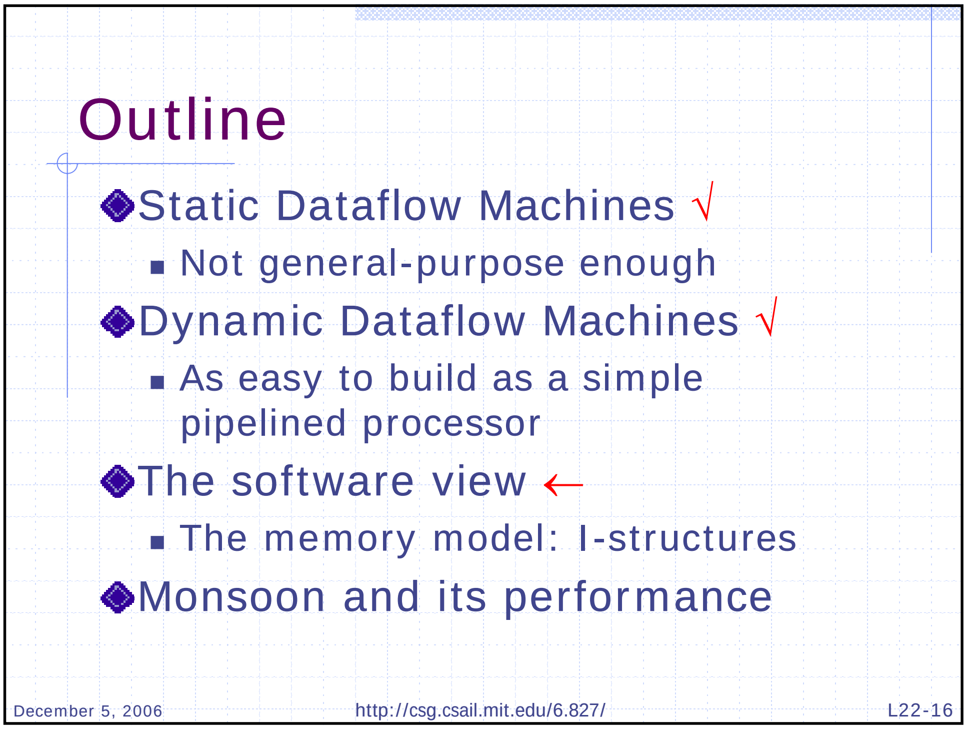
Form token: V_L X V_R X Tag_1 X Tag_2 X CC → $Token_1$ X $Token_2$



Procedure Linkage Operators

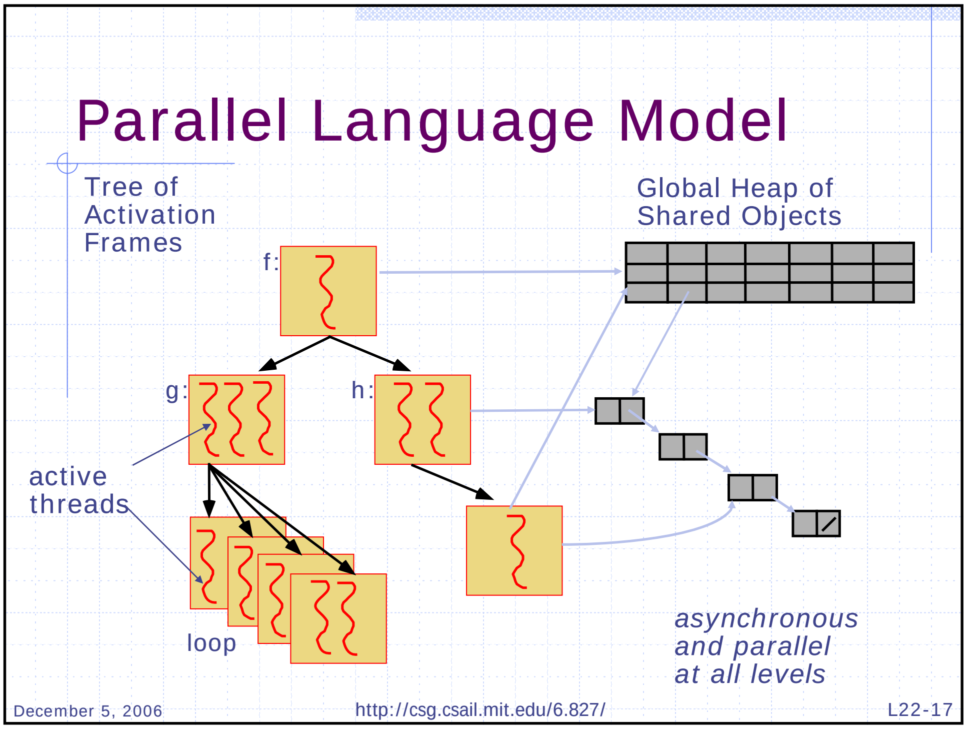
Like standard call/return but caller & callee can be active simultaneously

● token in frame 0
● token in frame 1



Outline

- 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



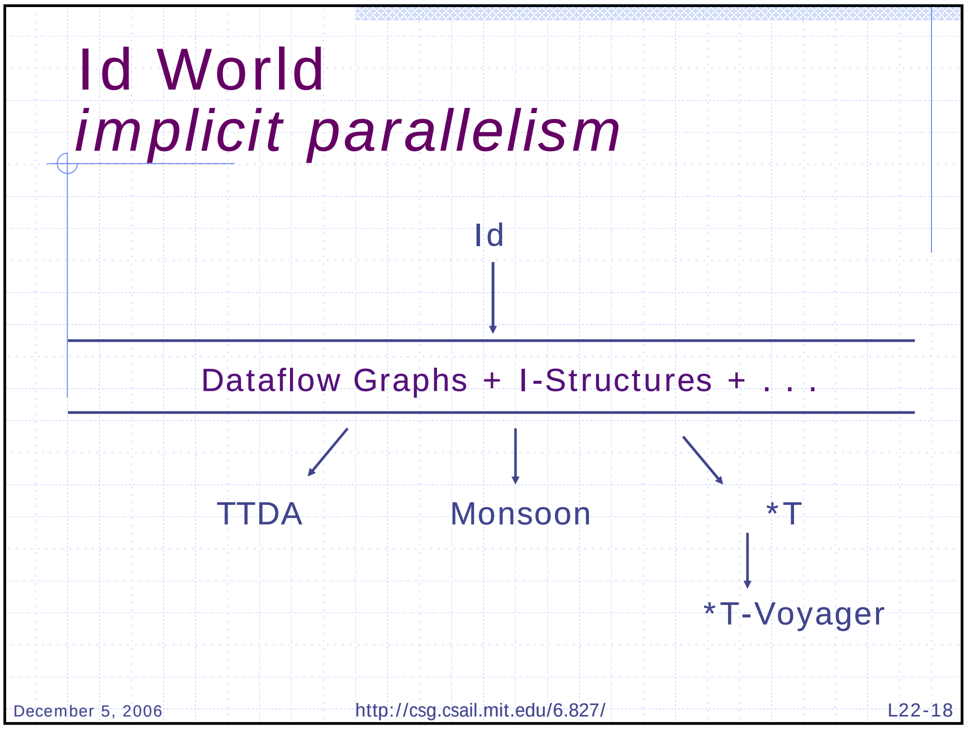
Parallel Language Model

Tree of Activation Frames

Global Heap of Shared Objects

active threads

asynchronous and parallel at all levels



Id World
implicit parallelism

Id

Dataflow Graphs + I-Structures + . . .

TTDA

Monsoon

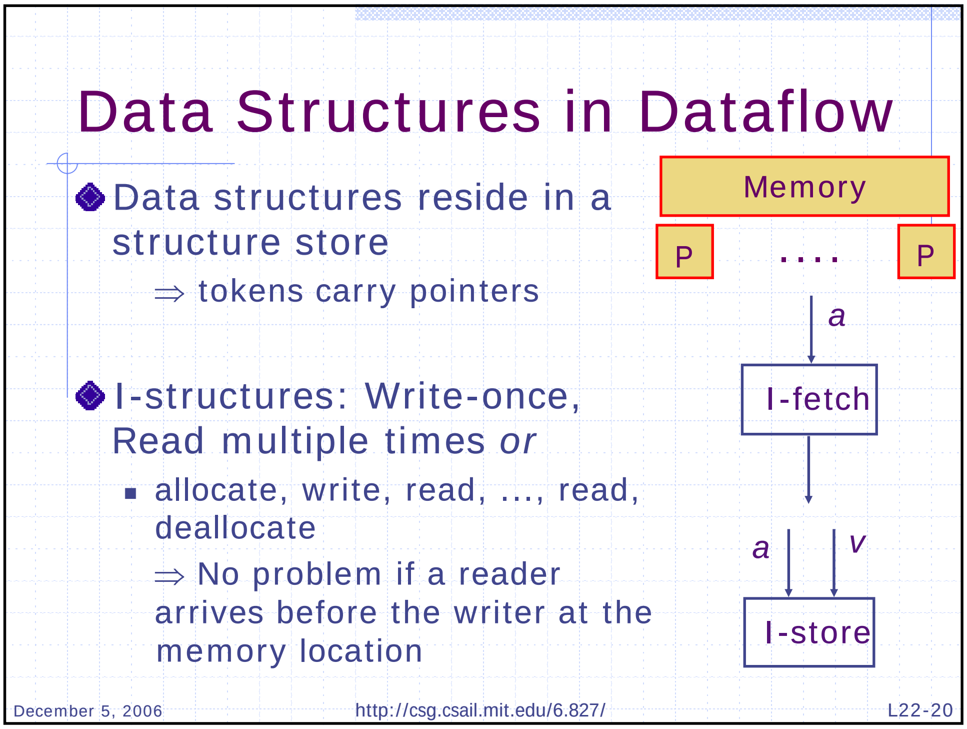
*T

*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),...



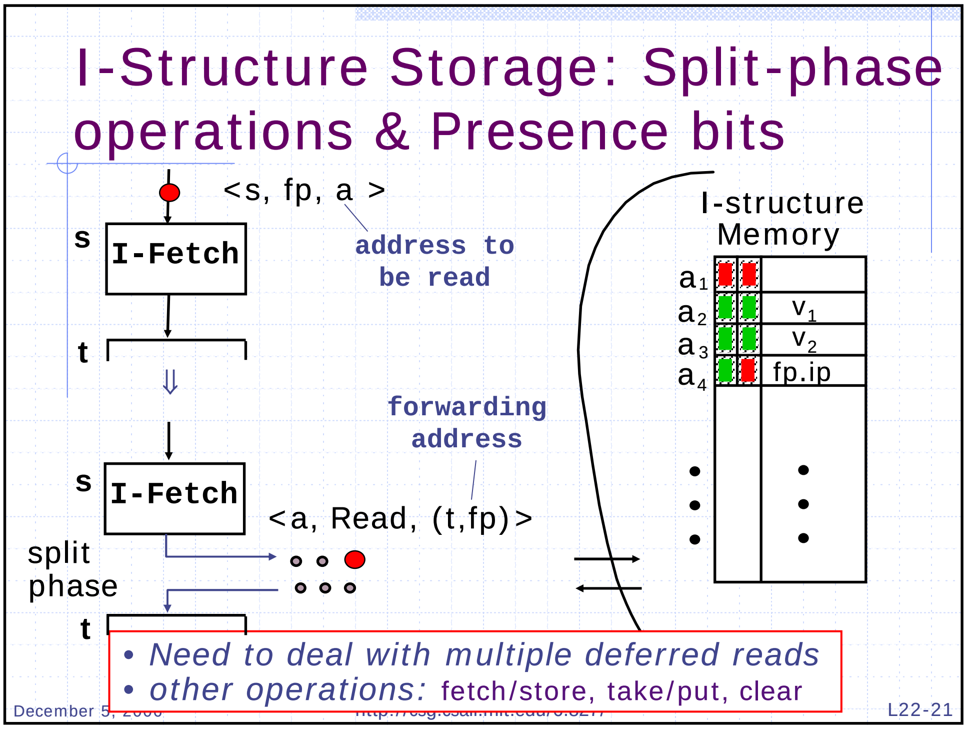
Data Structures in Dataflow

- Data structures reside in a structure store

  ⇒ tokens carry pointers

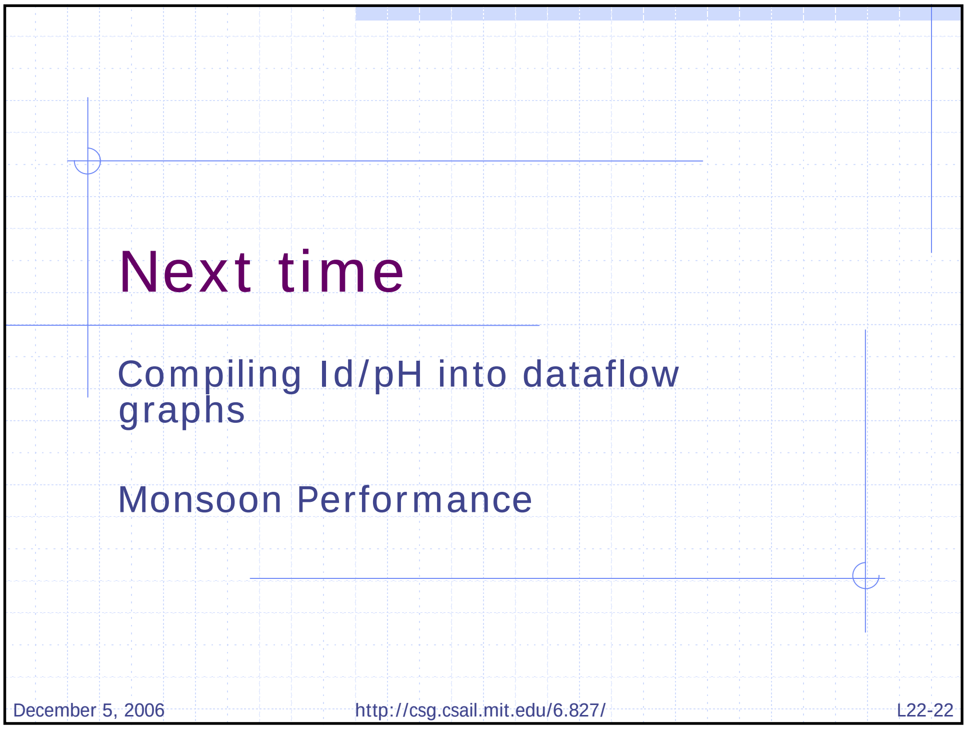
- I-structures: Write-once, Read multiple times *or*
  - allocate, write, read, ..., read, deallocate

    ⇒ 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



Next time

Compiling Id/pH into dataflow graphs

Monsoon Performance