Topic A
Dataflow Model of Computation

Guang R. Gao

ACM Fellow and IEEE Fellow
Endowed Distinguished Professor
Electrical & Computer Engineering
University of Delaware
ggao@capsl.udel.edu

Outline

- Parallel Program Execution Models
- Dataflow Models of Computation
- Dataflow Graphs and Properties
- Three Dataflow Models
  - Static
  - Recursive Program Graph
  - Dynamic
- Dataflow Architectures

Terminology Clarification

- Parallel Model of Computation
  - Parallel Models for Algorithm Designers
  - Parallel Models for System Designers
    - Parallel Programming Models
    - Parallel Execution Models
    - Parallel Architecture Models

What is a Program Execution Model?

User Code

- Application Code
- Software Packages
- Program Libraries
- Compilers
- Utility Applications

PXM (API)

System

- Hardware
- Runtime Code
- Operating System

Features a User Program Depends On

Features expressed within a Programming language

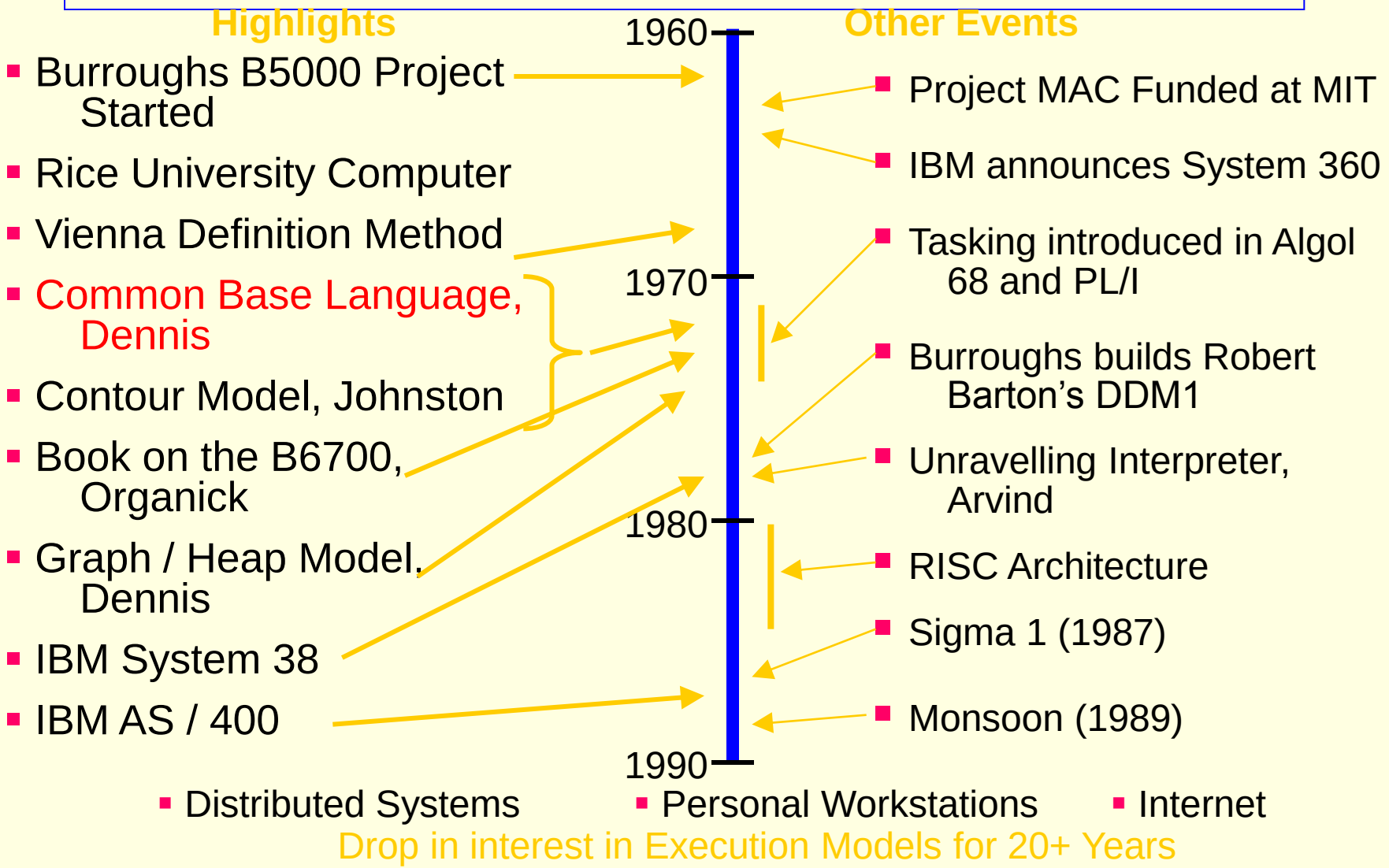
- Procedures; call/return
- Access to parameters and variables
- Use of data structures (static and dynamic)

But that's not all !!

Features expressed Outside a (typical) programming language

- File creation, naming and access
- Object directories
- Communication: networks and peripherals
- Concurrency: coordination; scheduling

Developments in the 1960s, 1970s



Contour Model: Algorithm; Nested Blocks and Contours

- Johnston, 1971

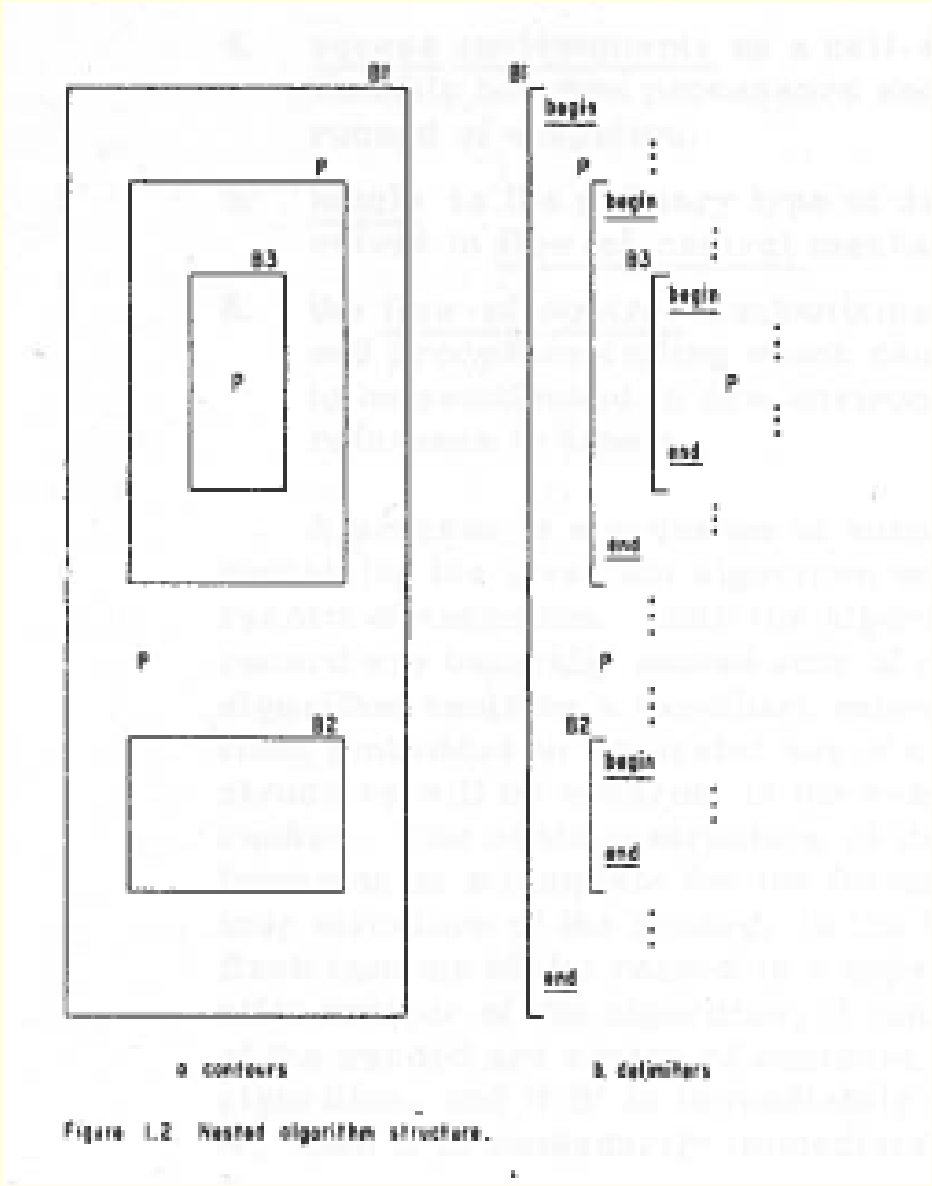
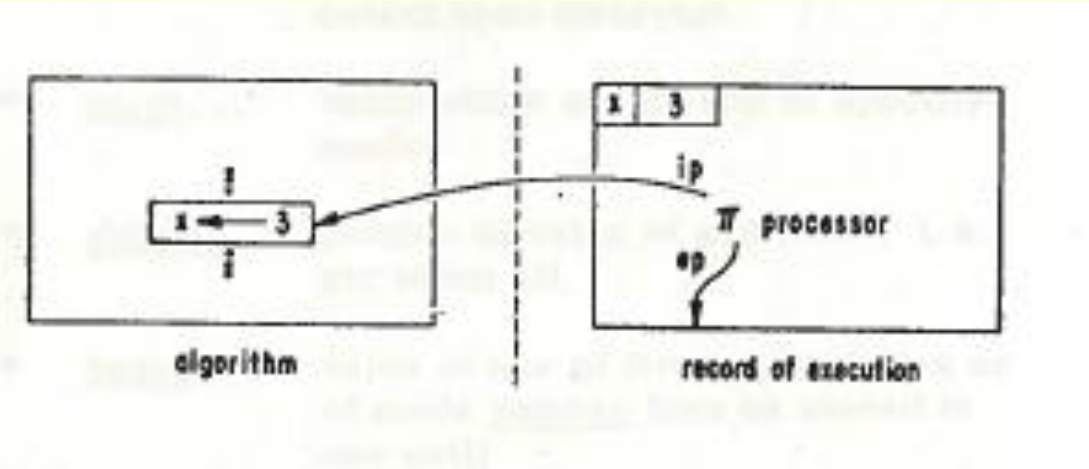


Figure 1.2 Nested algorithm structure.

Contour Model: Processor



- Johnston, 1971

Contour Model: A Snapshot

- Johnston, 1971

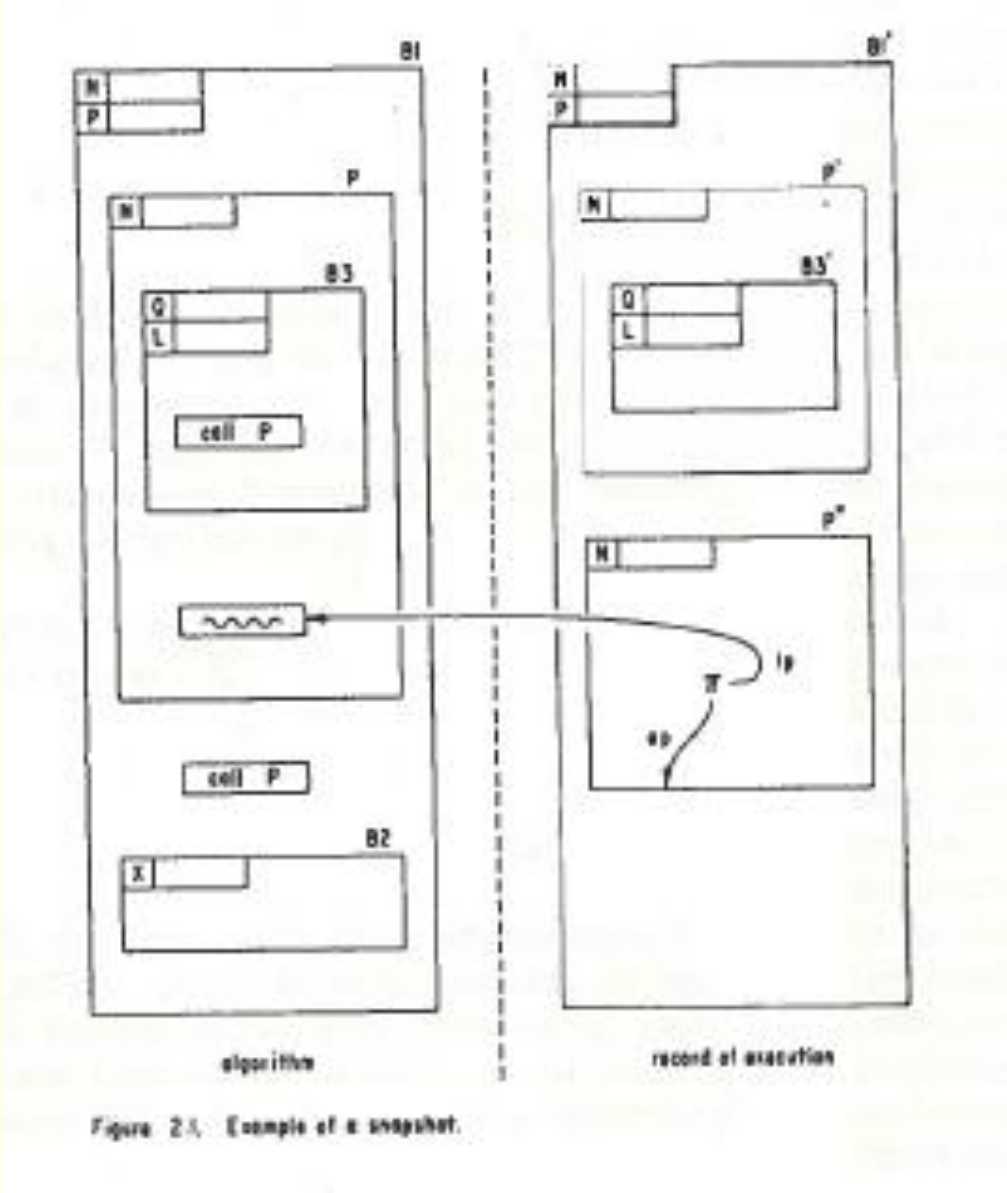


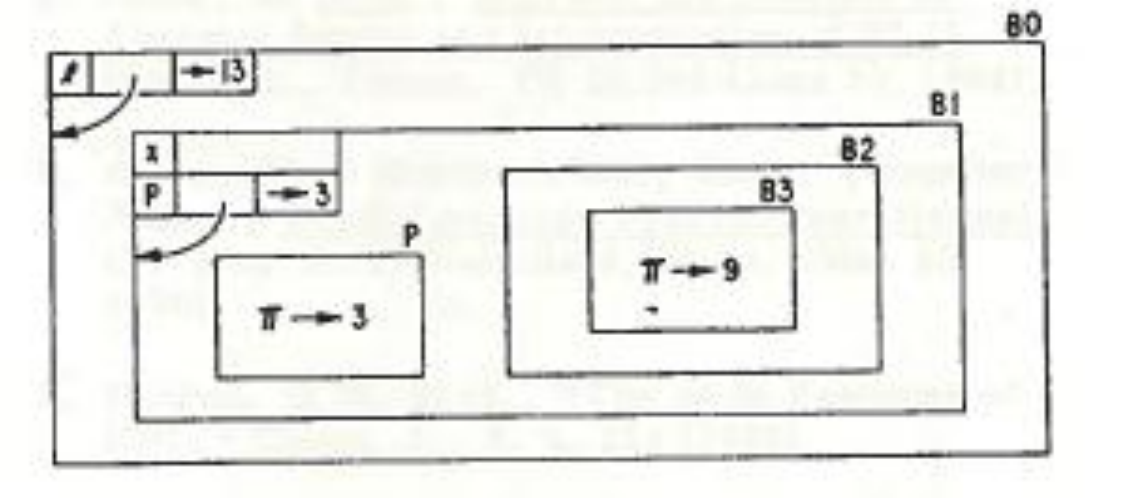
Figure 2.1 Example of a snapshot.

Two Processors Sharing Portions of Environment

- Berry, 1972

```
1  B0
       (---
2      B1   (proc p; int x
3           P   p:=<---x-
4                   --->;
5               ---
6           B2  (----
7                 p task
8           B3    (---
9                   goto ℓ;
10                )
11              )
12          )
13        ℓ:---
14    )
```

- Program with tasking



- Record of Execution

Idea: A Common Base Language

This is a report on the work of the Computation Structures Group of Project MAC toward the design of a common base language for programs and information structures. We envision that the meanings of programs expressed in practical source languages will be defined by rules of translation into the base language.

The meanings of programs in the base language is fixed by rules of interpretation which constitute a transition system called the interpreter for the base language.

We view the base language as the functional specification of a computer system in which emphasis is placed on programming generality -- the ability of users to build complex programs by combining independently written program modules.

- Dennis, 1972

What Does Program Xecution Model (PXM) Mean ?

- The notion of PXM

  The program execution model (PXM) is the basic low-level abstraction of the underlying system architecture upon which our programming model, compilation strategy, runtime system, and other software components are developed.

- The PXM (and its API) serves as *an interface between the architecture and the software*.

Program Execution Model (PXM) – Cont'd

Unlike an instruction set architecture (ISA) specification, which usually focuses on lower level details (such as instruction encoding and organization of registers for a specific processor), the PXM refers to machine organization at a higher level for a whole class of high-end machines as view by the users

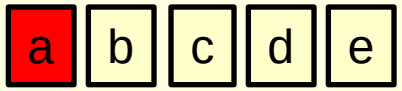
Gao, et. al., 2000

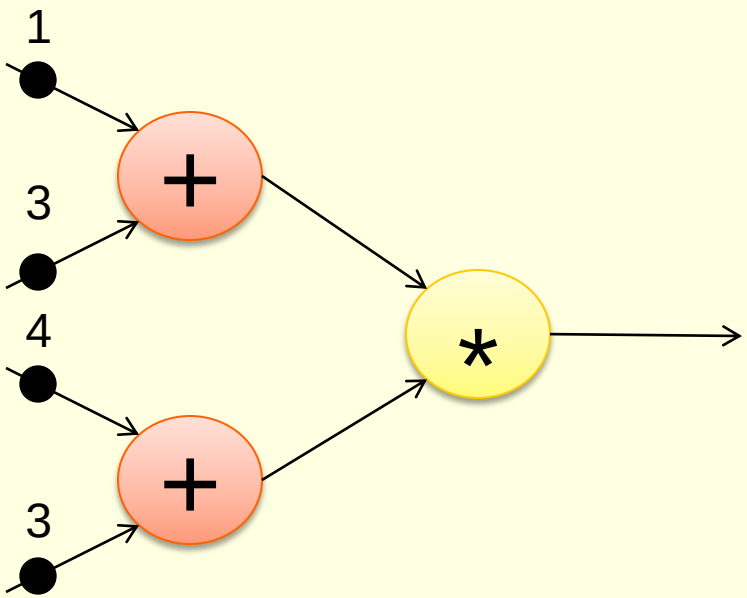
What is your "Favorite" Program Execution Model?

Outline

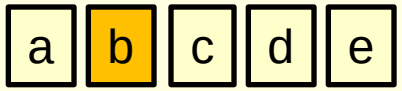
- Parallel Program Execution Models
- Dataflow Models of Computation
- Dataflow Graphs and Properties
- Three Dataflow Models
  - Static
  - Recursive Program Graph
  - Dynamic
- Dataflow Architectures

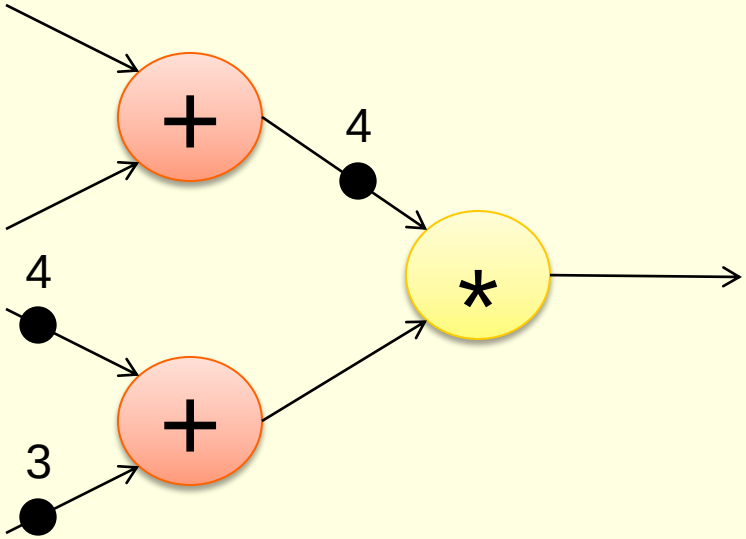
Dataflow Model of Computation



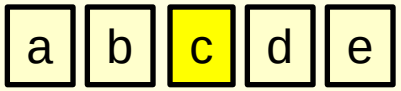


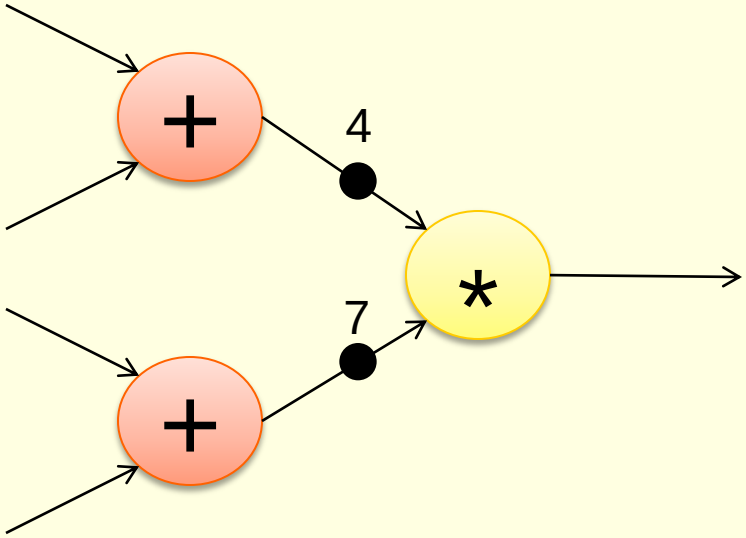
Dataflow Model of Computation



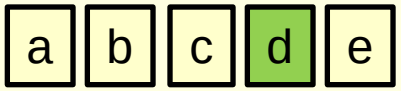


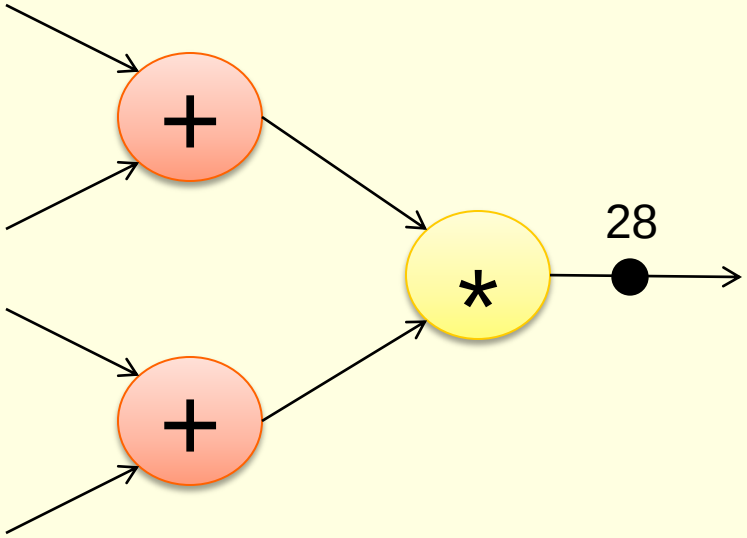
Dataflow Model of Computation



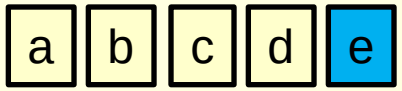


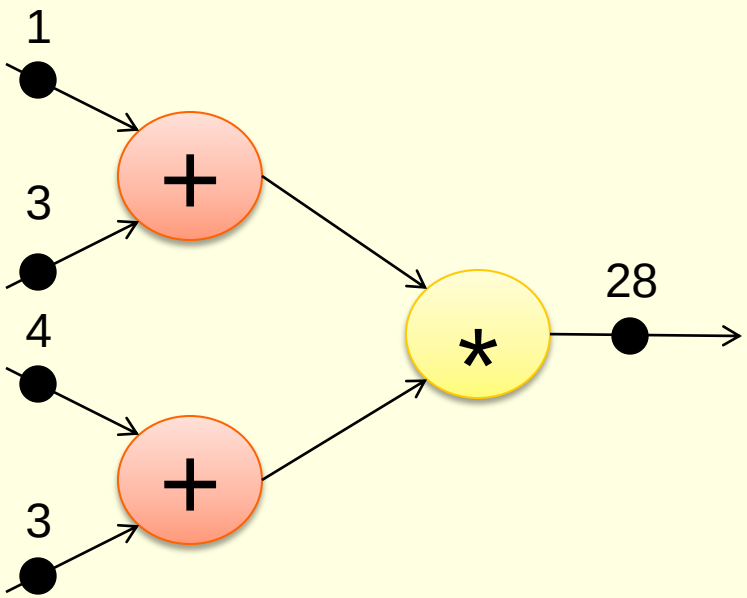
Dataflow Model of Computation





Dataflow Model of Computation





Dataflow Software Pipelining

A Base-Language

~ Data Flow Graphs ~

To serve as an intermediate-level language for high-level languages

To serve as a machine language for parallel machines

- J.B. Dennis

Jac

MIT -1964

- IBM announces System 360.
- Project Mac selects GE 645 for Multics.
- I decide to pursue research on relation of program structure to computer architecture.
- "Machine Structures Group formed."

By Jack B. Dennis

Karp, Miller

Parallel

Program

Schema

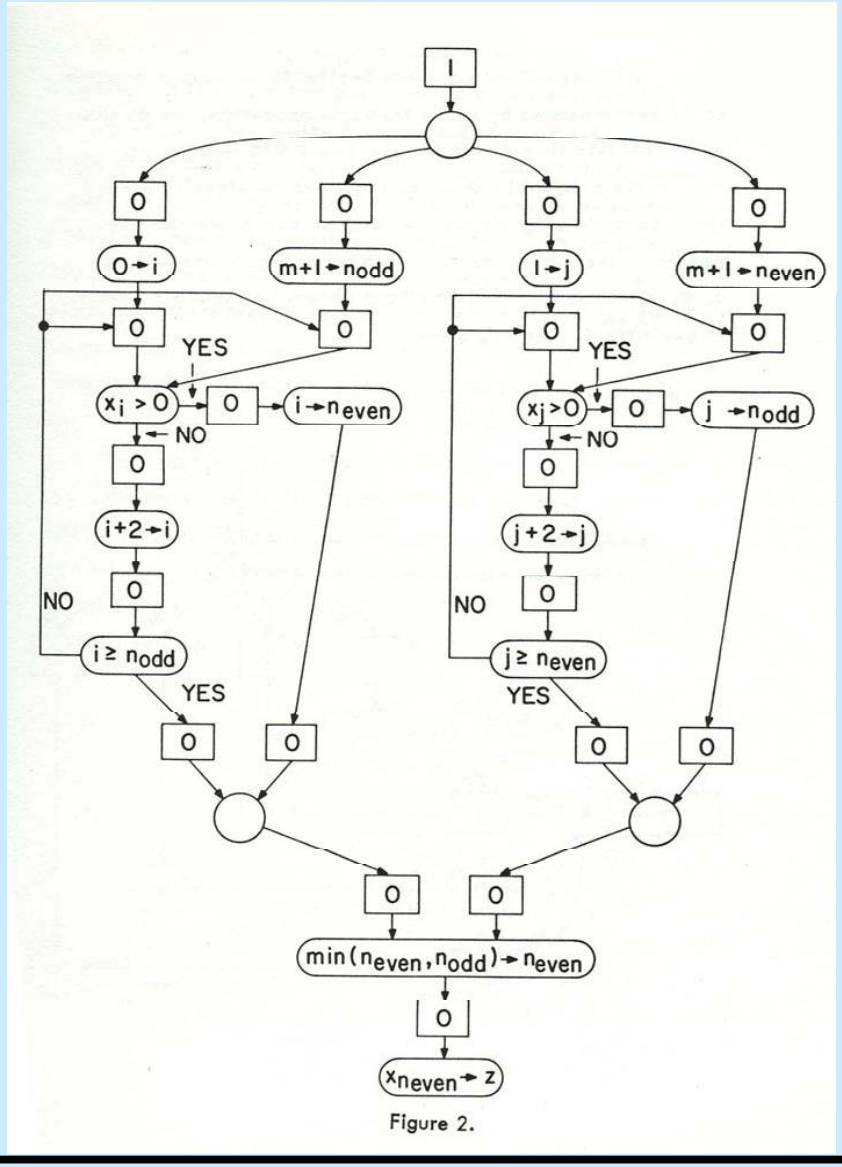


Figure 2.

Data Flow Years at MIT

1974 – 1975

- April 1974: Symposium on Programming, Paris. Dennis: "*First Version of a Data Flow Procedure Language*".
- January 1975: Second Annual Symposium on Computer Architecture, Houston. Dennis and Misunas: "*A Preliminary Architecture for a Basic Data-Flow Processo*r".
- August 1975: 1975 Sagamore Computer Conference on Parallel Processing:
  - Rumbaugh: "*Data Flow Languages*"
  - Rumbaugh: "*A Data Flow Multiprocessor*"
  - Dennis: "*Packet Communication Architecture*"
  - Misunas: *"Structure Processing in a Data-Flow Computer*"

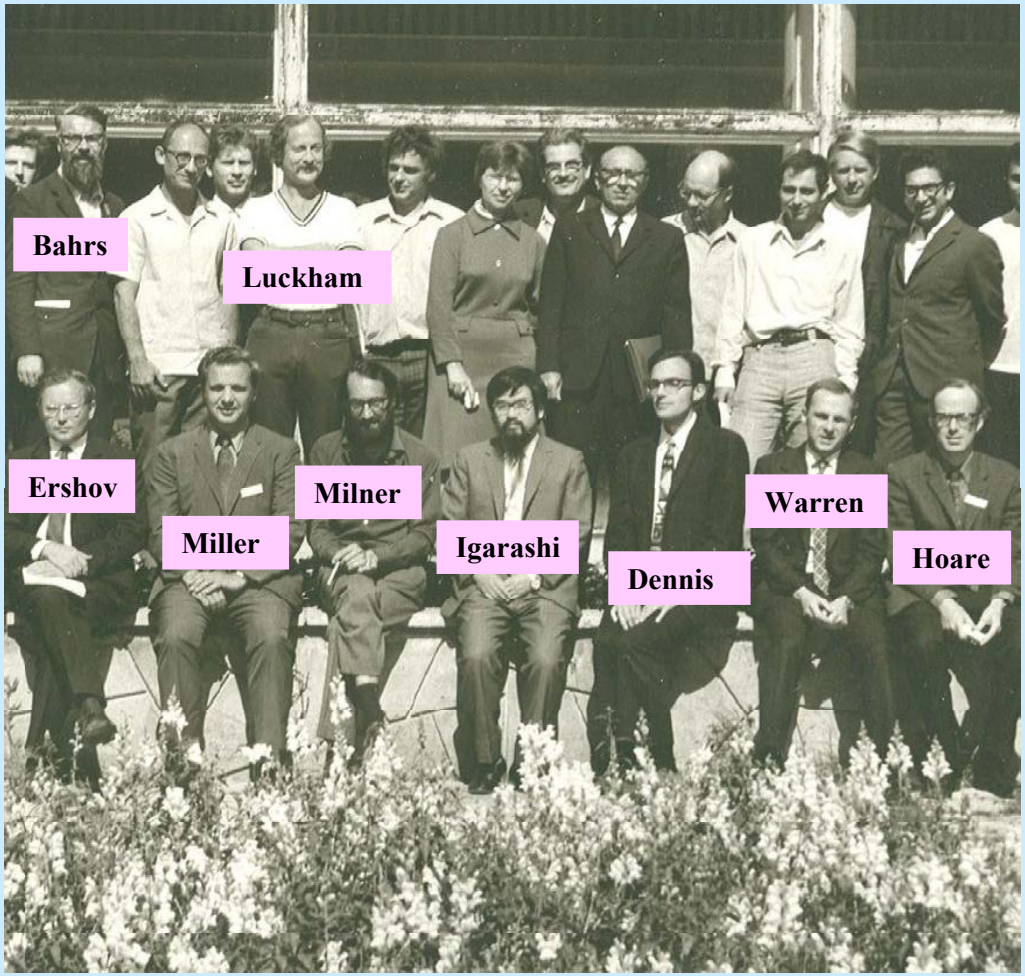
.

Early Roots on Dataflow Work at MIT in 70s

- Asynchronous Digital Logic [Muller, Bartky]
- Control Structures for Parallel Programming: [Conway, McIlroy, Dijkstra]
- Abstract Models for Concurrent Systems: [Petri, Holt]
- Theory of Program Schemes [Ianov, Paterson]
- Structured Programming [Dijkstra, Hoare]
- Functional Programming [McCarthy, Landin]

Symposium on Theoretical Programming Novosibirsk – 1972

Notables – Novosibirsk -1972

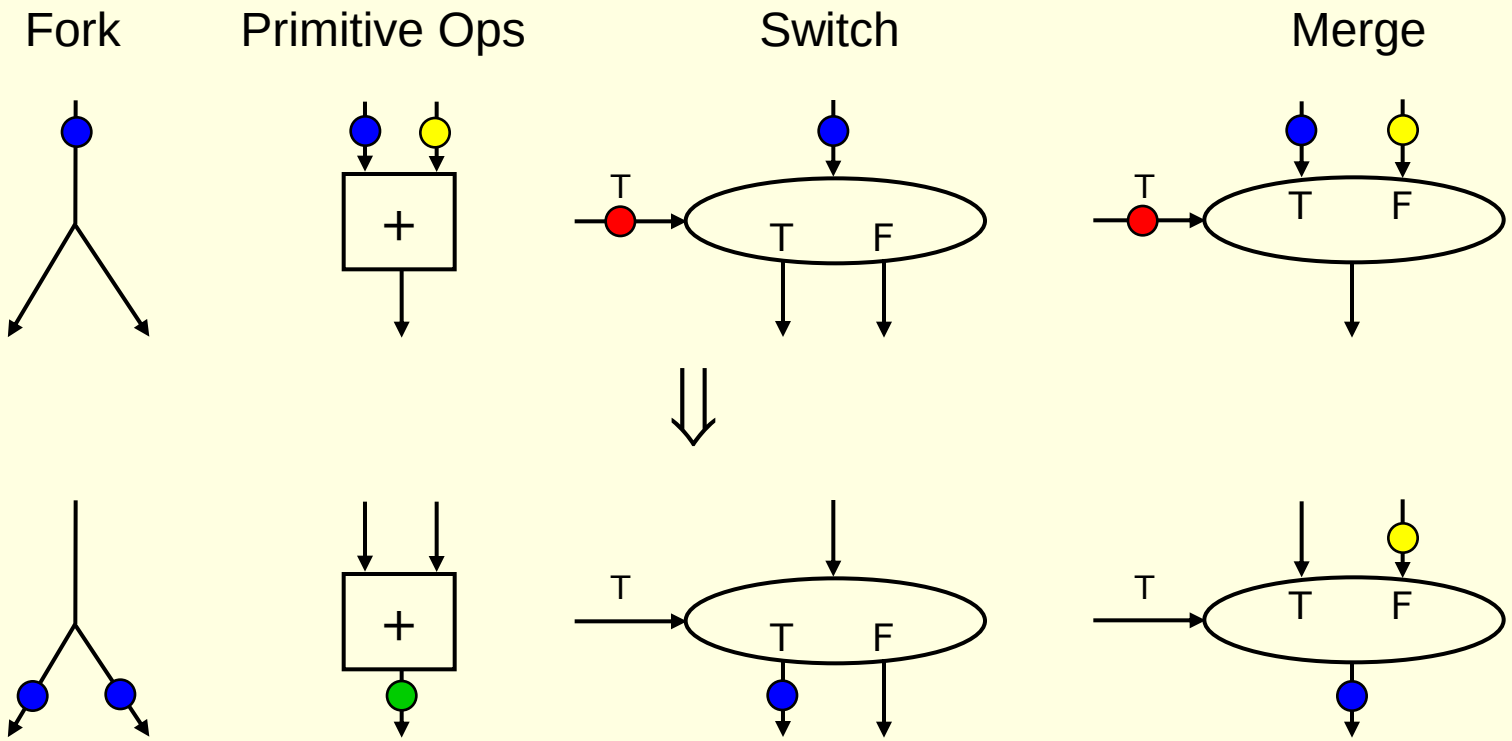


Outline

- Parallel Program Execution Models
- Dataflow Models of Computation
- Dataflow Graphs and Properties
- Three Dataflow Models
  - Static
  - Recursive Program Graph
  - Dynamic
- Dataflow Architectures

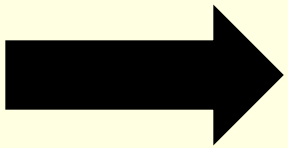
Dataflow Operators

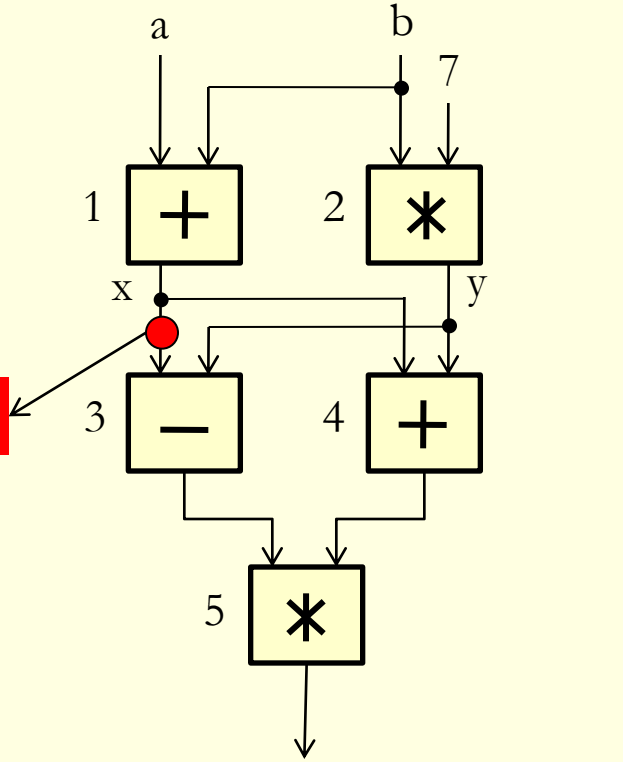
- A small set of dataflow operators can be used to define a general programming language



Dataflow Graphs

```
x = a + b;
z = b * 7;
z = (x-y) * (x+y);
```





Values in dataflow graphs are represented as tokens of the form:

<ip, p, v>

<3, Left, value>

Where ***ip*** is the instruction pointer ***p*** is the port and ***v*** represents the 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

Operational Semantics
(Firing Rule)

- Values represented by tokens
- Placing tokens on the arcs (assignment)
  - snapshot/configuration: state
- Computation

  configuration ⟶ configuration

Operational Semantics Firing Rule

- Tokens ➔ Data
- Assignment ➔ Placing a token in the output arc
- Snapshot / configuration: state
- Computation

  The intermediate step between snapshots / configurations
- An actor of a dataflow graph is enabled if there is a token on each of its input arcs

Operational Semantics Firing Rule

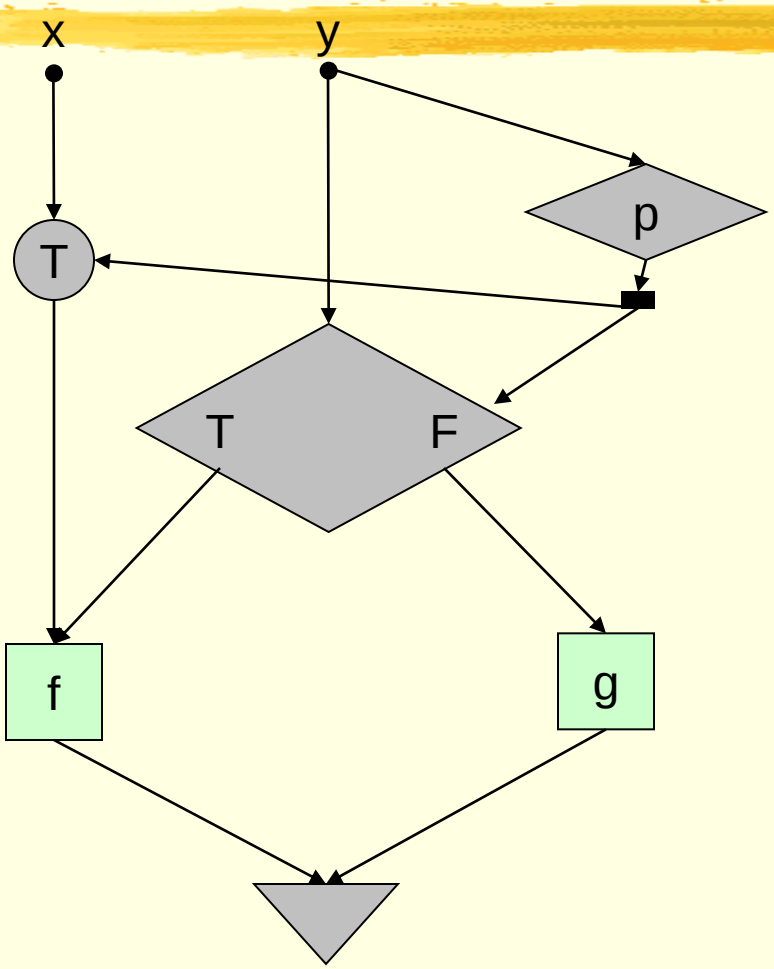
- Any enabled actor may be fired to define the "next state" of the computation
- An actor is fired by removing a token from each of its input arcs and placing tokens on each of its output arcs.
- Computation ➔ A Sequence of Snapshots
  - Many possible sequences as long as firing rules are obeyed
  - Determinacy
  - "Locality of effect"

General Firing Rules

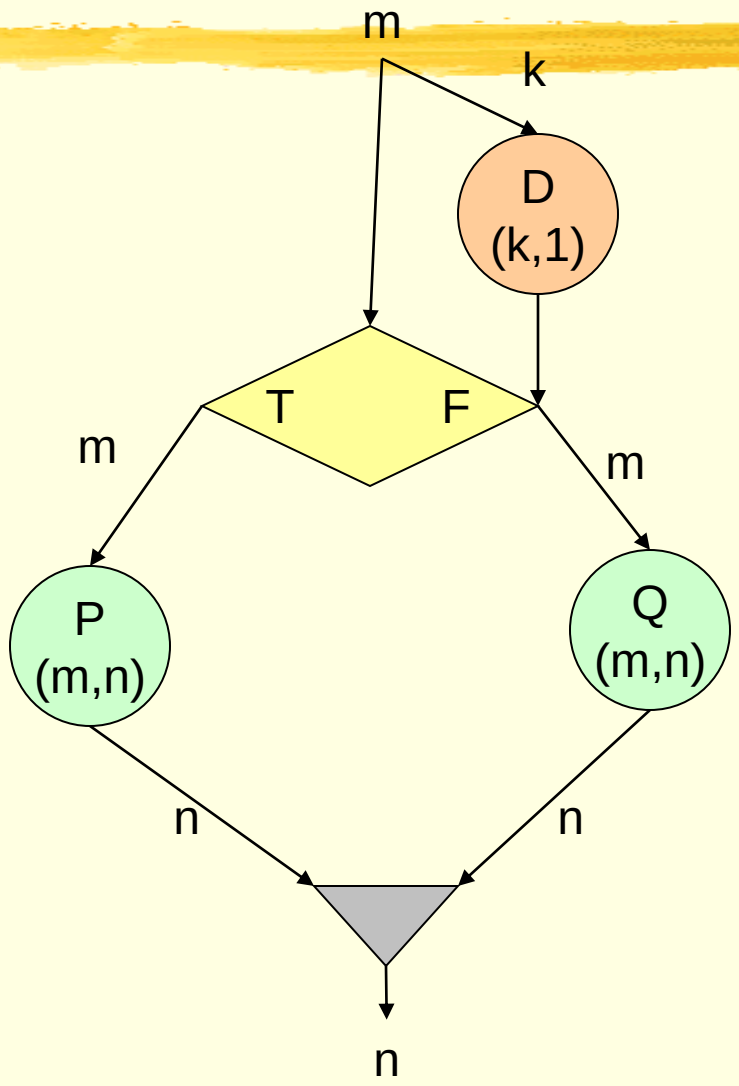
- A switch actor is enabled if a token is available on its control input arc, as well as the corresponding data input arc.
  - The firing of a switch actor will remove the input tokens and deliver the input data value as an output token on the corresponding output arc.
- A (unconditional) merge actor is enabled if there is a token available on any of its input arcs.
  - An enabled (unconditional) merge actor may be fired and will (non-deterministically) put one of the input tokens on the output arc.

Conditional Expression

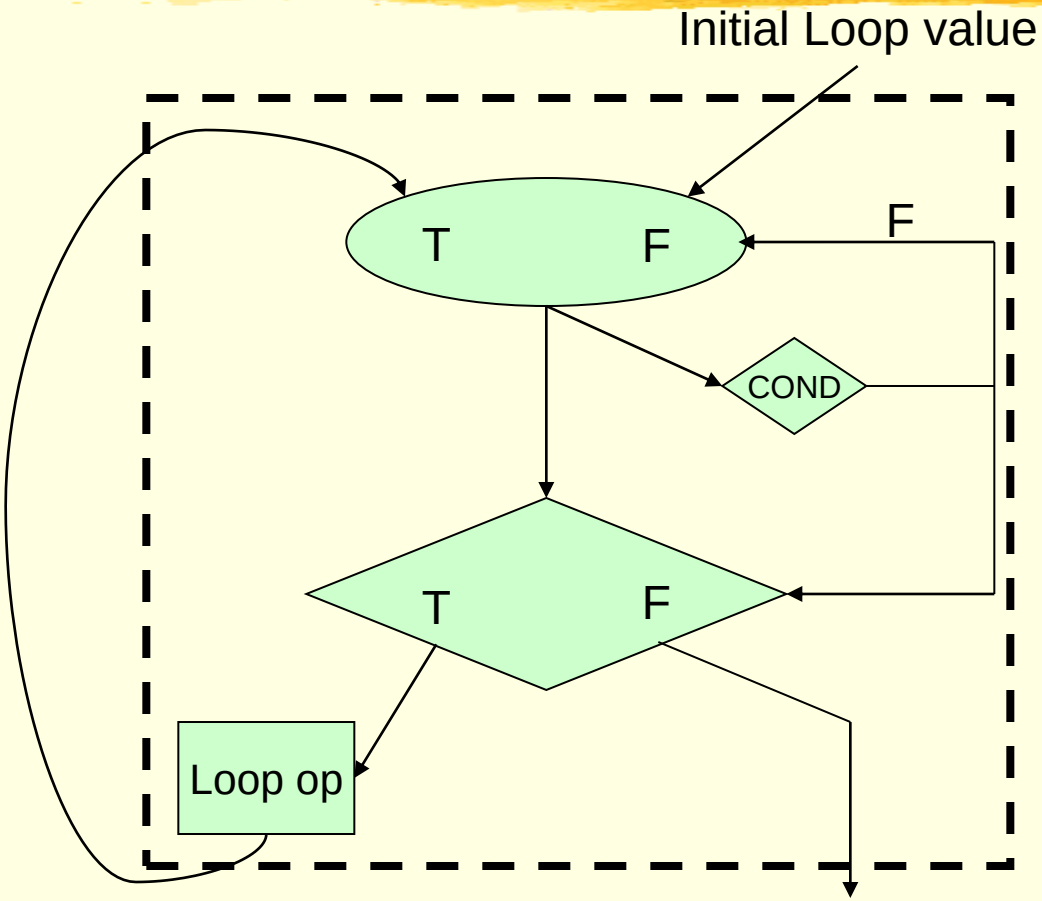
```
if (p(y)){
   f(x,y);
}
else{
   g(y);
}
```



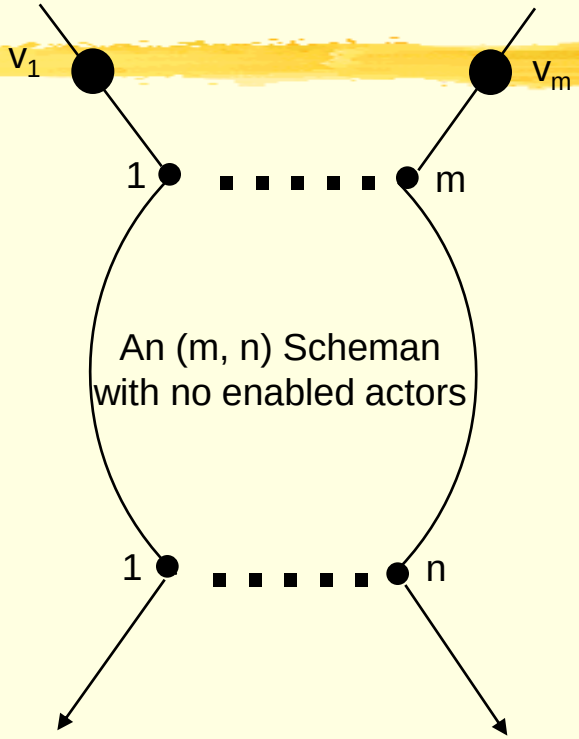
A Conditional Schema



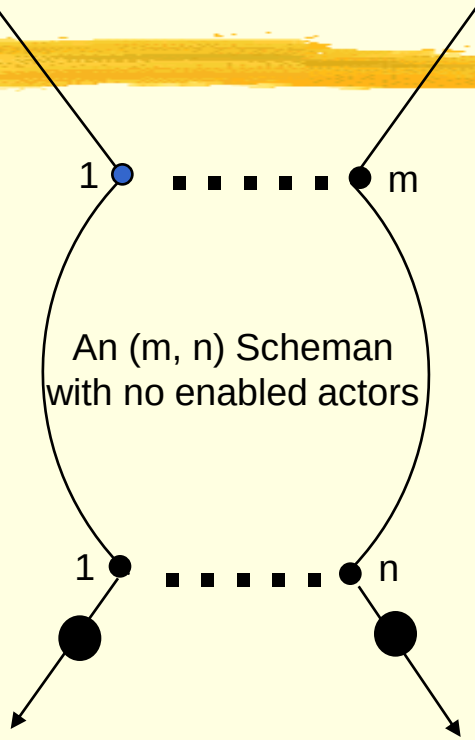
A Loop Schema



Properties of Well-Behaved Dataflow Schemata



(a) Initial Snapshot



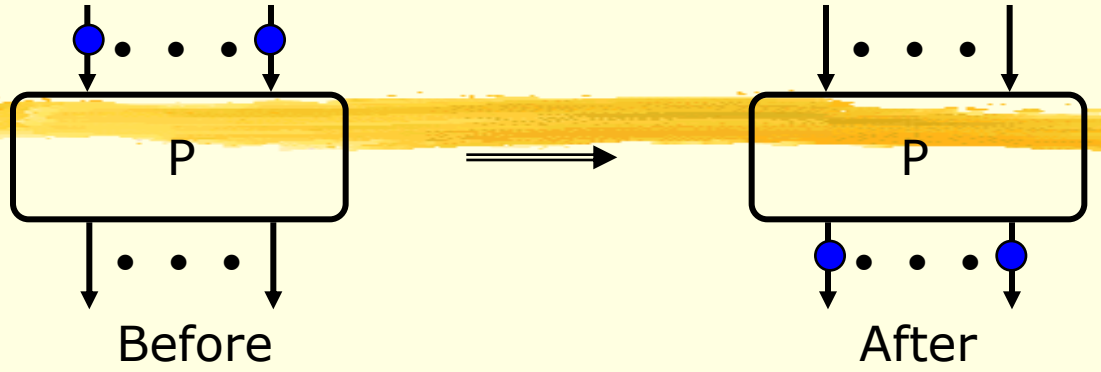
(a) Final Snapshot

Well-behaved Data Flow Graphs

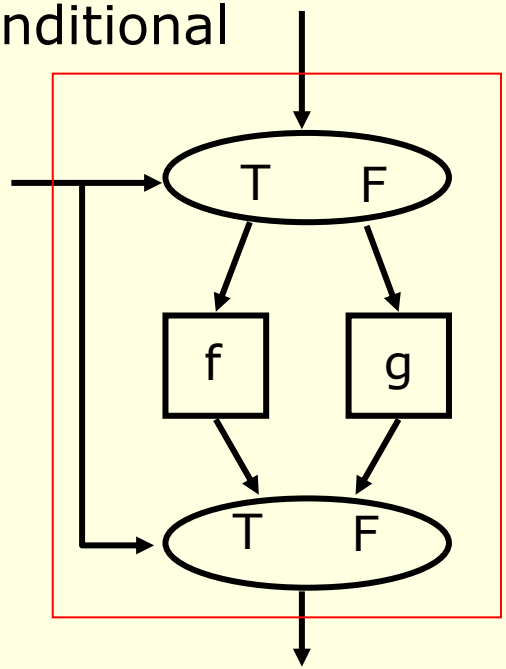
- Data flow graphs that produce exactly one set of result values at each output arcs for each set of values presented at the input arcs
- Implies the initial configuration is re-established
- Also implies *determinacy*

Well Behaved Schemas

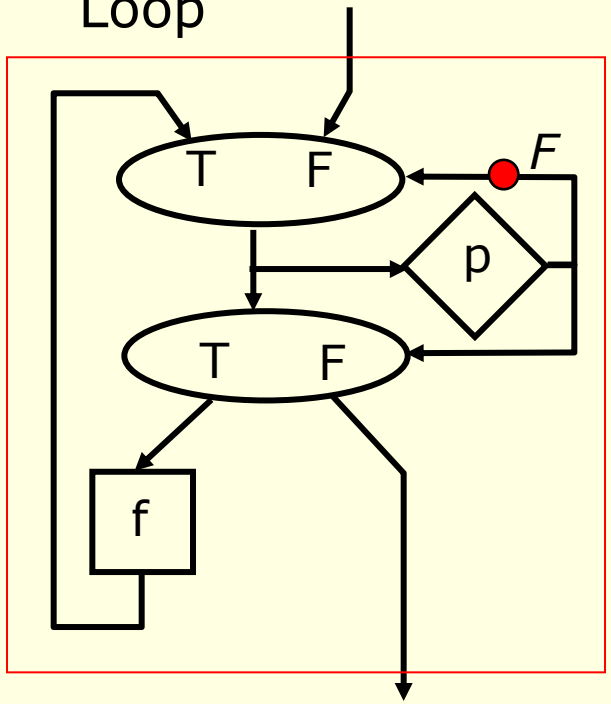
one-in-one-out
& self cleaning



Conditional



Loop



Well-formed Dataflow Schema
(Dennis & Fossen 1973)

- An operator is a WFDS
- A conditional schema is a WFDS
- A iterative (loop) schema is a WFDS
- An acyclic composition of component WFDS is a WFDS

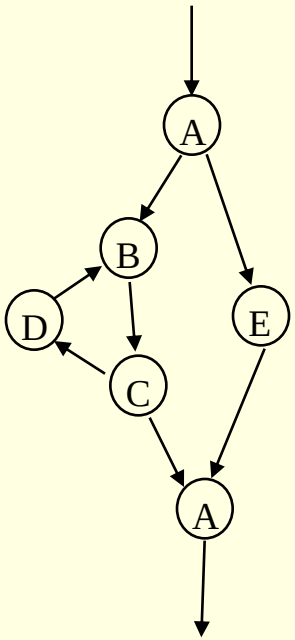
Theorem

"A well-formed data flow graph is well-behaved"

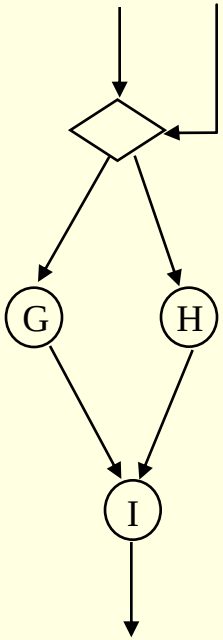
proof by induction

Example of "Sick" Dataflow Graphs

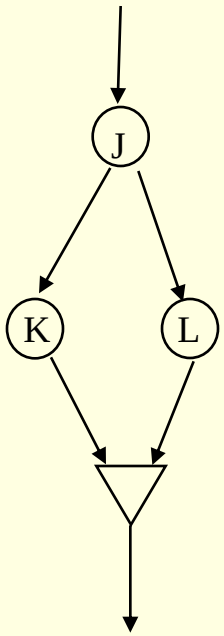
Arbitrary connections of data flow operators can result in pathological programs, such as the following:



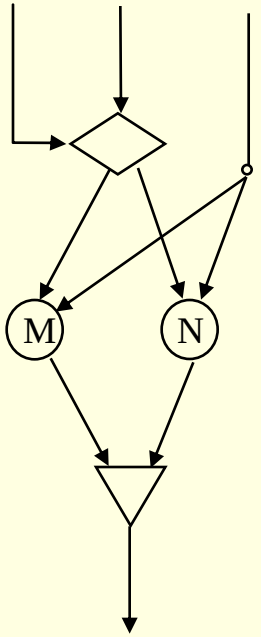
1. Deadlock



2. Hangup



3. Conflict



4. Unclean

Well-behaved Program

- Always determinate in the sense that a unique set of output values is determined by a set of input values
- References:

Rodriquez, J.E. 1966, “A Graph Model of Parallel Computation”, MIT, TR-64]
Patil, S. “Closure Properties of Interconnections of Determinate Systems”, Records of the project MAC conf. on concurrent systems and parallel Computation, ACM, 1970, pp 107-116]
Denning, P.J. “On the Determinacy of Schemata” pp 143-147
Karp, R.M. & Miller, R.E., “Properties of a Model of Parallel Computation Termination, termination, queuing”, Appl. Math, 14(6), Nov. 1966

Remarks on Dataflow Models

- A fundamentally sound and simple parallel model of computation (features very few other parallel models can claim)
- Few dataflow architecture projects survived passing early 1990s. But the ideas and models live on ..
- In the new multi-core age: we have many reasons to re-examine and explore the original dataflow models and learn from the past

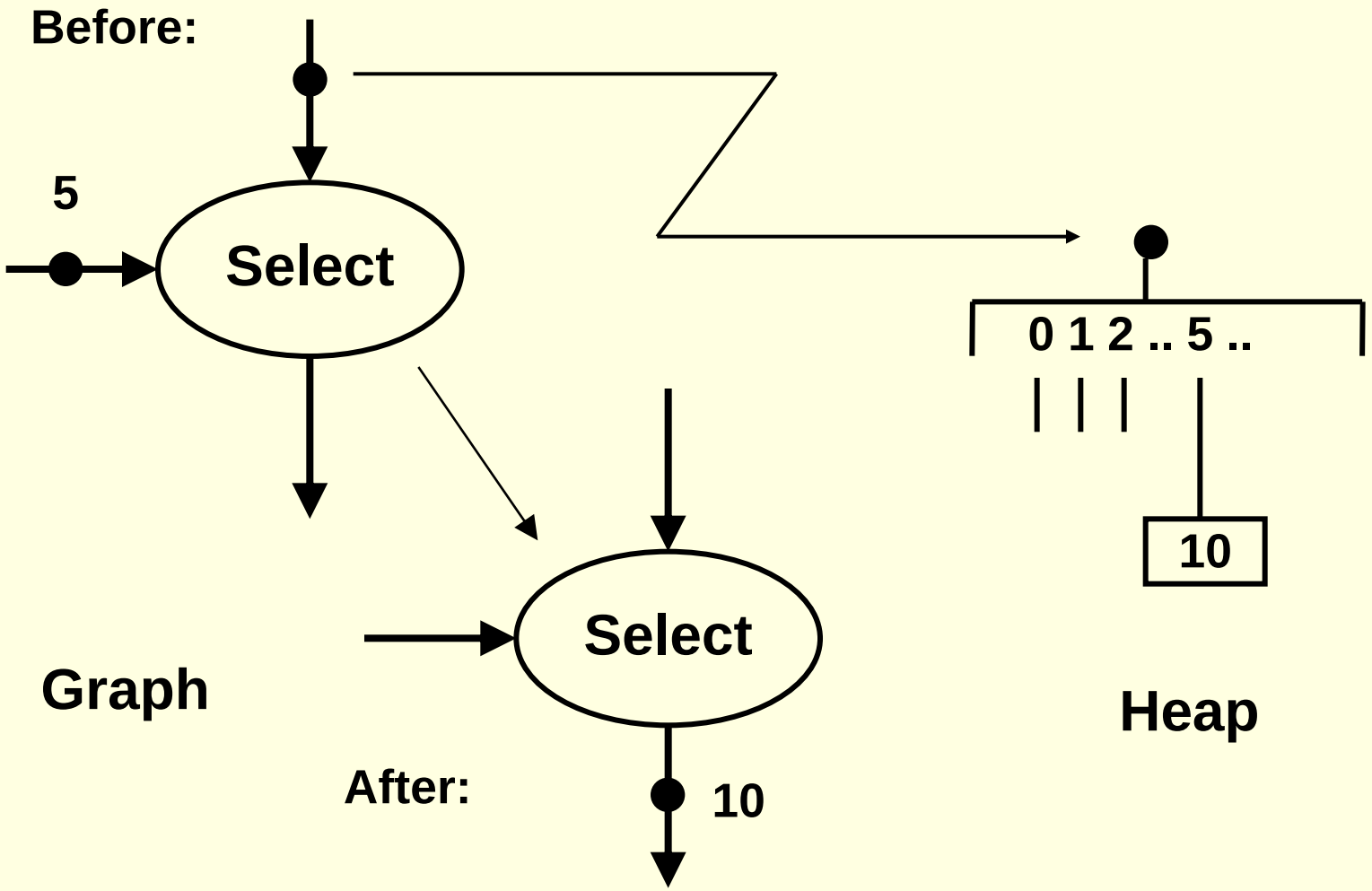
Graph / Heap Model Of Program Execution

In our semantic model for extended data flow programs, values are represented by a heap, which is a finite, acyclic, directed graph having one or more root nodes, and such that each node of the heap may be reached over some path from some root node.

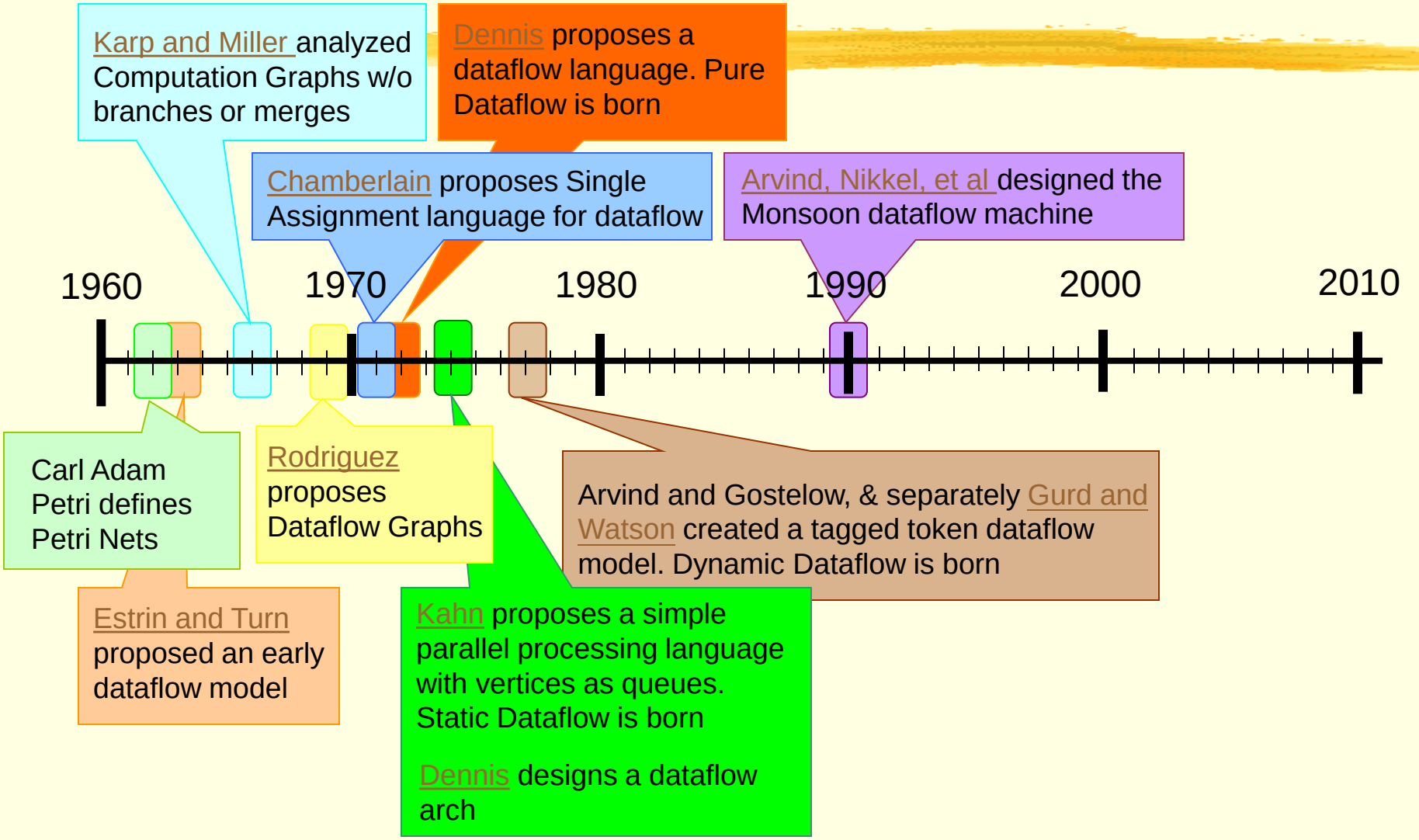
- A snapshot of a data flow program in execution will now have two parts: a token distribution on the graph of the program, and a heap.
  - For each execution step some enabled link or actor is selected to fire; the result of firing is a new token distribution,
  - and in some cases, a modified heap.

- Dennis, 1974

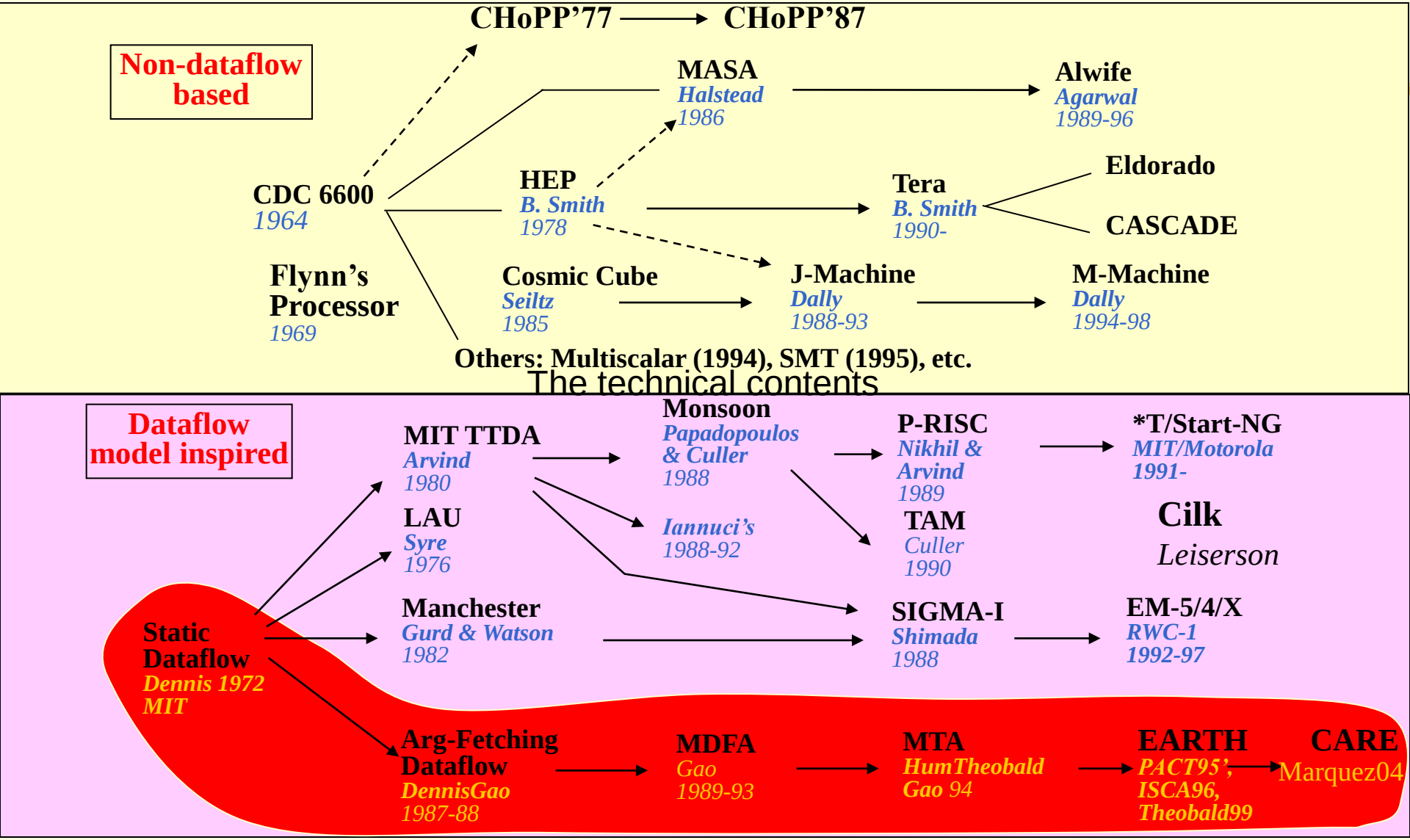
The Graph and Heap Model



A Short Story



Evolution of Multithreaded Execution and Architecture Models



Jack's History Note

Prof. Estrin was author of a number of paper relating to parallel graph models for omputation.

The ones I recall were written with Prof. David W--- (?) who was a visiting scientist at MIT for a year or so (I don't recall what year).

The Dennis Static data flow model was implicit in the Dennis,Misunas 1975 paper for ISCA and was the subject of my lectures as IEEE Distinguished speaker, but I can't quickly determine the year. I presented a definitive paper at the "Symposium on Theoretical Programming", Novosibirsk, 1972, and it was published in LNCS. If I recall correctly it was CSG Memo 81, but a copy is not in my file. So I think the date (1972) for static data flow on the second slide is correct (and I believe precedes Kahn). So I think the box "Dennis proposes ... " is wrong (perhaps depending on what is meant by "pure dataflow").

My view is that my 1974 paper is the first treatment of a

reasonably complete "dynamic" data flow model, including

arbitrary recursion and tree-structured data objects (to be

followed in two or three years by Arvin/Gostelow/Plouff

Jack Dennis

Personal Communication

Sept. 11, 2011

Some Note on History

Some History on Dataflow