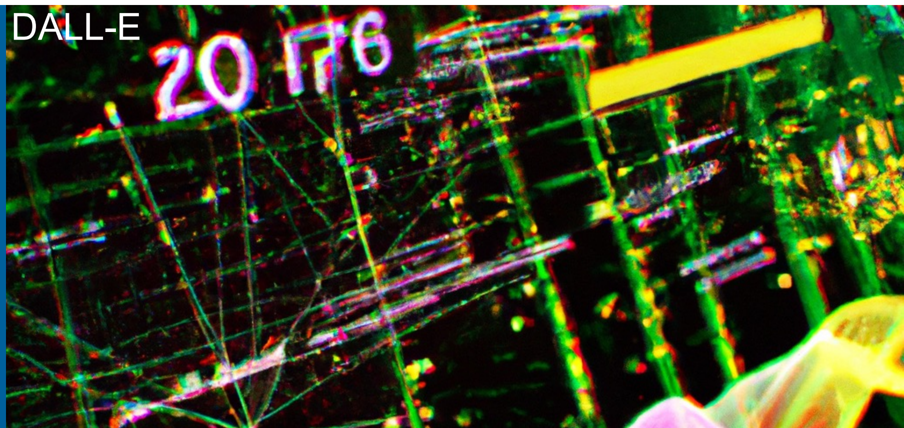
ATPESC 2023 – TRACK 1: HARDWARE ARCHITECTURES

Argonne NATIONAL LABORATORY

INTRODUCTION ON DATAFLOW ARCHITECTURES AND TRENDS



JOSE M MONSALVE DIAZ
Postdoctoral Researcher
Mathematics and Computer science
Argonne National Laboratory

SIDDHISANKET (SID) RASKAR
Postdoctoral Researcher
Argonne Leadership Computing Facility
Argonne National Laboratory

July 31st, 2023
St. Charles, IL

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

OUTLINE

Introduction on Dataflow Architectures and Trends

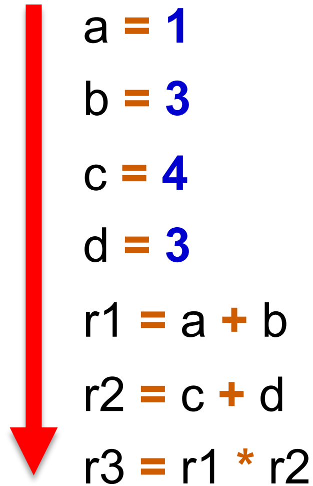
- Von Neumann vs Dataflow
- Dataflow Model of computation
- Evolution of Dataflow architectures and Advanced Concepts
- Modern Architectures
- Challenges of Dataflow architectures

VON NEUMANN VS DATAFLOW

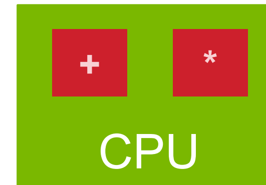
VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions

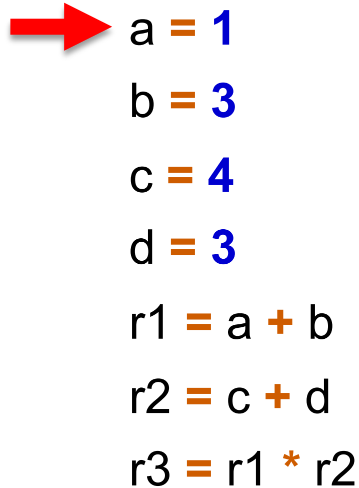


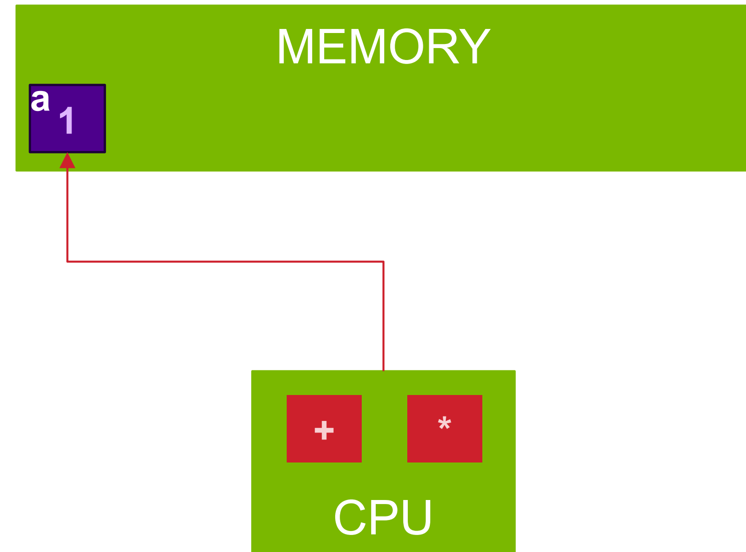
MEMORY



VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES





VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions

a = 1

b = 3

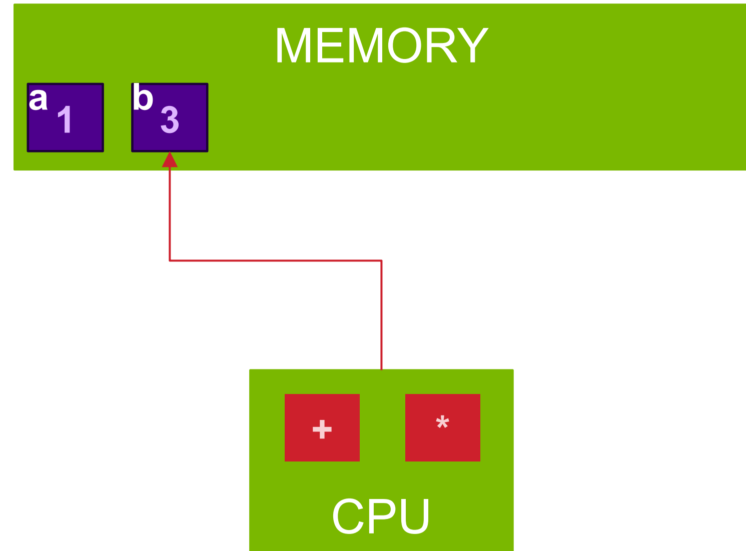
c = 4

d = 3

r1 = a + b

r2 = c + d

r3 = r1 * r2



VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions

a = 1

b = 3

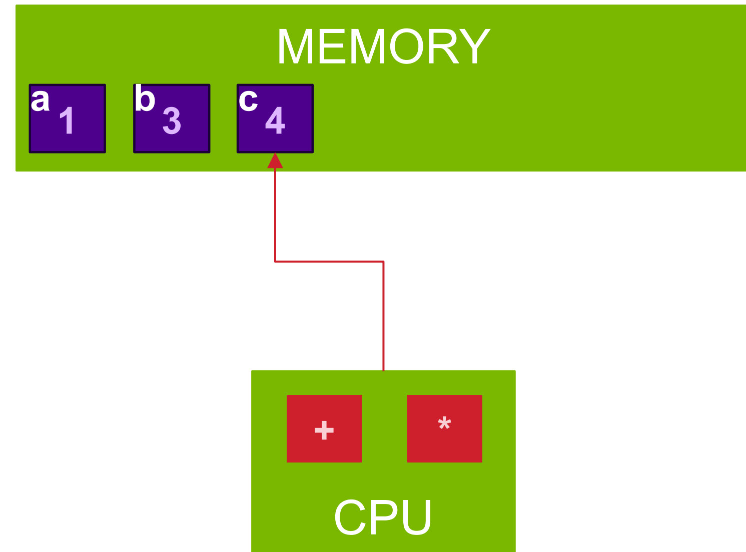
c = 4

d = 3

r1 = a + b

r2 = c + d

r3 = r1 * r2



VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions

a = 1

b = 3

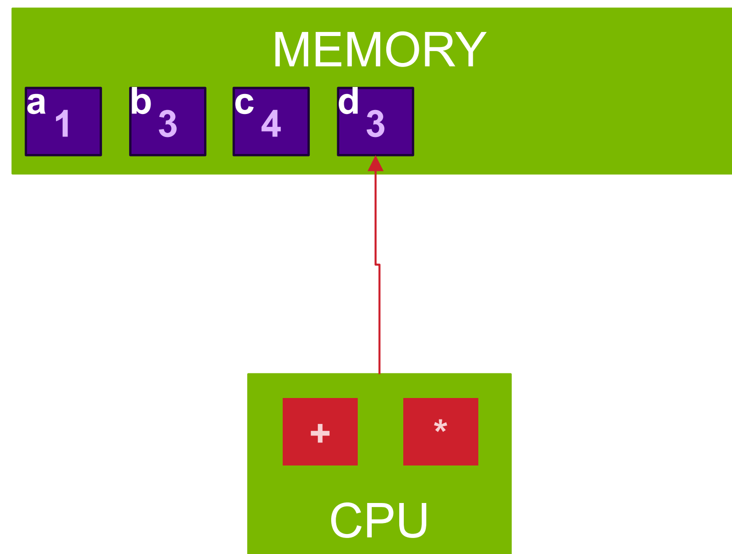
c = 4

d = 3

r1 = a + b

r2 = c + d

r3 = r1 * r2



VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions

a = 1

b = 3

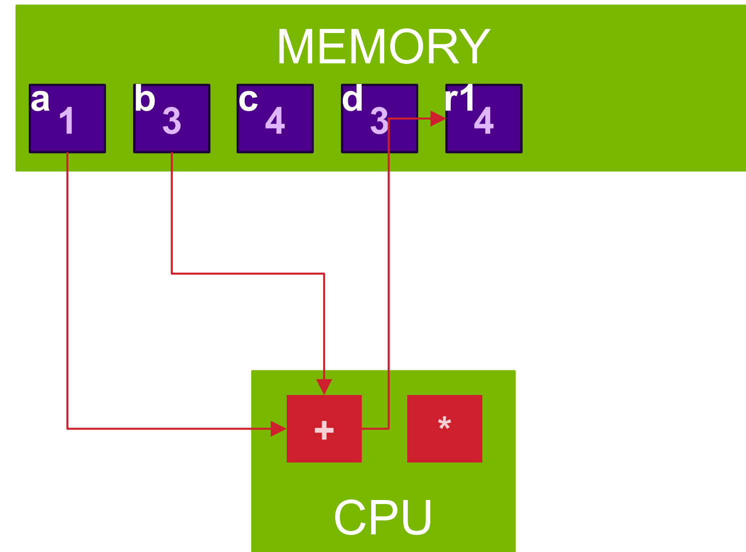
c = 4

d = 3

r1 = a + b

r2 = c + d

r3 = r1 * r2



VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions

a = 1

b = 3

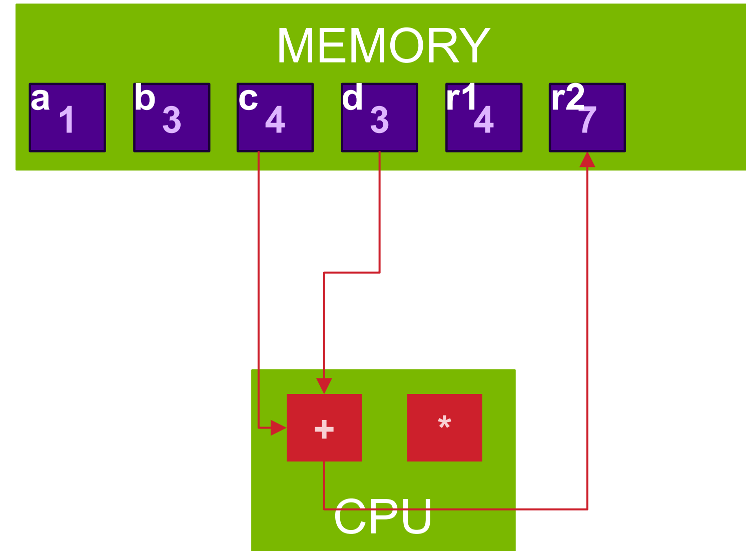
c = 4

d = 3

r1 = a + b

r2 = c + d

r3 = r1 * r2



VON NEUMANN

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions

a = 1

b = 3

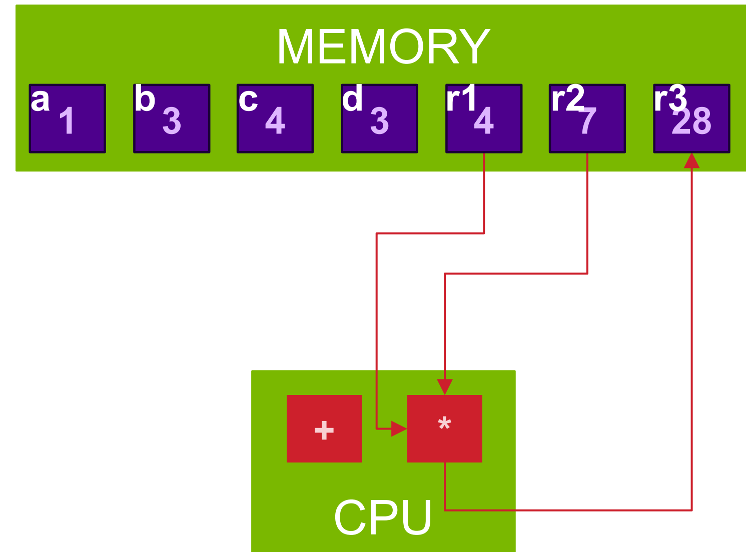
c = 4

d = 3

r1 = a + b

r2 = c + d

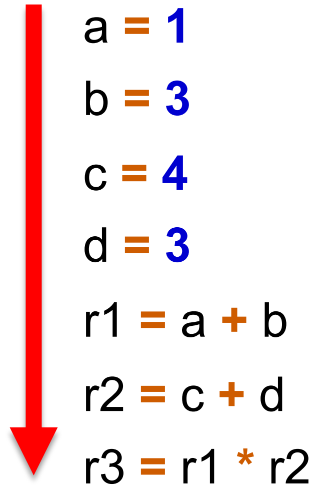
r3 = r1 * r2



VON NEUMANN VS DATAFLOW

SEQUENTIAL VON NEUMANN ARCHITECTURES

Instructions



a = 1

b = 3

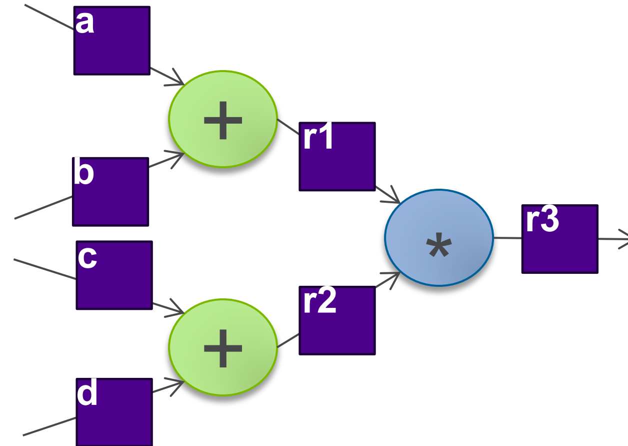
c = 4

d = 3

r1 = a + b

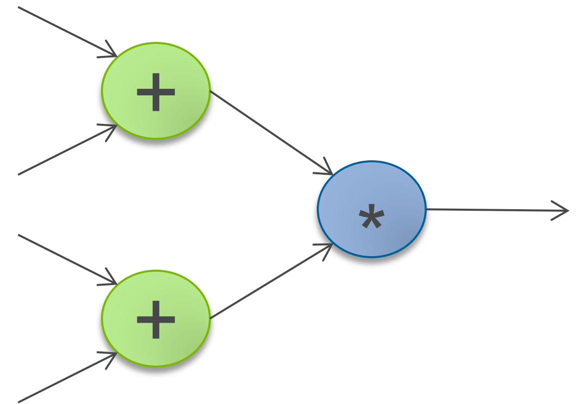
r2 = c + d

r3 = r1 * r2

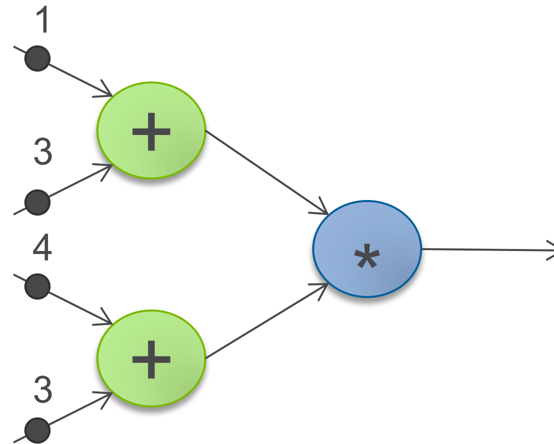


DATAFLOW PROGRAMS

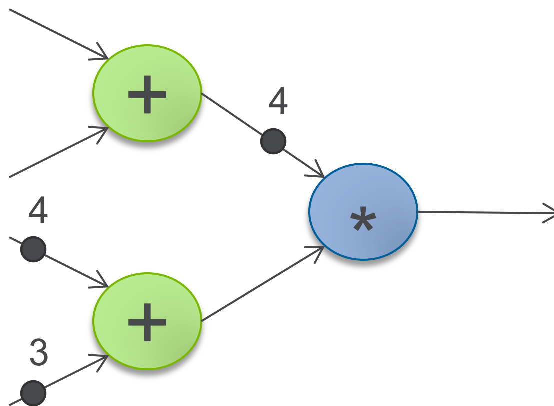
A program is represented as a graph. Nodes are operations. Arcs are operands that contain tokens.



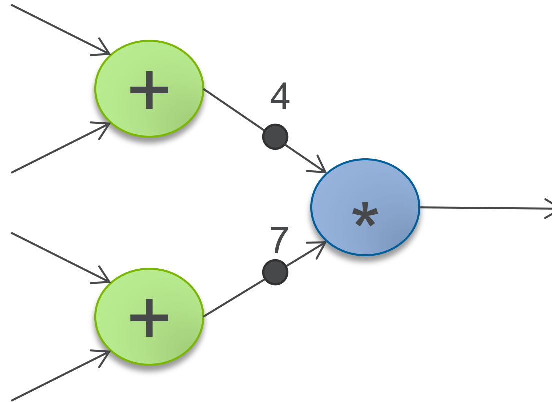
DATAFLOW MODEL OF COMPUTATION



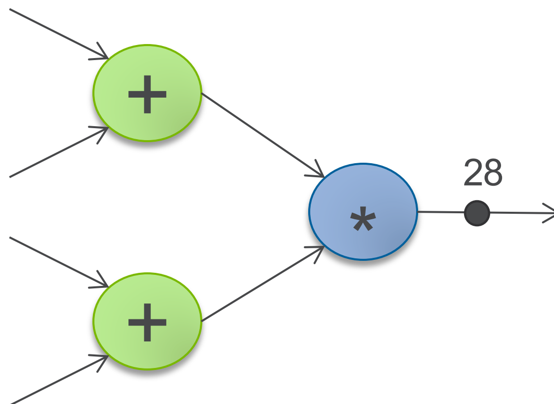
DATAFLOW MODEL OF COMPUTATION



DATAFLOW MODEL OF COMPUTATION



DATAFLOW MODEL OF COMPUTATION



DATAFLOW MODEL OF COMPUTATION

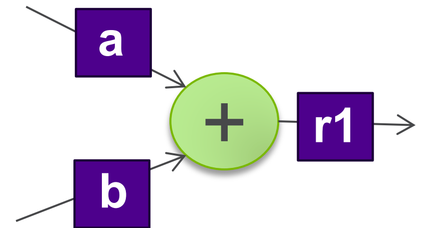
Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

DATAFLOW MODEL OF COMPUTATION

- Define what a program is
- What are the operands
- What is well defined dataflow graphs?

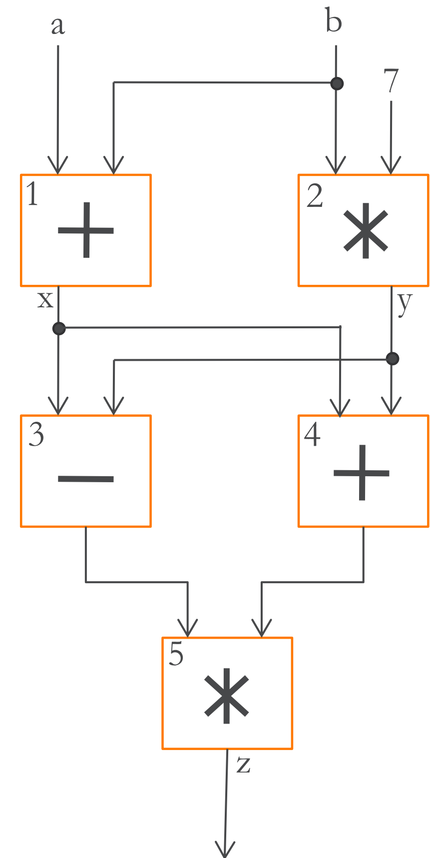
DATAFLOW OPERATIONAL SEMANTICS

- Tokens ➔ Data values
- Firing Rules ➔ All tokens are present in the input arcs
  – Actor removes tokens from each of its input arcs
  – Actor executes operation
  – Actor places tokens on each of its output arcs
- Assignment operation ➔ Placing a token in the output arc



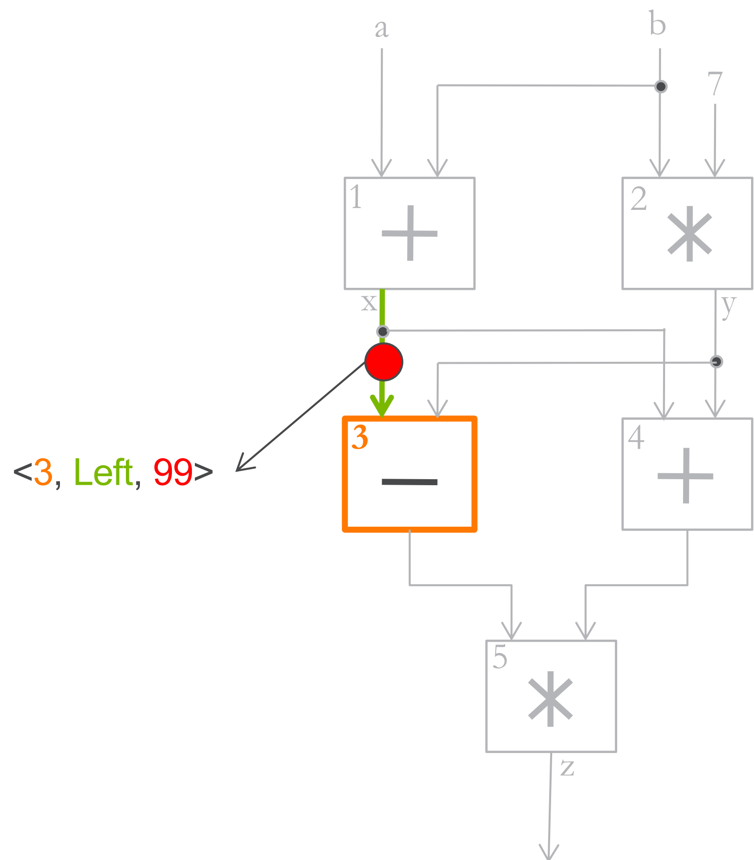
DATAFLOW GRAPHS

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



DATAFLOW GRAPHS

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



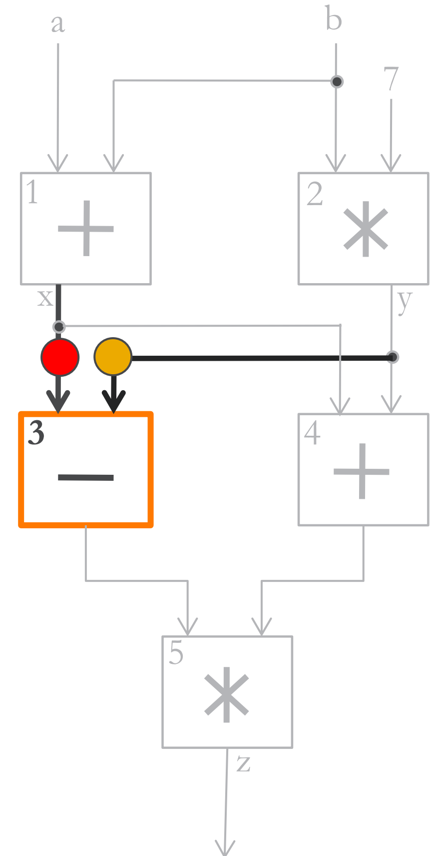
Values in dataflow graphs represented as tokens

<*instruction_ptr*, *port*, *value*>

DATAFLOW GRAPHS

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

No separate control flow

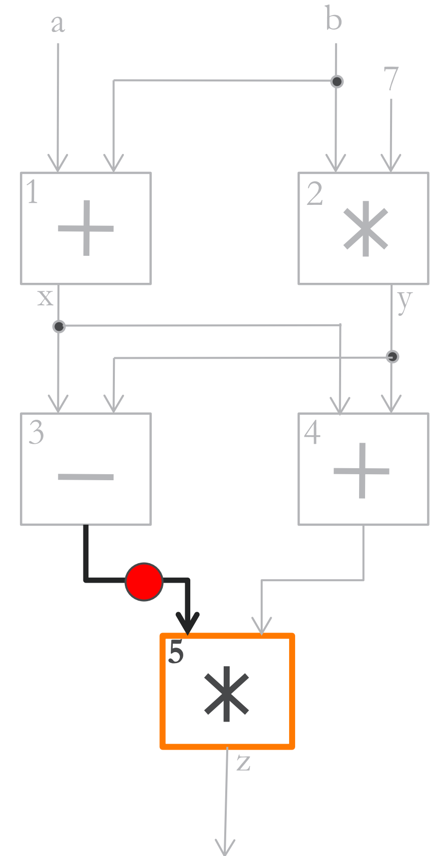


An **operator executes** when all its input tokens are present

DATAFLOW GRAPHS

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

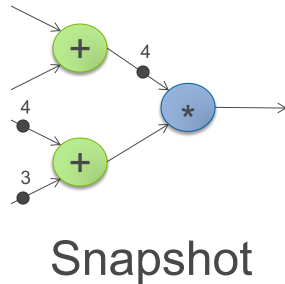
No separate control flow



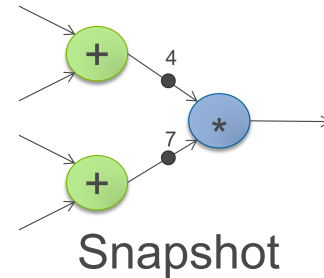
Copies of the **result token** are distributed to the destination operators

DATAFLOW OPERATIONAL SEMANTICS

- Snapshot/Configuration ➔ State of the program
- Next state ➔ Any enabled actor fired defines the “next state” of the computation

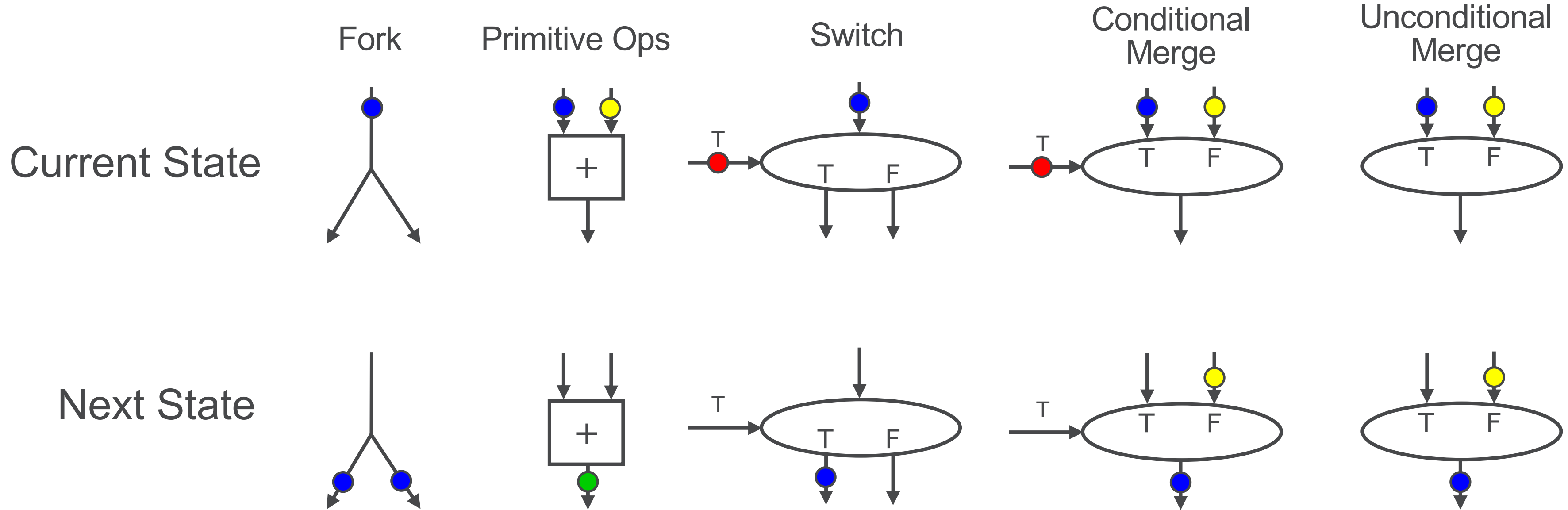


Snapshot



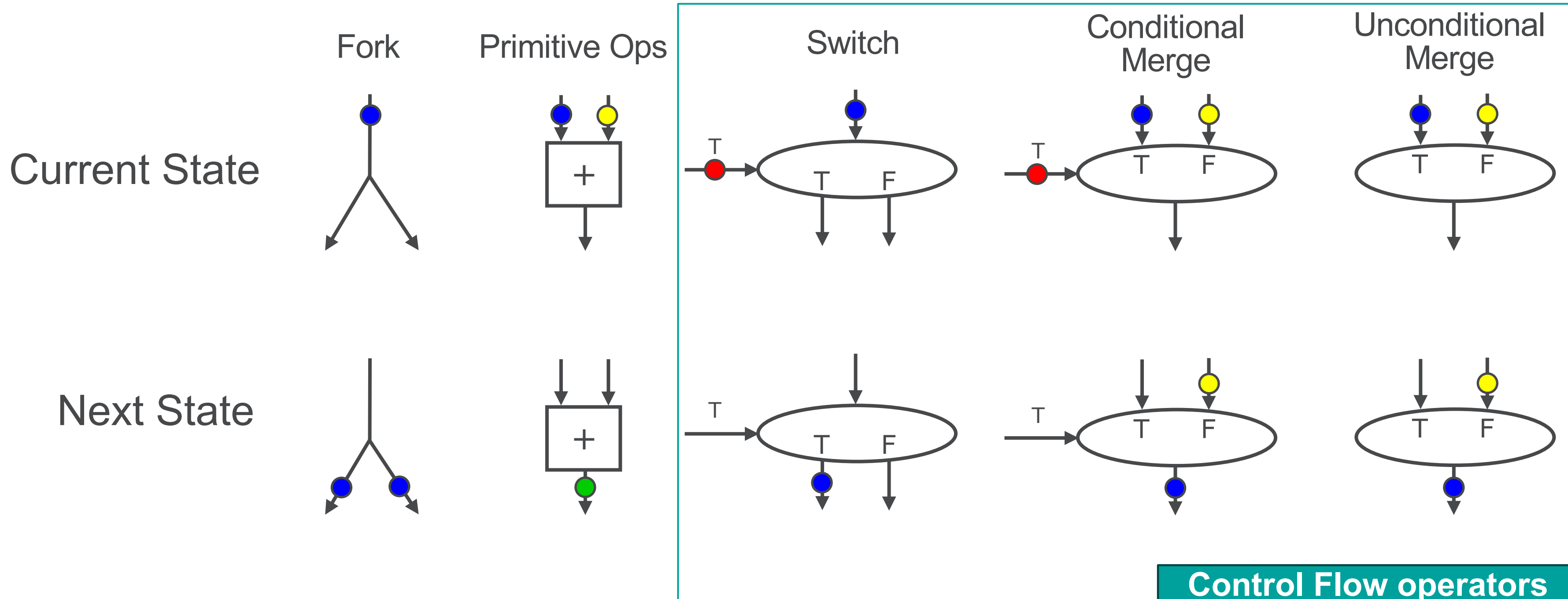
Snapshot

DATAFLOW OPERATORS



Dennis, J.B. (1974). First version of a data flow procedure language. In: Robinet, B. (eds) Programming Symposium. Lecture Notes in Computer Science, vol 19. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-06859-7_145

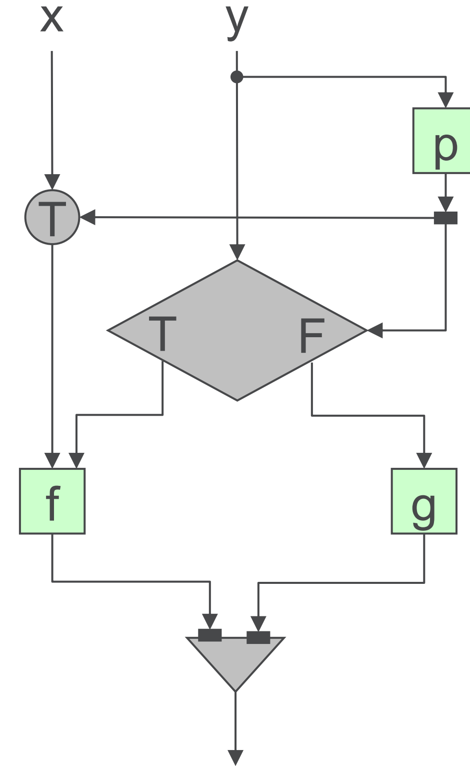
DATAFLOW OPERATORS



Dennis, J.B. (1974). First version of a data flow procedure language. In: Robinet, B. (eds) Programming Symposium. Lecture Notes in Computer Science, vol 19. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-06859-7_145

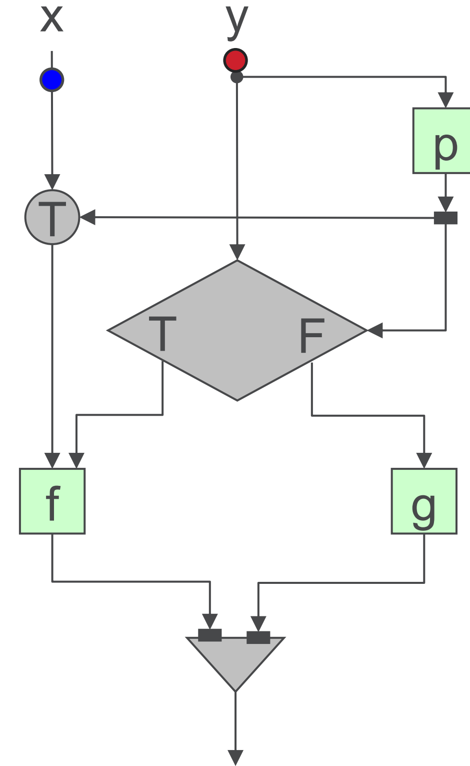
CONDITIONAL EXPRESSIONS

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



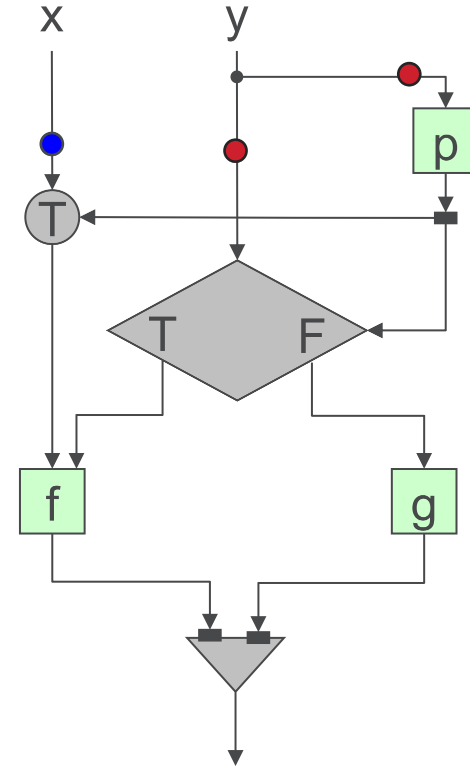
CONDITIONAL EXPRESSIONS

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



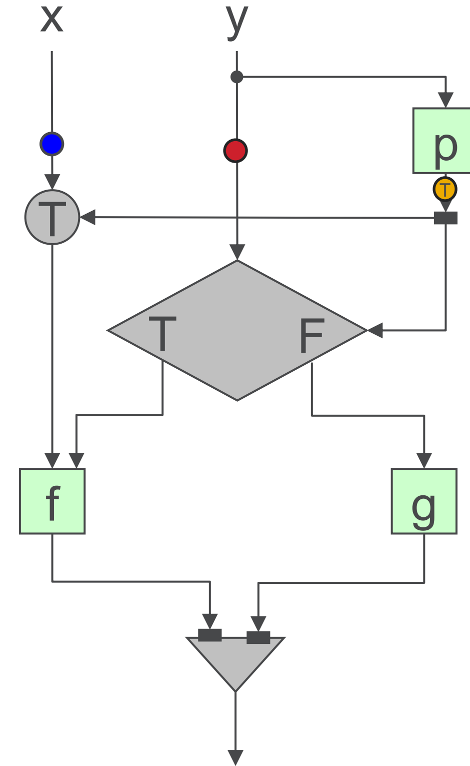
CONDITIONAL EXPRESSIONS

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



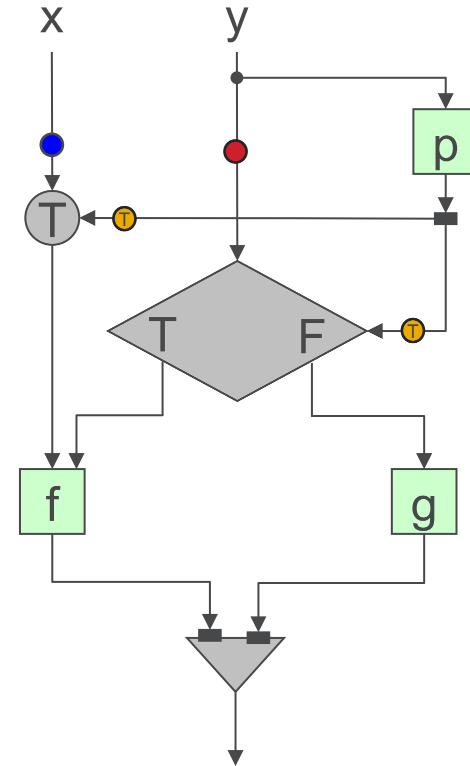
CONDITIONAL EXPRESSIONS

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



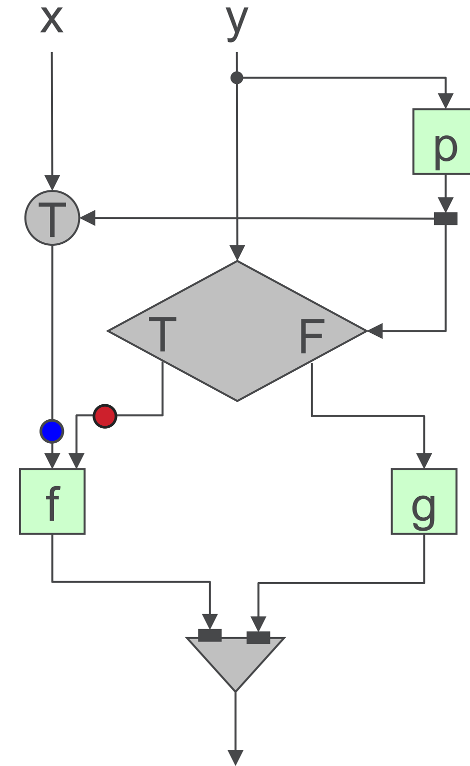
CONDITIONAL EXPRESSIONS

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



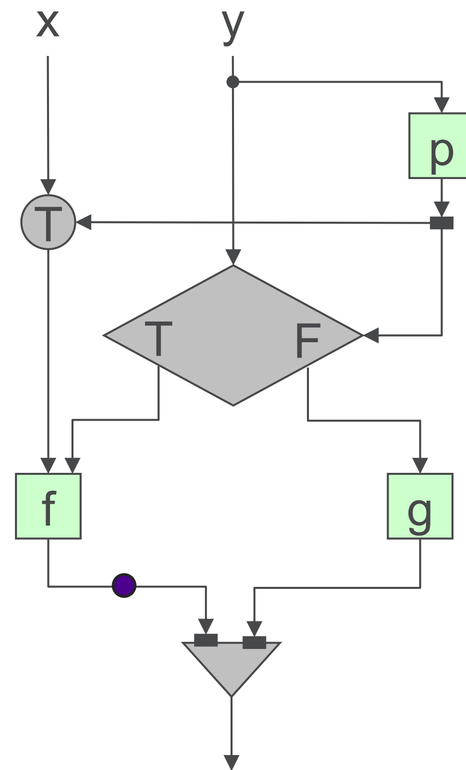
CONDITIONAL EXPRESSIONS

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



CONDITIONAL EXPRESSIONS

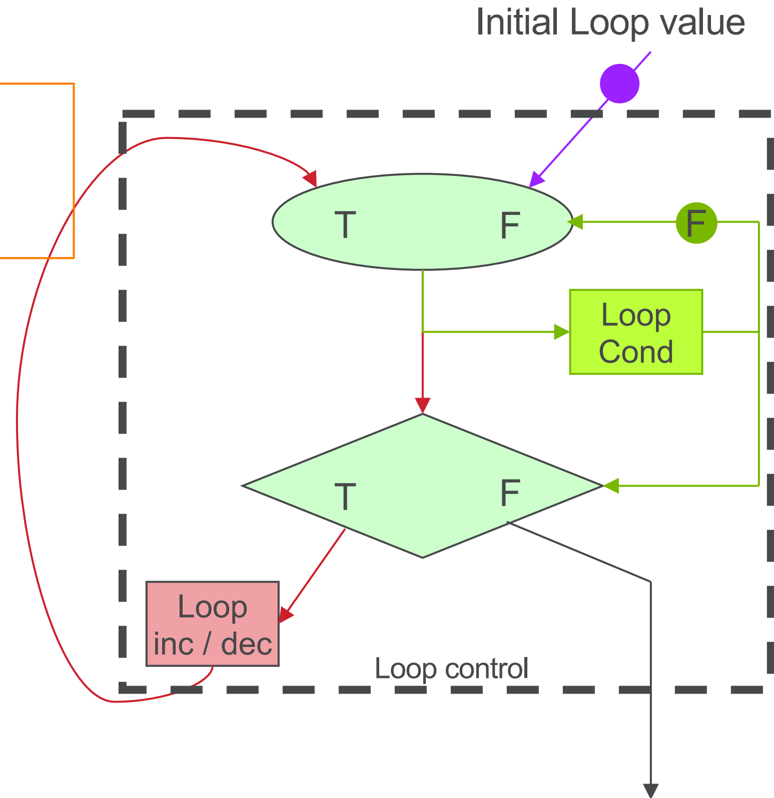
```
if( p(y) ) {
  f(x, y);
} else {
  g(y);
}
```



LOOP SCHEMA

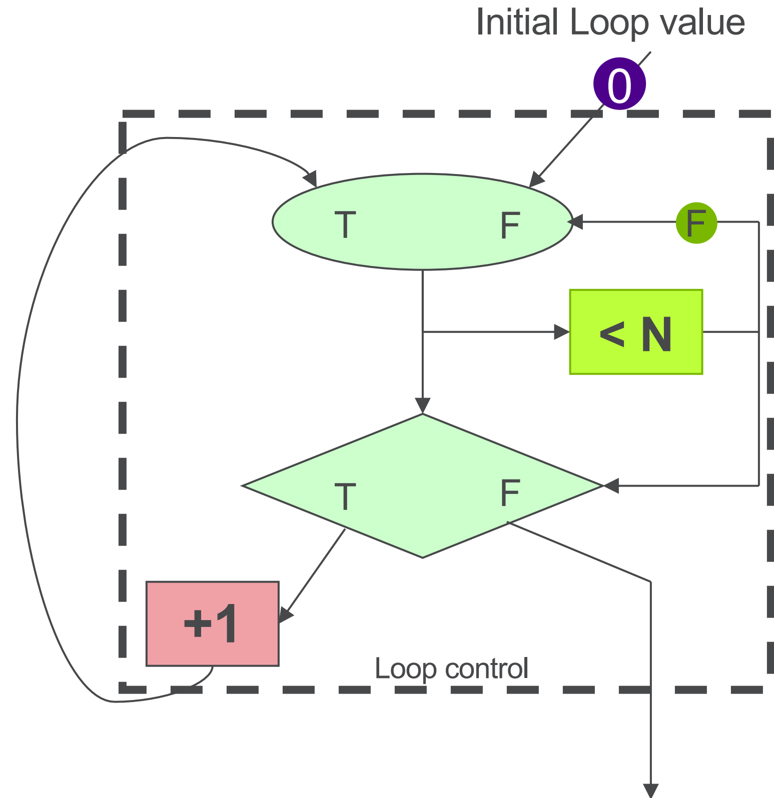
for (init; condition ; increment)
{ //…..// }

Loop control logic



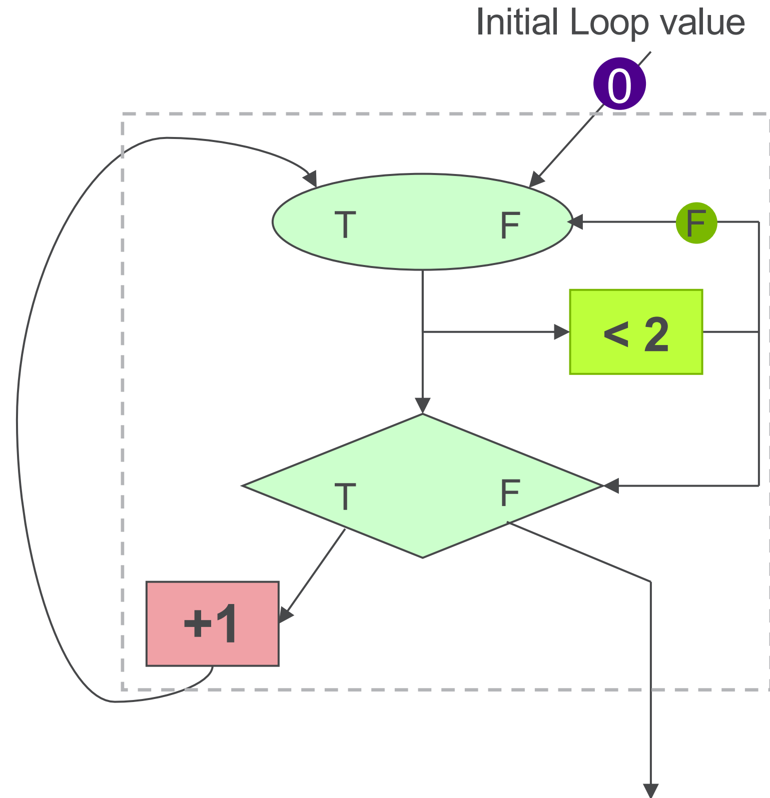
LOOP SCHEMA

```
for (int i = 0; i < N; i ++)
{ //....// }
```



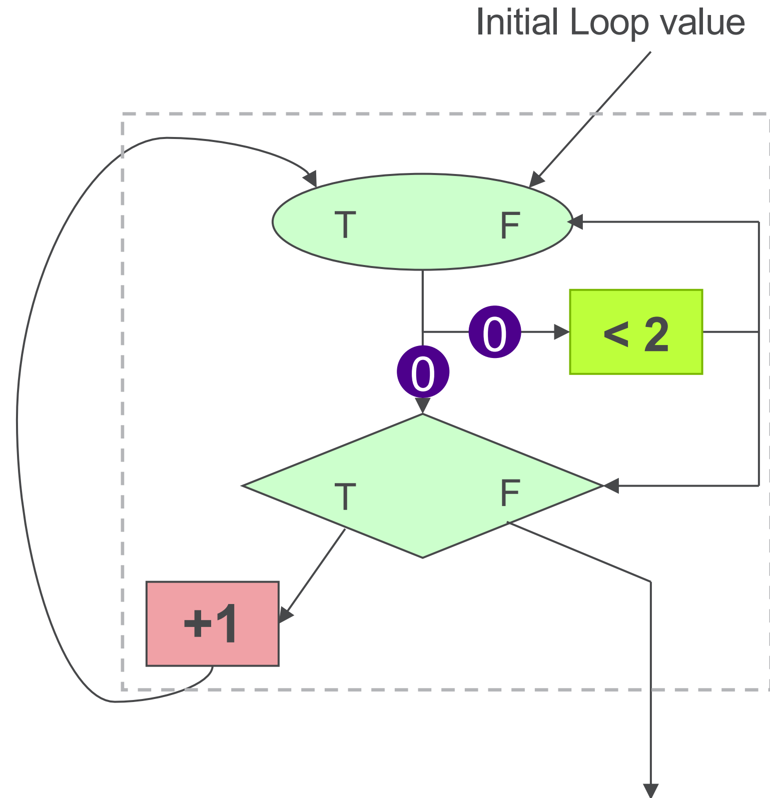
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



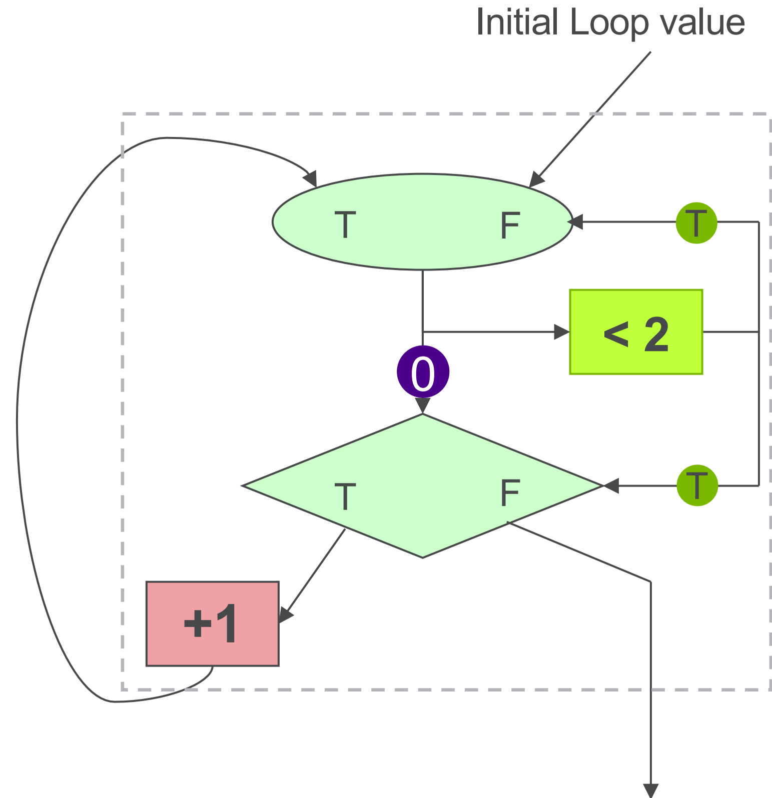
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



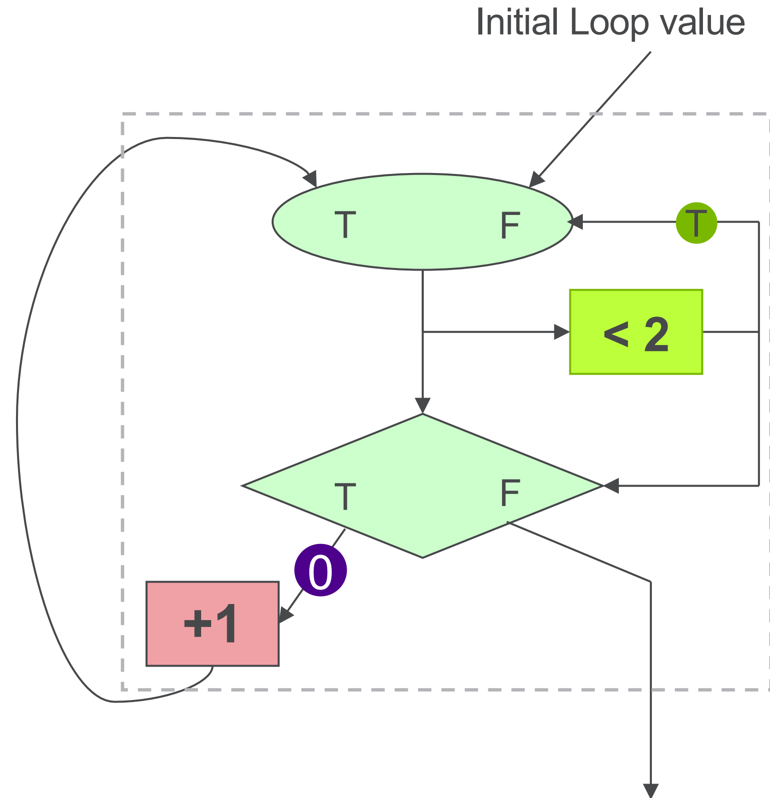
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



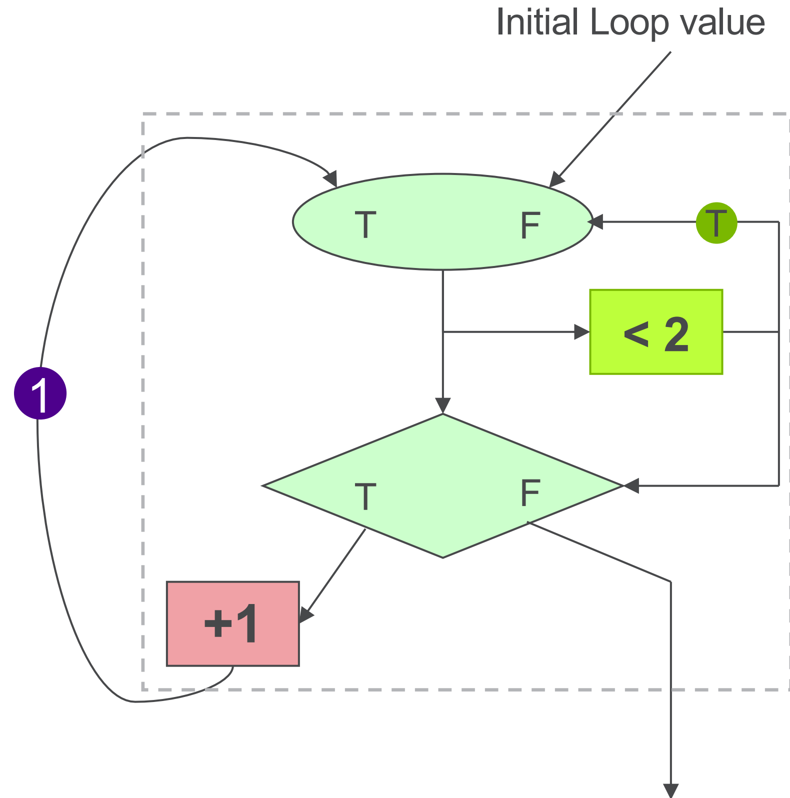
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



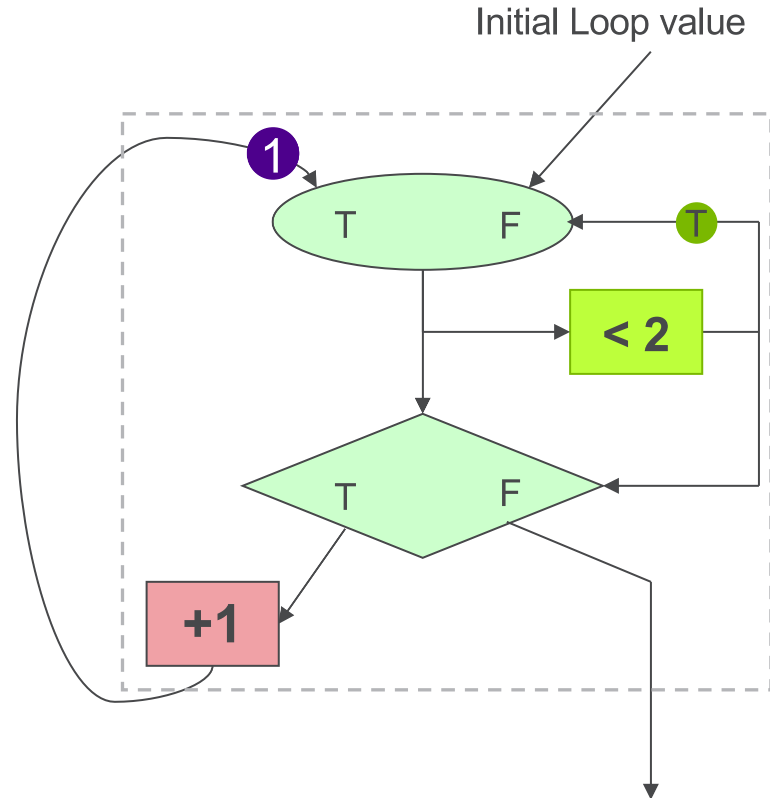
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



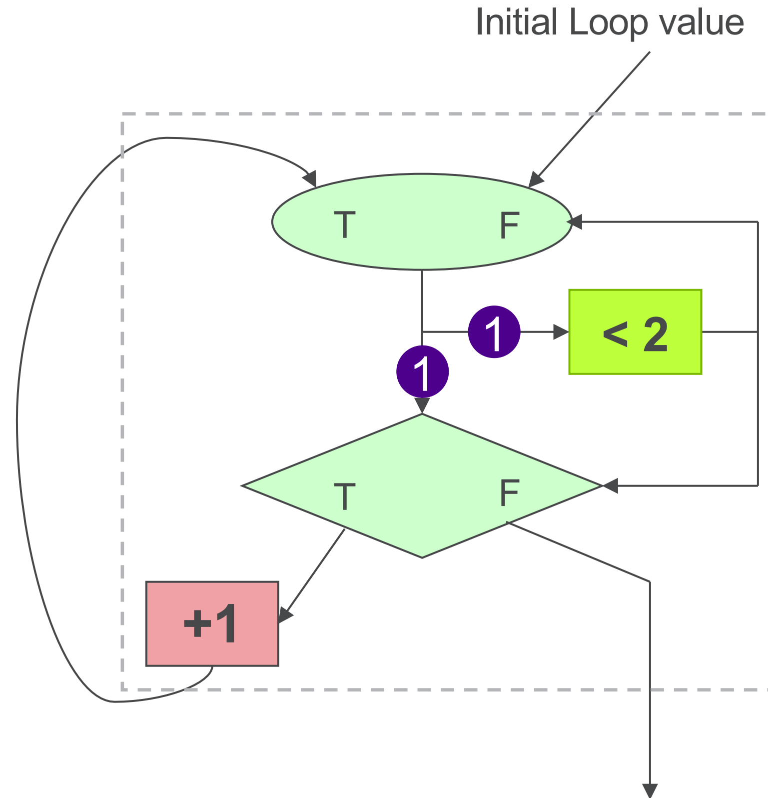
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //…..// }
```



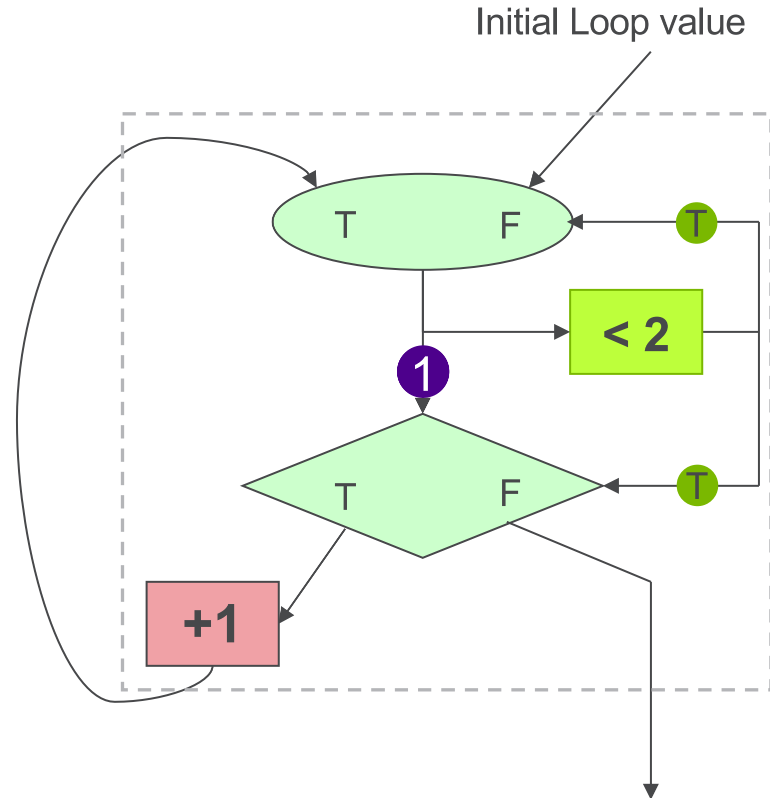
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



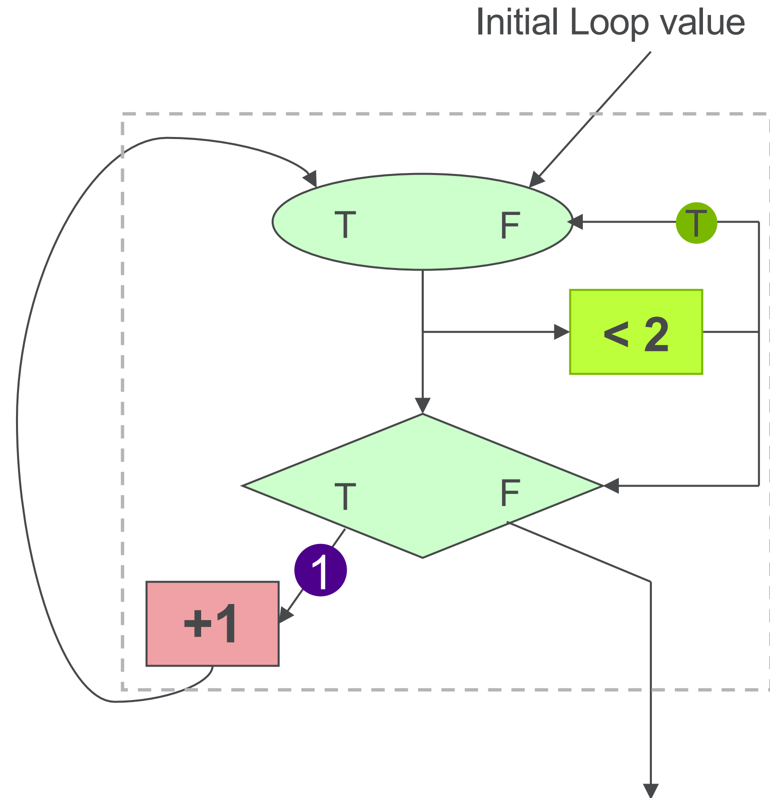
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



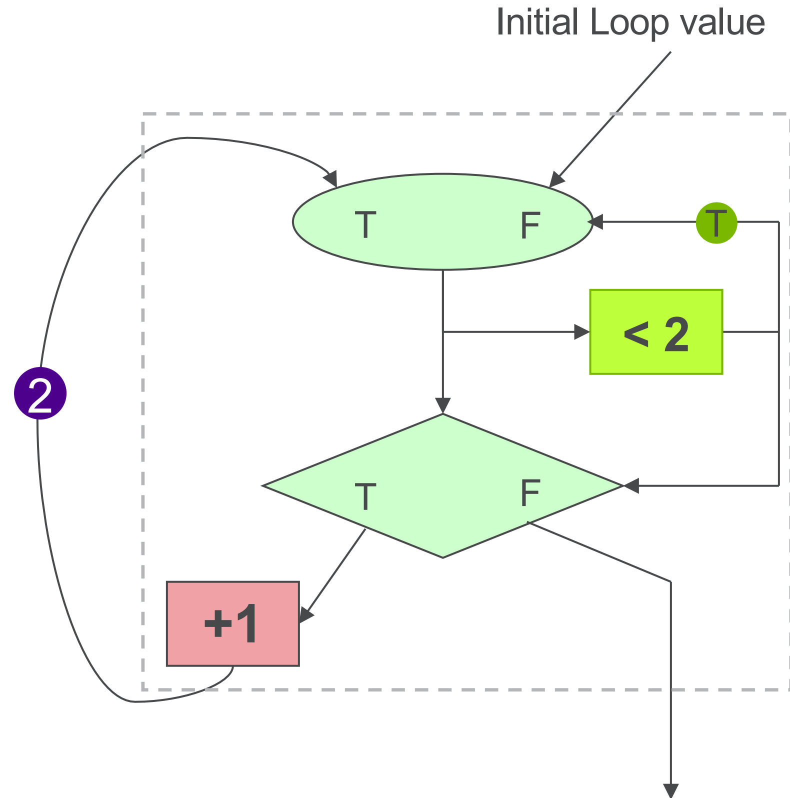
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



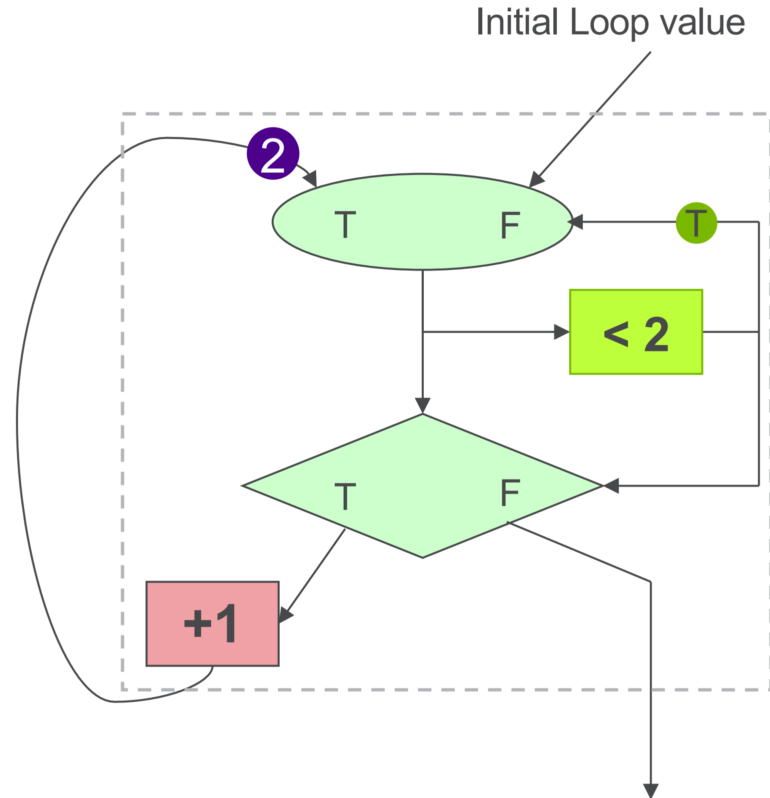
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



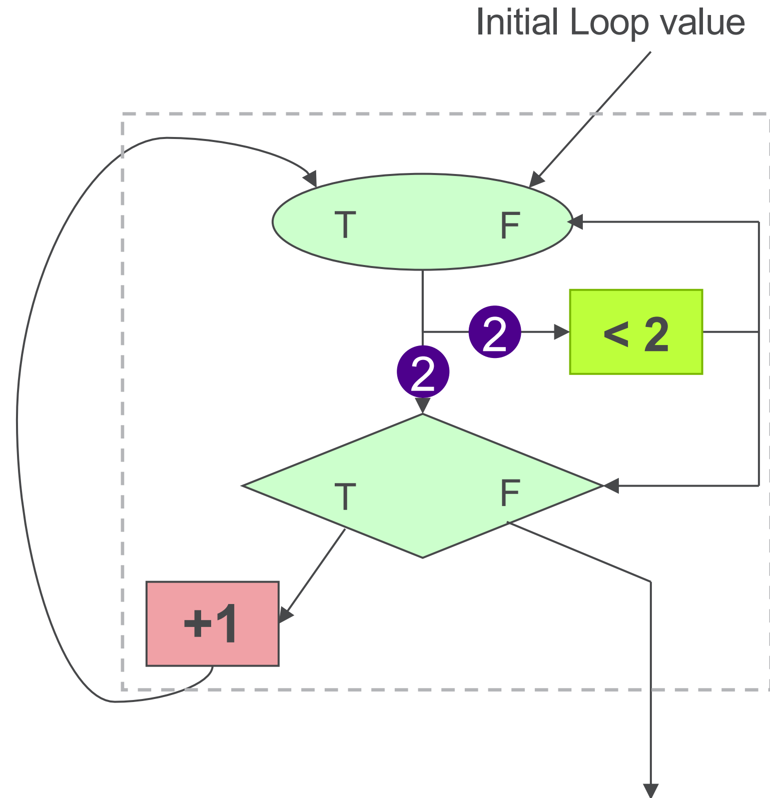
LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```



LOOP SCHEMA

```
for (int i = 0; i < 2; i ++)
{ //....// }
```

Initial Loop value

T F

F

< 2

2

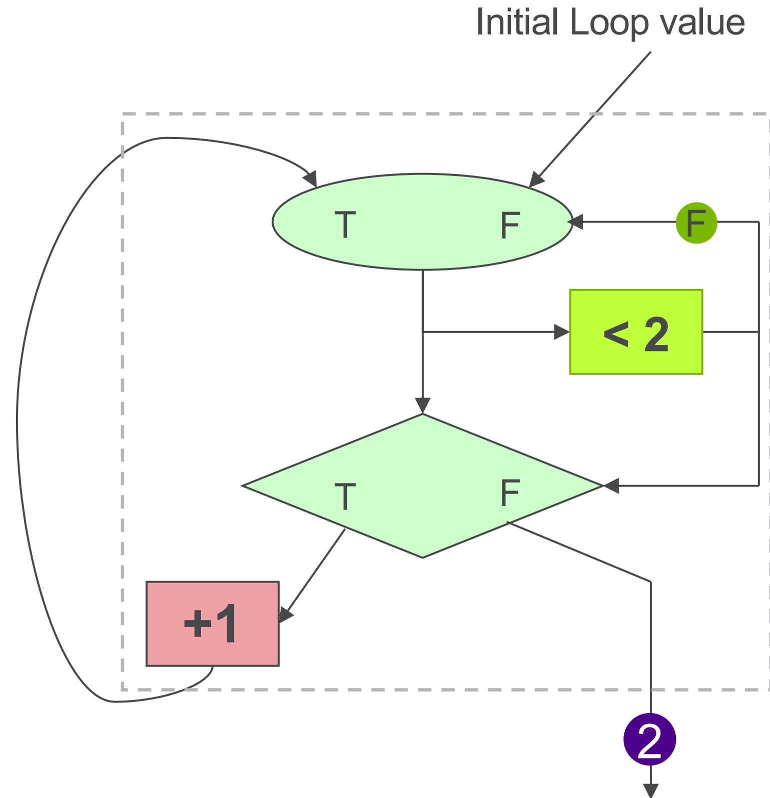
T F

F

+1

LOOP SCHEMA

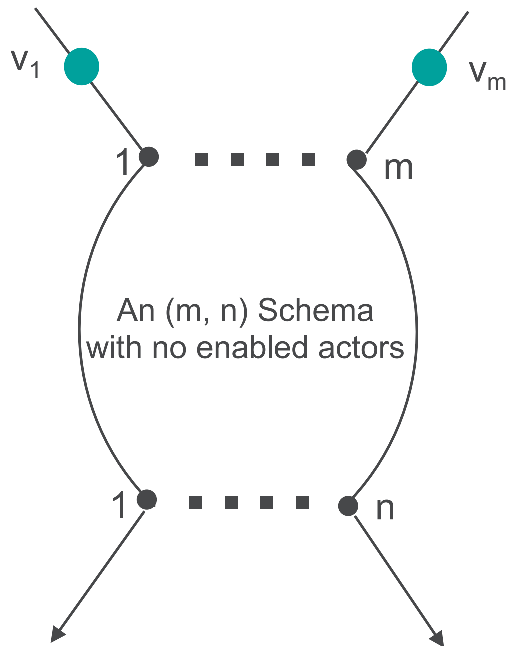
```
for (int i = 0; i < 2; i ++)
{ //....// }
```



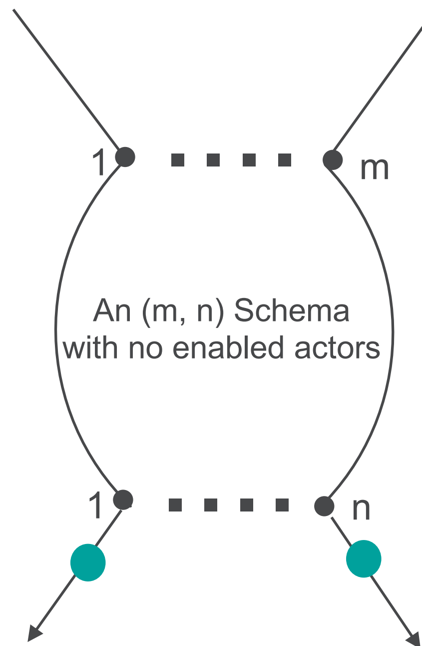
WELL BEHAVED DATAFLOW GRAPHS

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

WELL BEHAVED DATAFLOW GRAPHS



(a) Initial Snapshot



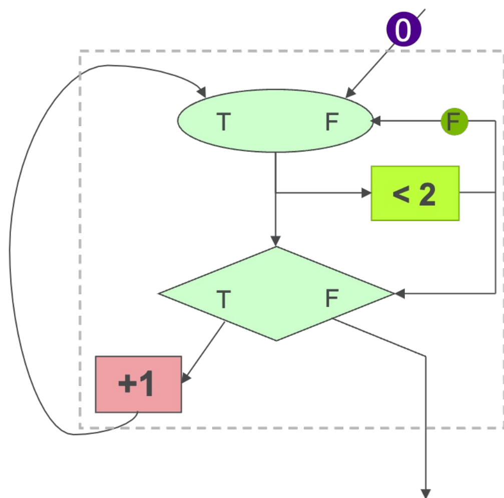
(a) Final Snapshot

WELL BEHAVED DATAFLOW GRAPHS

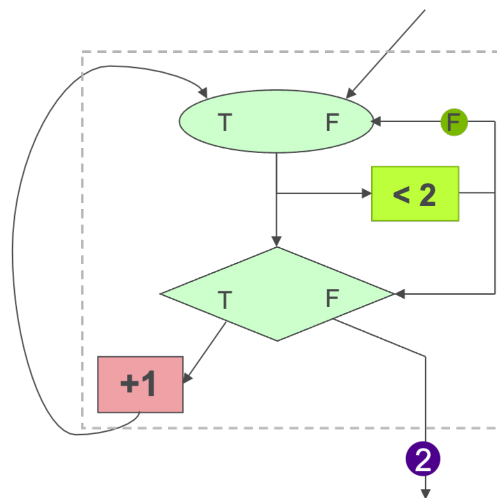
Implies Initial configuration is re-established

LOOP SCHEMA IS WELL BEHAVED

Initial Configuration

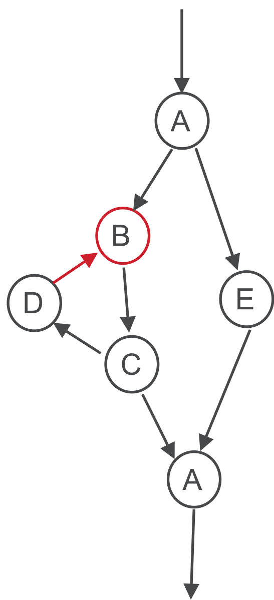


Final Configuration

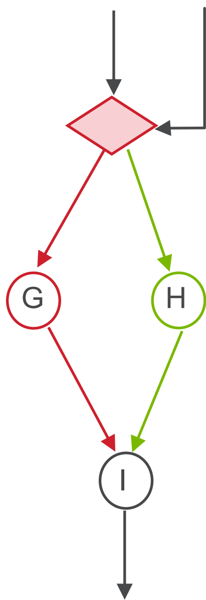


"SICK" DATAFLOW GRAPHS

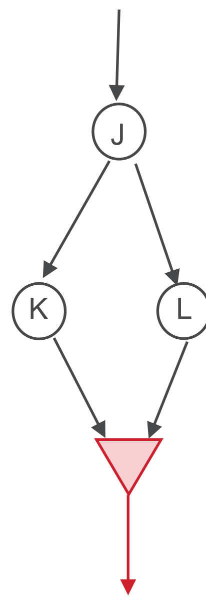
Arbitrary connections of data flow operators can result in pathological programs



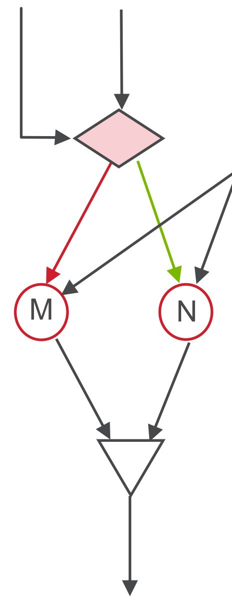
Deadlock



Hang up



Conflict



Unclean

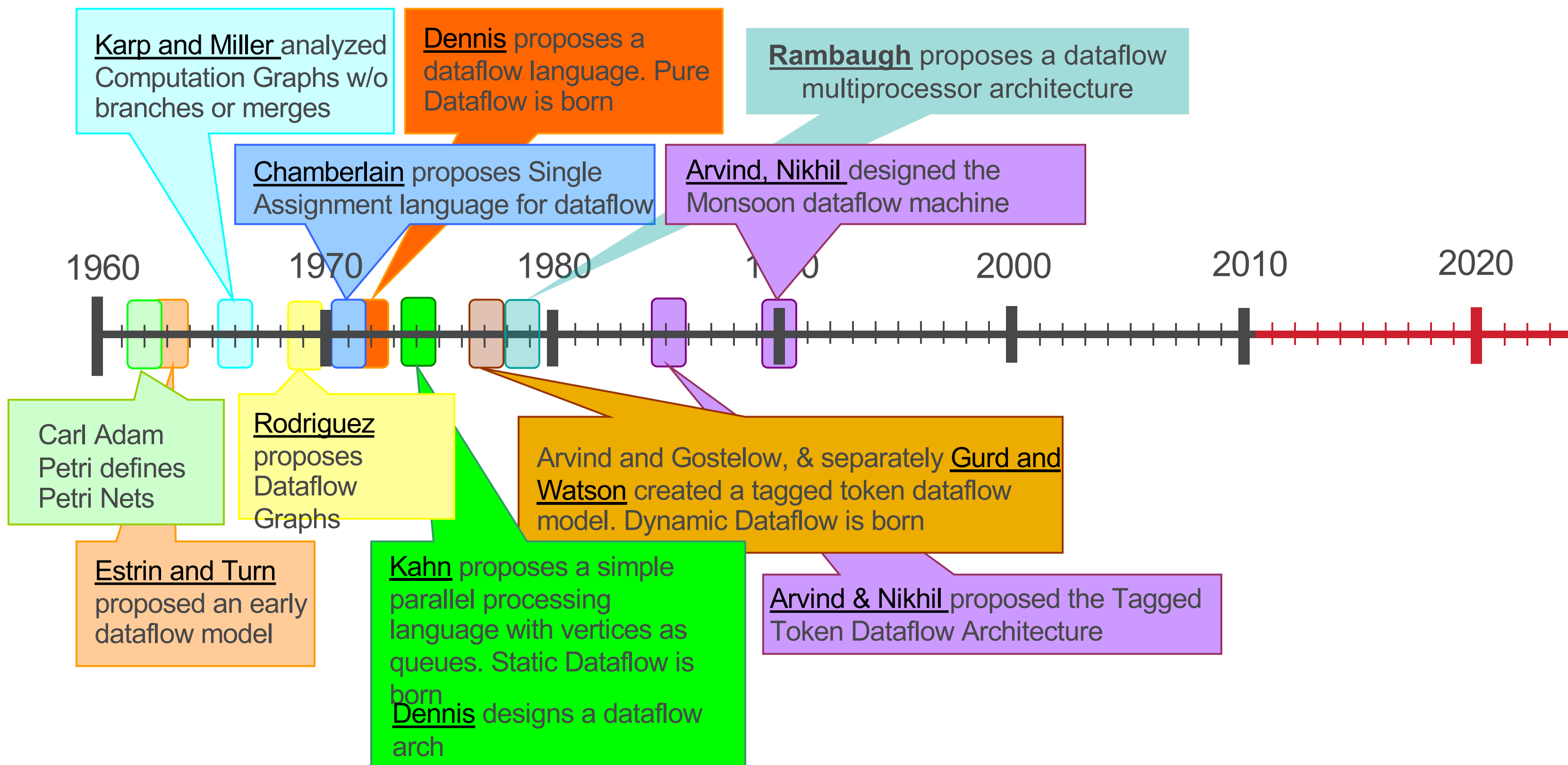
DATAFLOW MODEL OF COMPUTATION

- Define what a program is
- What are the operands
- What is well defined dataflow graphs?

EVOLUTION OF DATAFLOW ARCHITECTURES AND CONCEPTS

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

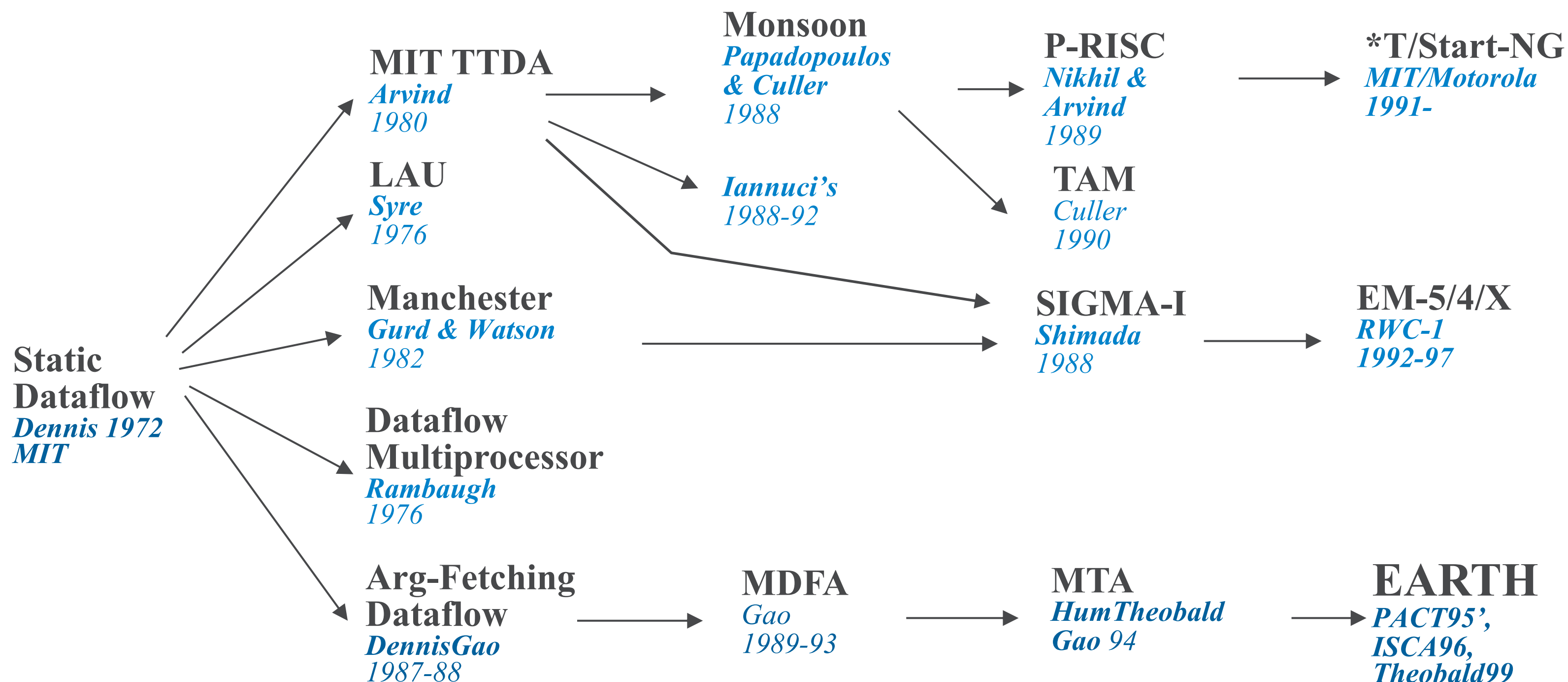
BRIEF HISTORY OF DATAFLOW



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

DATAFLOW ARCHITECTURES

Machines from the ”first spring” of dataflow



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

A DATAFLOW MULTIPROCESSOR
JAMES RUMBAUGH

A Data Flow Multiprocessor

JAMES RUMBAUGH

Abstract—This paper presents the architecture of a highly concurrent multiprocessor which runs programs expressed in *data flow* notation. Sequencing of data flow instruction execution depends only on the availability of operands required by instructions. Because data flow instructions have no side effects, unrelated instructions can be executed concurrently without interference if each has its required operands.

The data flow multiprocessor is hierarchically constructed as a network of simple modules. All module interactions are asynchronous. The principal working elements of the machine are a set of *activation processors*, each of which performs the execution of one invocation of a data flow procedure held in a local memory within the processor. A pipeline of logical units within each processor executes several concurrently active instructions. All data flow operations are performed within single processors except procedure calls, which cause the creation of new activations in other processors, and operations on large data structures, which are performed by *structure controller* modules using values stored in a central memory. Concurrency within a data flow procedure provides a processor with something to do while a slow operation is being processed.

The behavior of the machine has been specified by a formal description language and has been shown to correctly implement the data flow language. The principal advantages of the data flow multiprocessor over conventional designs are reduced complexity of the processor-memory connection, greater use of pipelining, and a simpler representation and implementation of concurrent activity.

Index Terms—Asynchronous logic, cache, concurrency, data-driven instruction execution, data flow program, modularity, multiprocessor, parallel processor, pipelining.

Manuscript received September 19, 1975; revised June 17, 1976. This work was performed in part while on assignment from the General Electric Company. The work was also supported in part by IBM, and in part by the Advanced Research Projects Agency under Contract N00014-70-A-0362-0006. This paper is based on a dissertation submitted to the Massachusetts Institute of Technology, Cambridge, MA, in partial fulfillment of the requirements for the Ph.D. degree.

The author was with Project MAC, Massachusetts Institute of Technology, Cambridge, MA. He is now with General Electric Corporate Research and Development, Schenectady, NY.

I. Introduction

THE DESIGN of a highly concurrent computer is complicated by potential interdependencies within programs and between modules of the machine. This paper presents the architecture of a multiprocessor and an associated program notation which reduce such interdependencies. Because interactions are restricted, modules

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

RAMBAUGH'S ARCHITECTURE ACTIVATION PROCESSOR

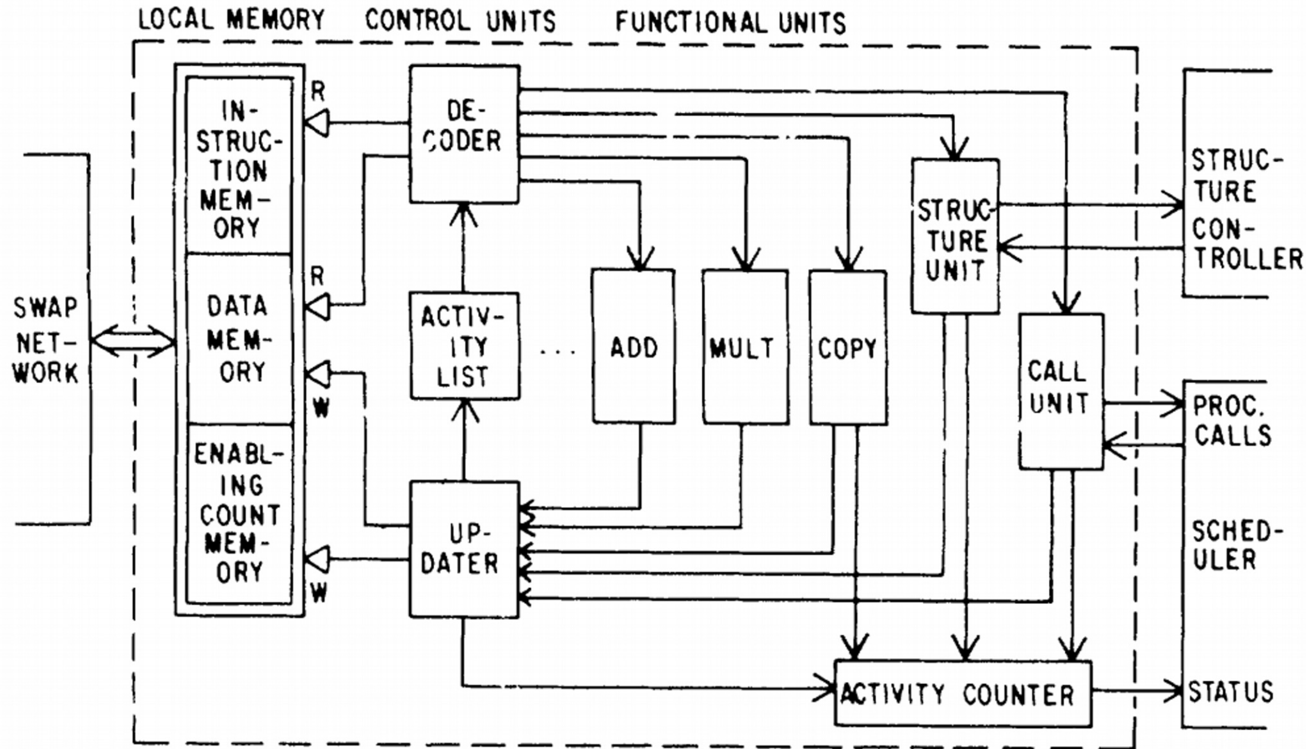


Fig. 14. Activation processor.

RAMBAUGH'S ARCHITECTURE LOCAL MEMORY

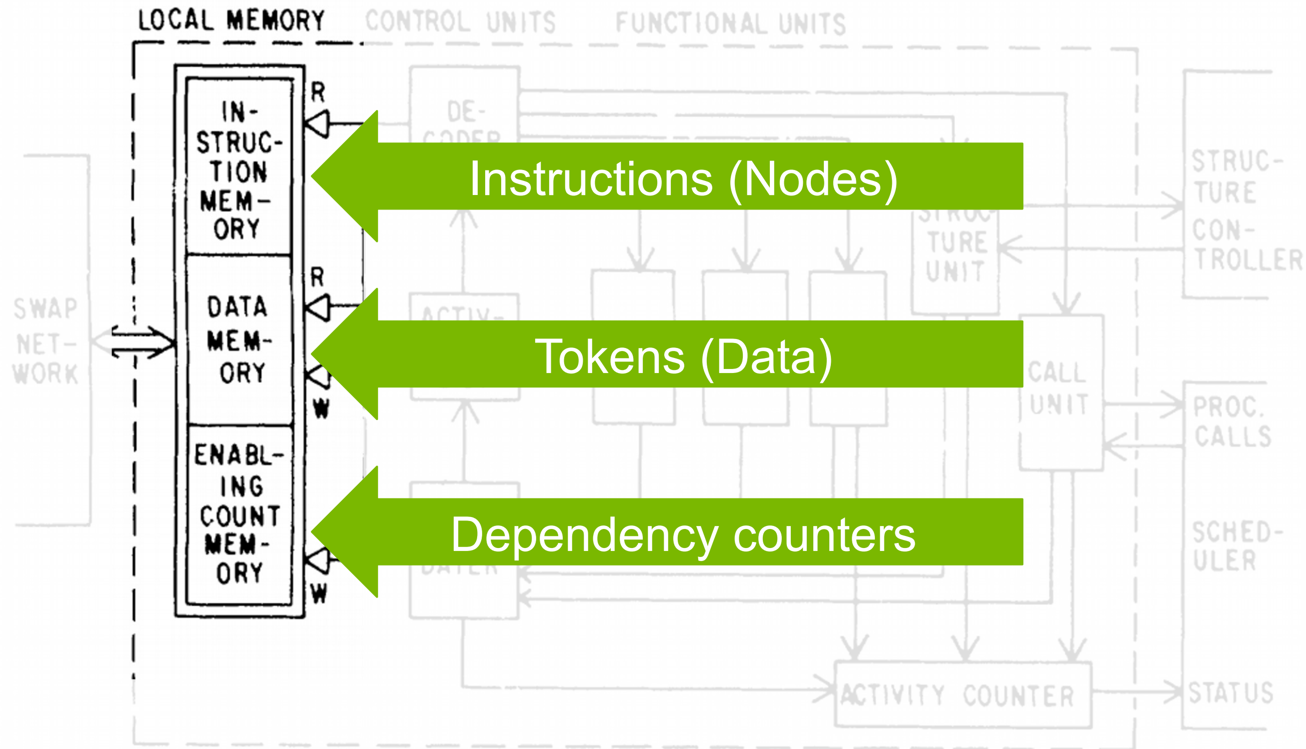


Fig. 14. Activation processor.

RAMBAUGH'S ARCHITECTURE SCHEDULER

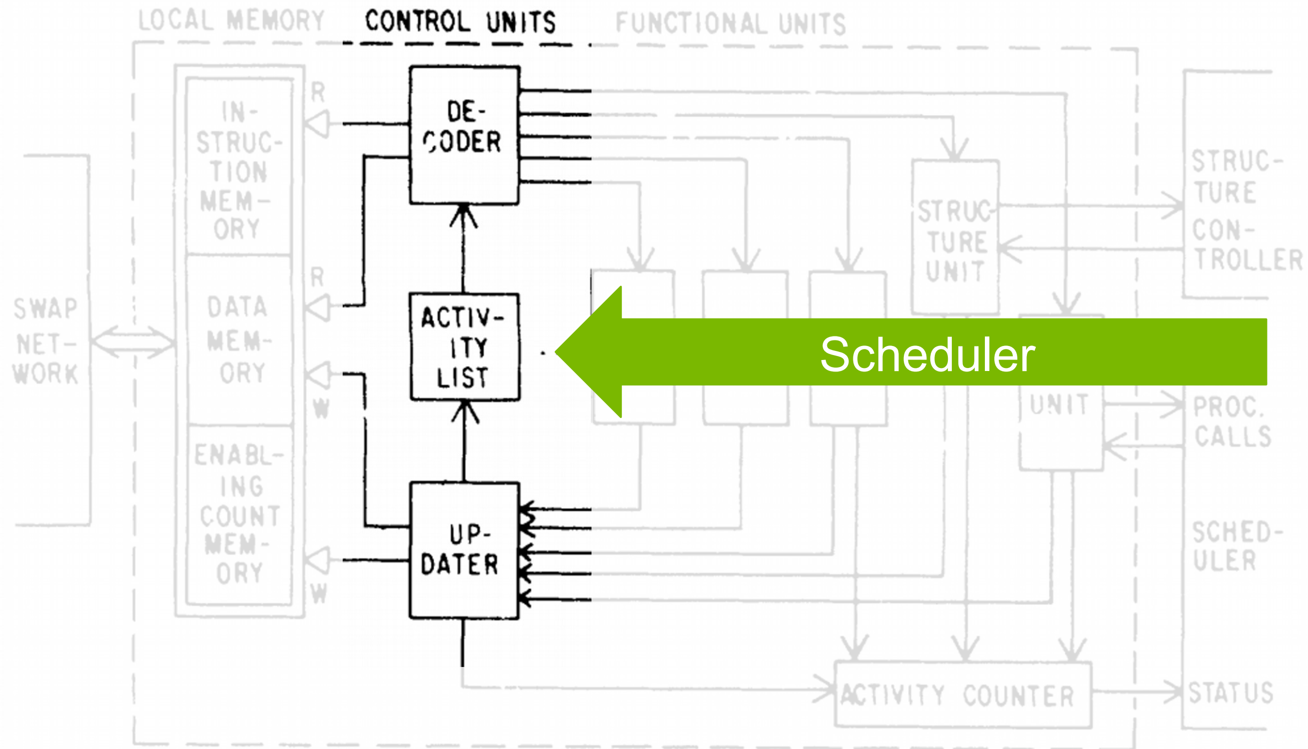


Fig. 14. Activation processor.

RAMBAUGH'S ARCHITECTURE SCHEDULER

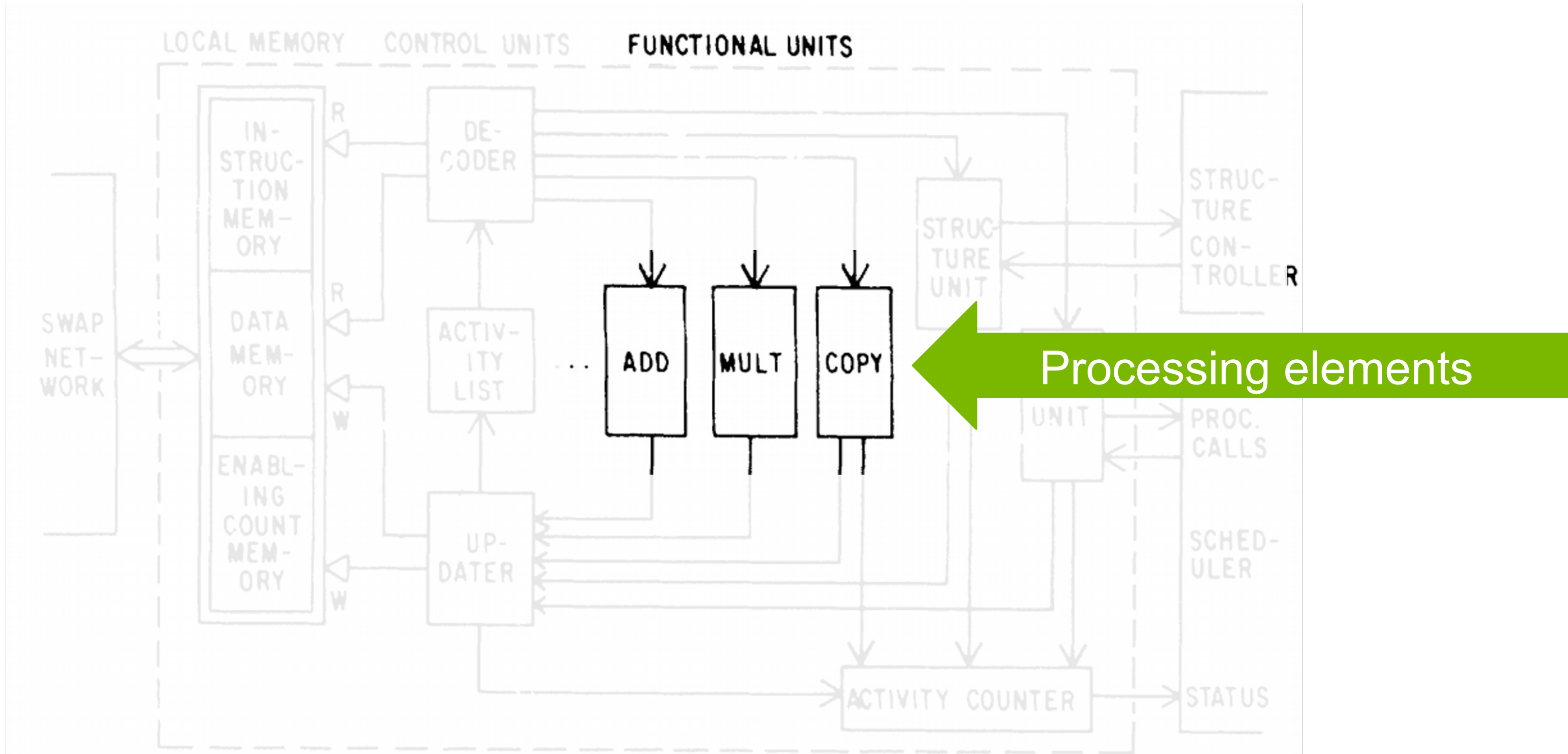


Fig. 14. Activation processor.

RAMBAUGH'S ARCHITECTURE ACTIVATION PROCESSOR

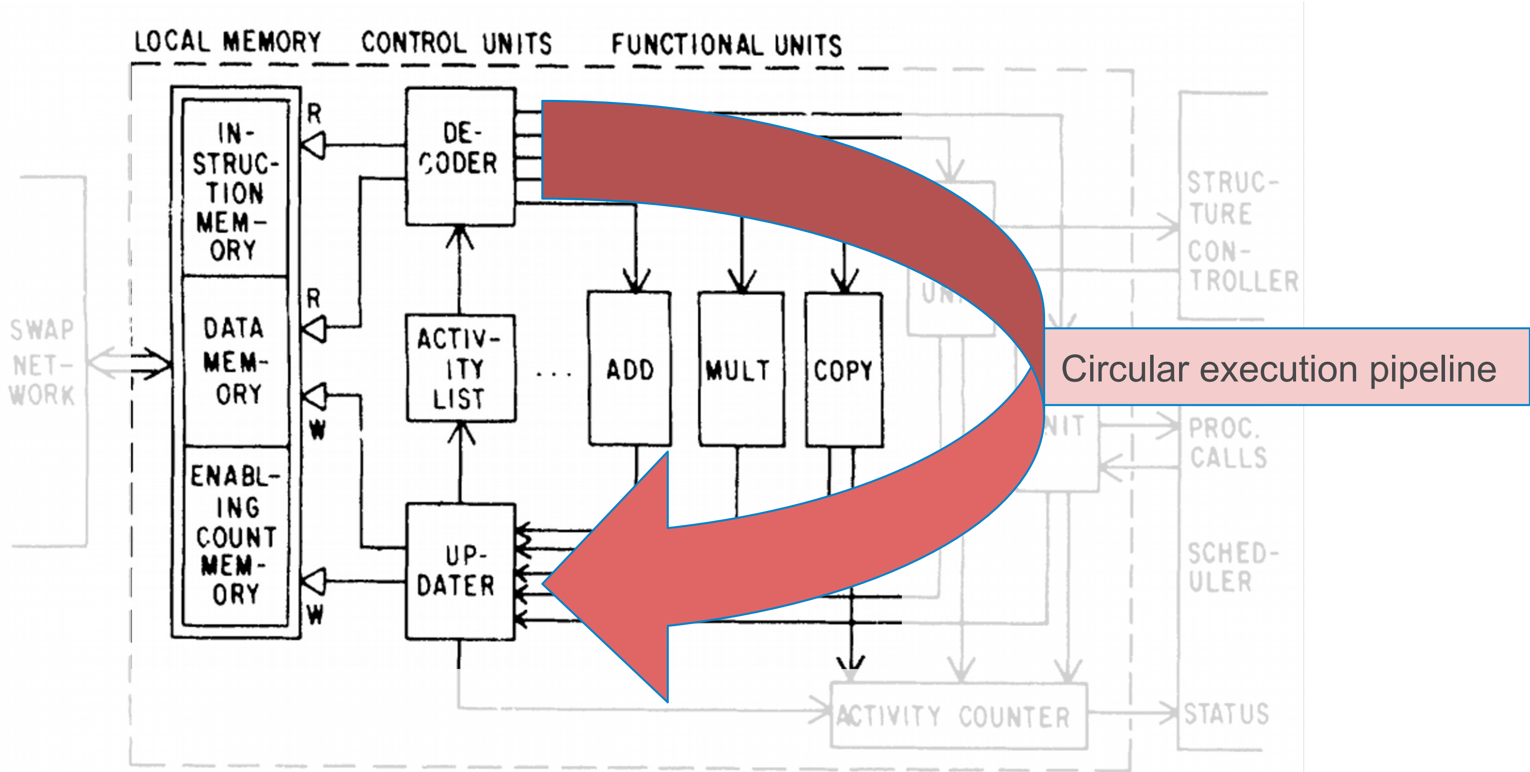
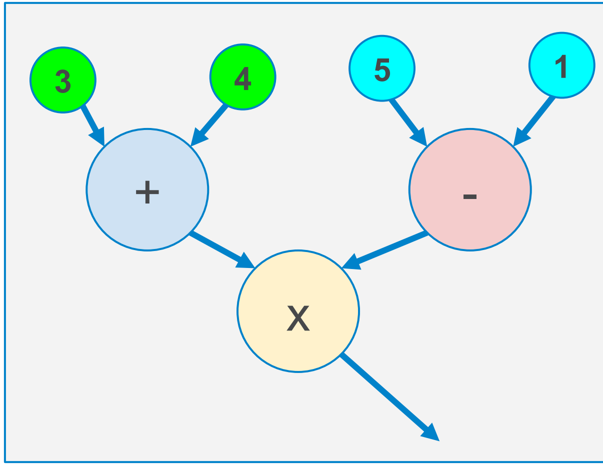


Fig. 14. Activation processor.

RAMBAUGH'S ARCHITECTURE LOCAL MEMORY

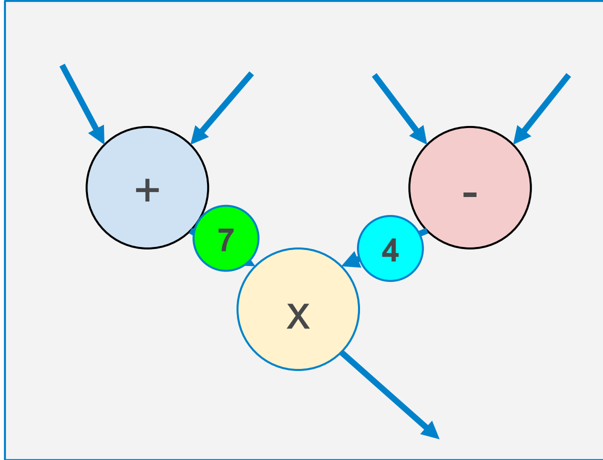


Enabling Count Memory	
Address	**Count**
0x20	**0**
0x24	**0**
0x28	2

Data Memory		
Address	**Type**	**Value**
0x44	int	**3**
0x48	int	**4**
0x4C	int	xxx
0x50	int	**5**
0x54	int	**1**
0x58	int	xxx
0x5C	double	xxx

Instruction Memory					
Address	**Opcode**	**Op1 Addr**	**Op2 Addr**	**Res Addr**	**Succ Addr**
0x20	+	0x44	0x48	0x4C	0x28
0x24	-	0x50	0x54	0x58	0x28
0x28	x	0x4C	0x58	0x5C	...

RAMBAUGH'S ARCHITECTURE LOCAL MEMORY

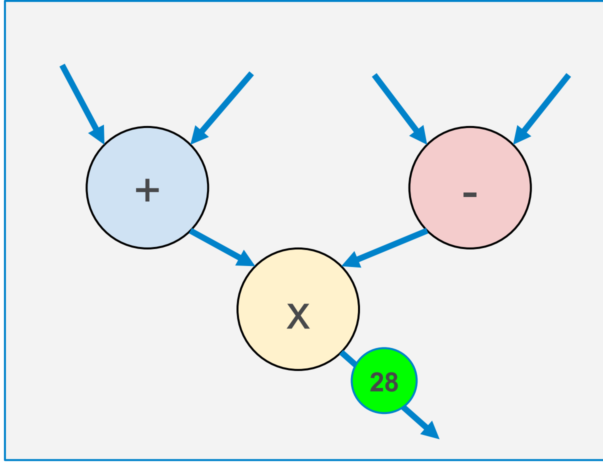


Enabling Count Memory	
Address	**Count**
0x20	2
0x24	2
0x28	**0**

Data Memory		
Address	**Type**	**Value**
0x44	int	3
0x48	int	4
0x4C	int	**7**
0x50	int	5
0x54	int	1
0x58	int	**4**
0x5C	double	xxx

Instruction Memory					
Address	**Opcode**	**Op1 Addr**	**Op2 Addr**	**Res Addr**	**Succ Addr**
0x20	+	0x44	0x48	0x4C	0x28
0x24	-	0x50	0x54	0x58	0x28
0x28	x	0x4C	0x58	0x5C	...

RAMBAUGH'S ARCHITECTURE LOCAL MEMORY



Enabling Count Memory	
Address	**Count**
0x20	2
0x24	2
0x28	2

Data Memory		
Address	**Type**	**Value**
0x44	int	3
0x48	int	4
0x4C	int	7
0x50	int	5
0x54	int	1
0x58	int	4
0x5C	**double**	**28**

Instruction Memory					
Address	**Opcode**	**Op1 Addr**	**Op2 Addr**	**Res Addr**	**Succ Addr**
0x20	+	0x44	0x48	0x4C	0x28
0x24	-	0x50	0x54	0x58	0x28
0x28	x	0x4C	0x58	0x5C	...

DATAFLOW ARCHITECTURES

- Execute operational semantics of dataflow in hardware
- Represent tokens and their matching to instructions
- Memory is split into name/value pairs explicitly associated with the instructions

DATAFLOW PERFORMANCE

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

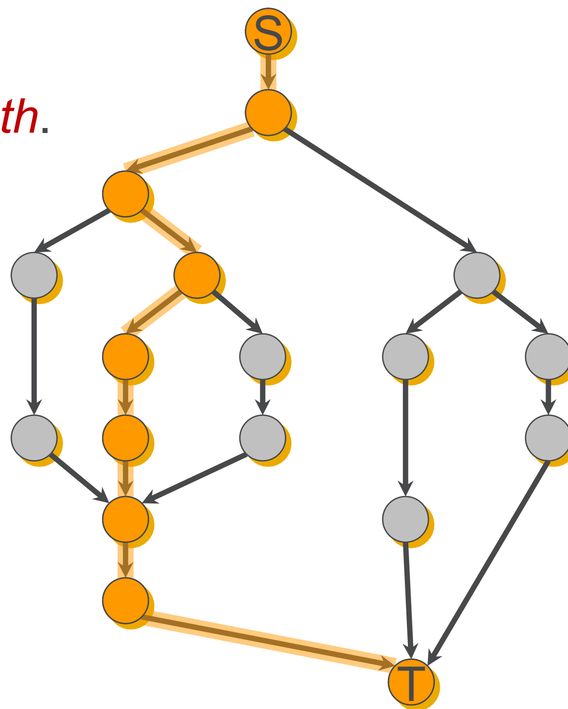
Argonne
NATIONAL LABORATORY

DATAFLOW PERFORMANCE

Critical path

longest path from S to T is known as the *critical path*.

Dataflow Minimum execution time is determined by the critical path and scheduling of ready tokens.

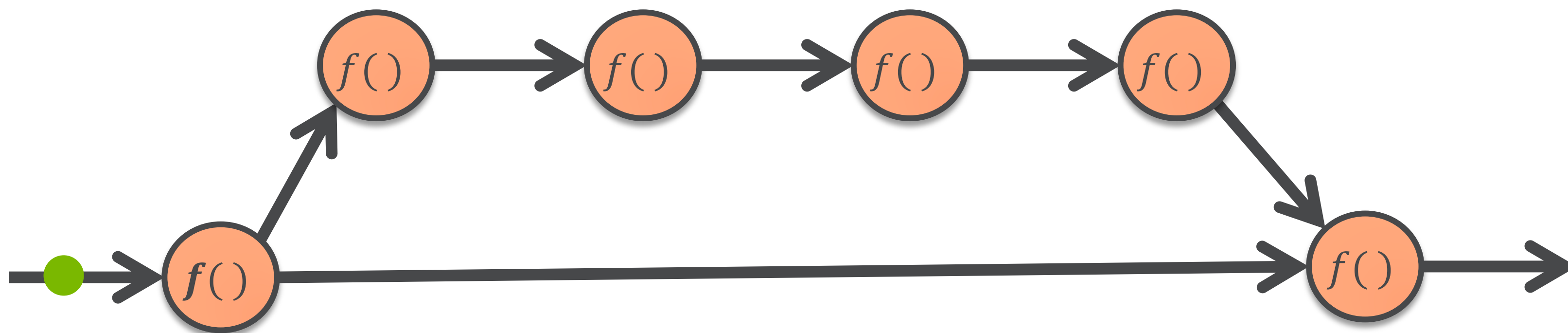


Guang R. Gao, Algorithmic aspects of balancing techniques for pipelined data flow code generation, Journal of Parallel and Distributed Computing, Volume 6, Issue 1, 1989, Pages 39-61, ISSN 0743-7315, https://doi.org/10.1016/0743-7315(89)90041-5.

DATAFLOW PERFORMANCE

Critical Path

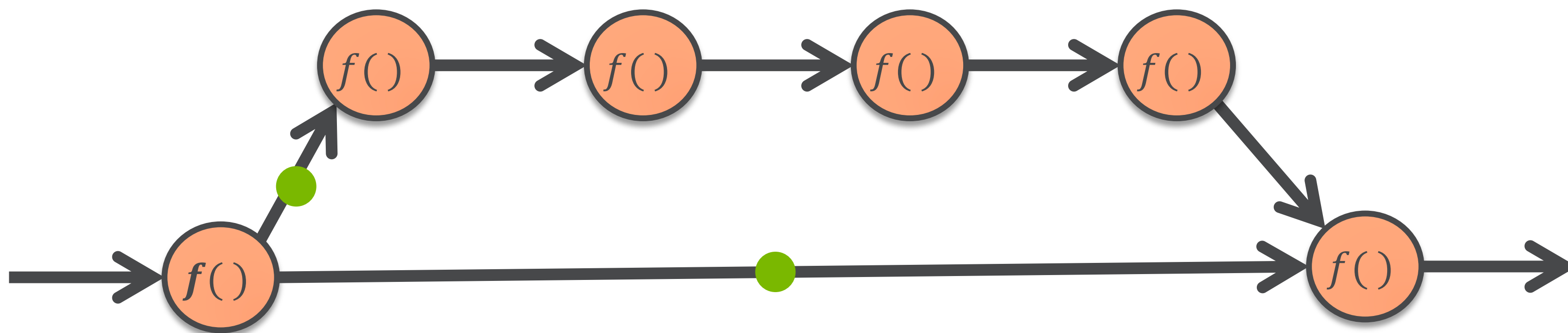
T = 0



DATAFLOW PERFORMANCE

Critical Path

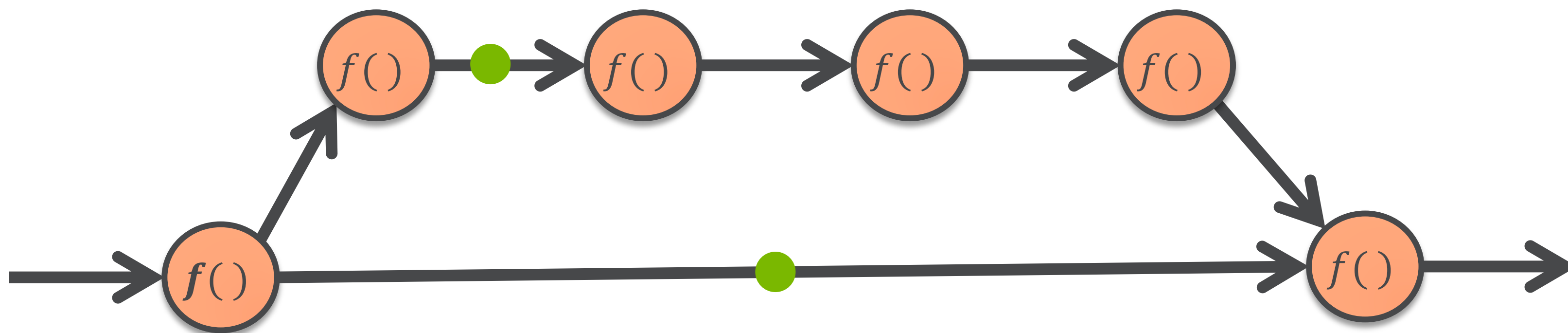
T = 1



DATAFLOW PERFORMANCE

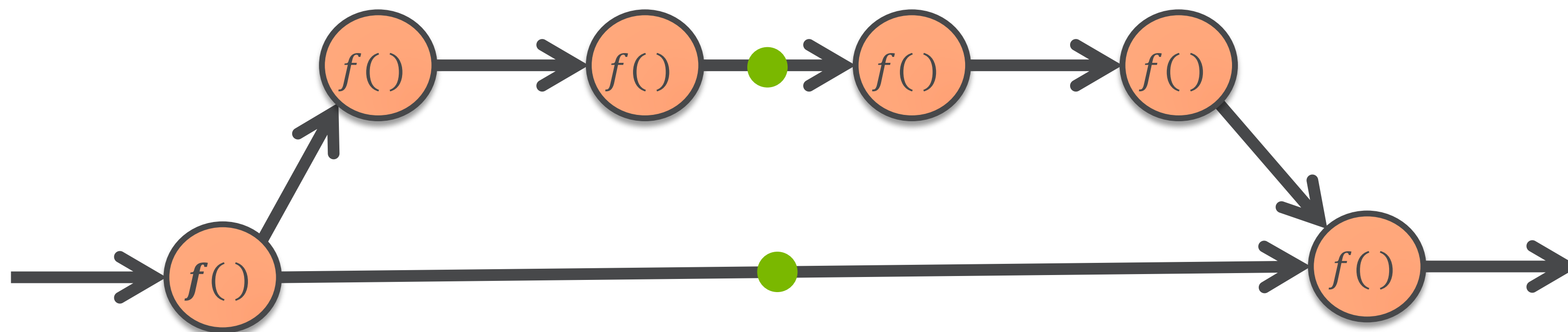
Critical Path

T = 2



DATAFLOW PERFORMANCE

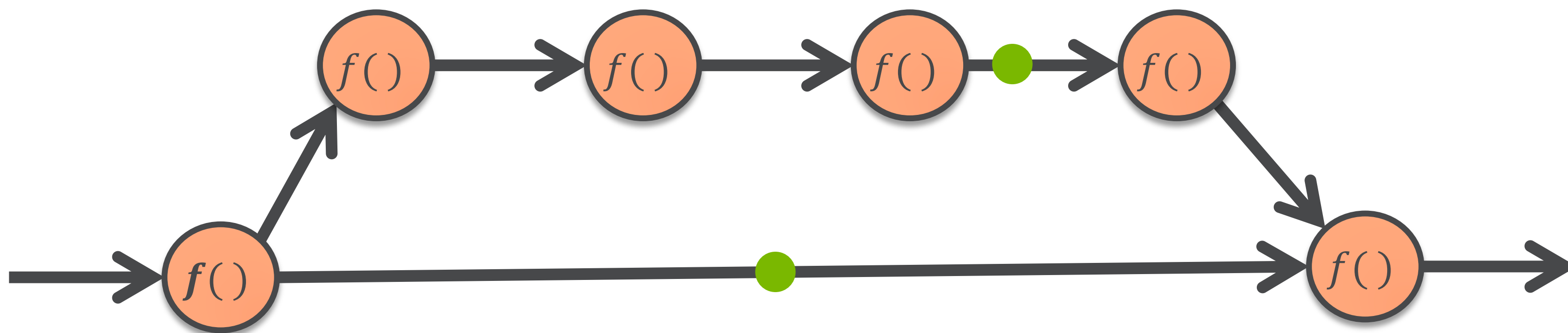
Critical Path



DATAFLOW PERFORMANCE

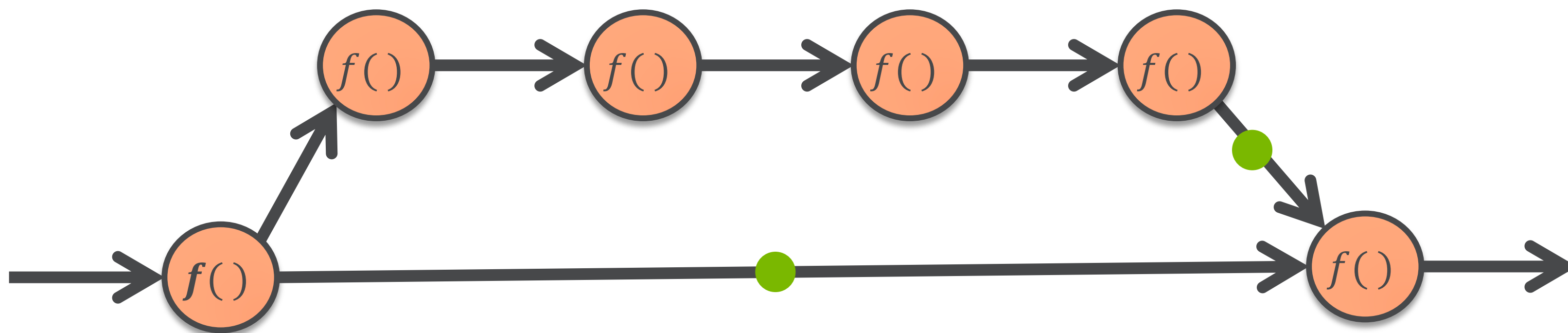
Critical Path

T = 4



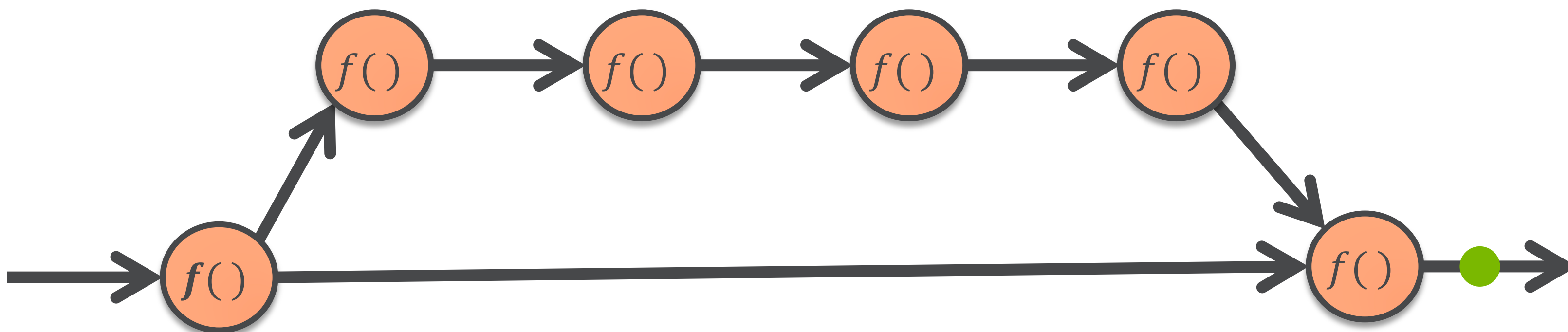
DATAFLOW PERFORMANCE

Critical Path



DATAFLOW PERFORMANCE

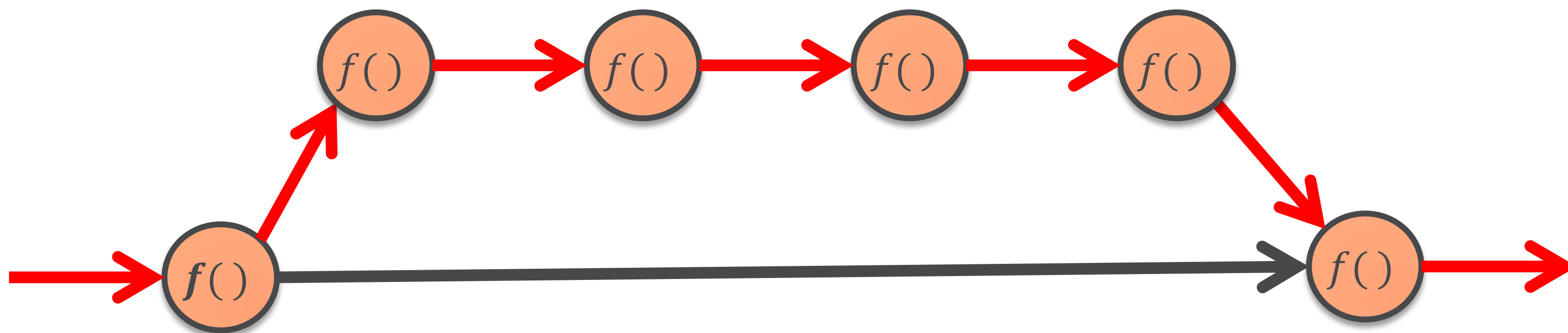
Critical Path



DATAFLOW PERFORMANCE

Critical Path

T = 6

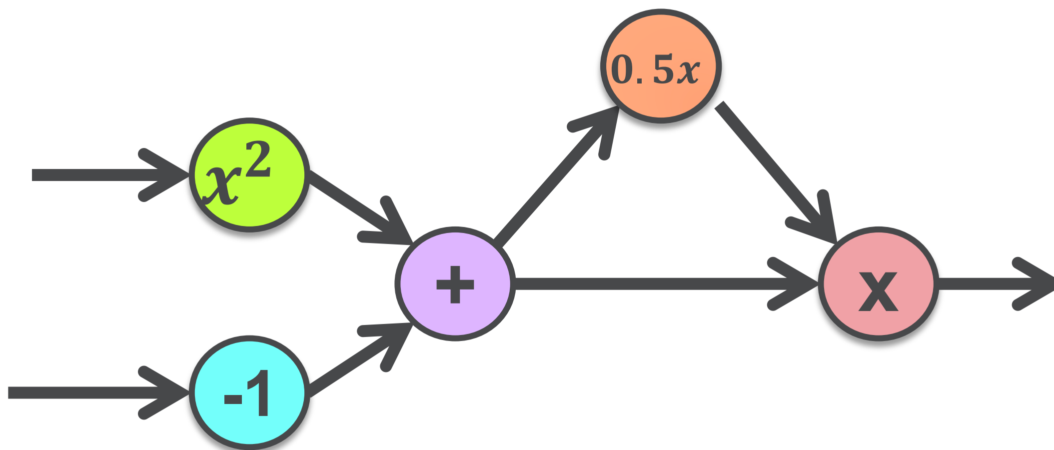


Execution time determined by the critical path

DATAFLOW PIPELINING

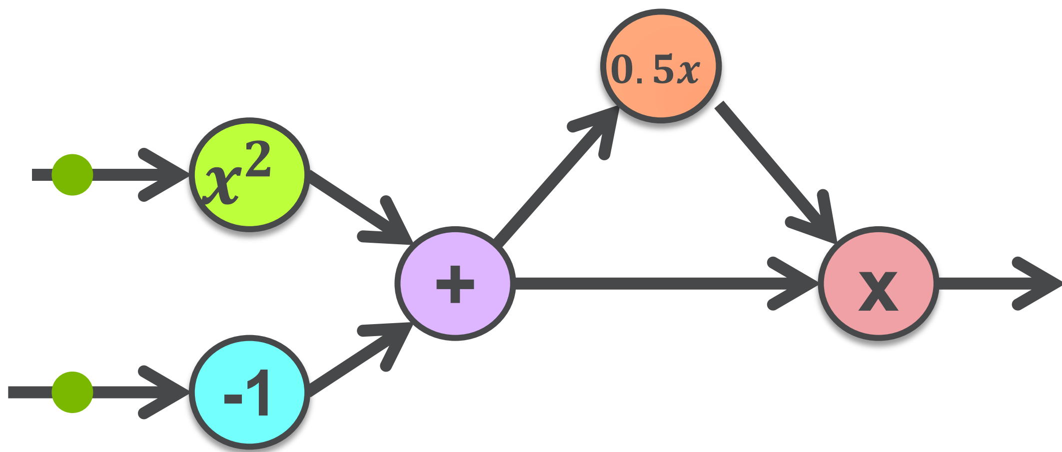
DATAFLOW PIPELINING

Allow multiple executions of the same dataflow graph



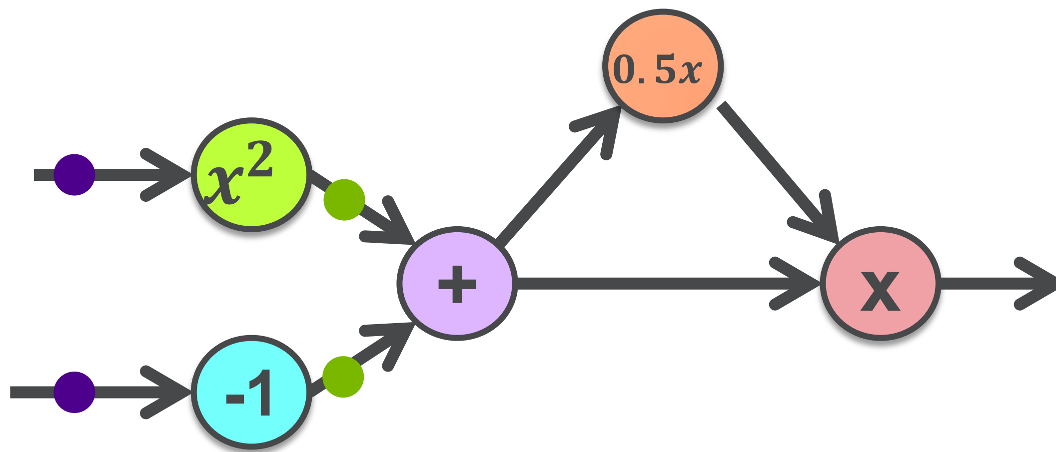
DATAFLOW PIPELINING

Allow multiple executions of the same dataflow graph



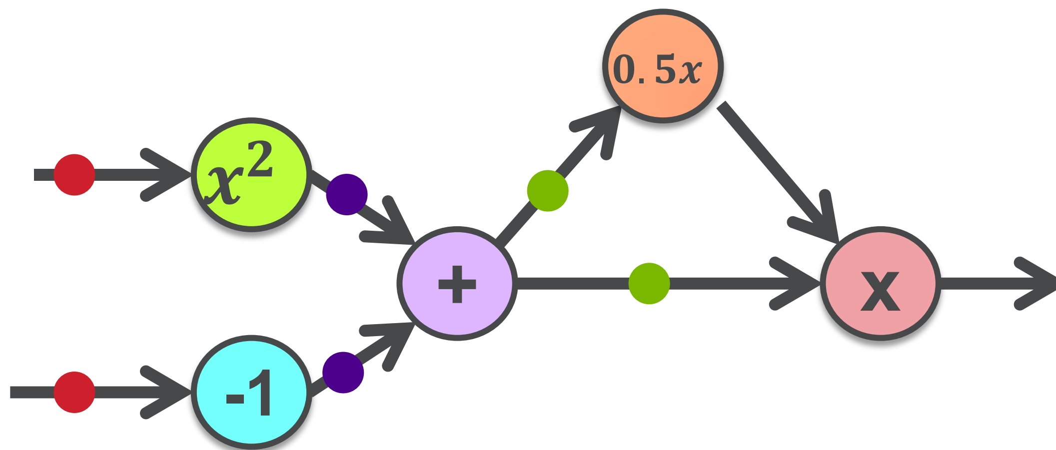
DATAFLOW PIPELINING

Allow multiple executions of the same dataflow graph



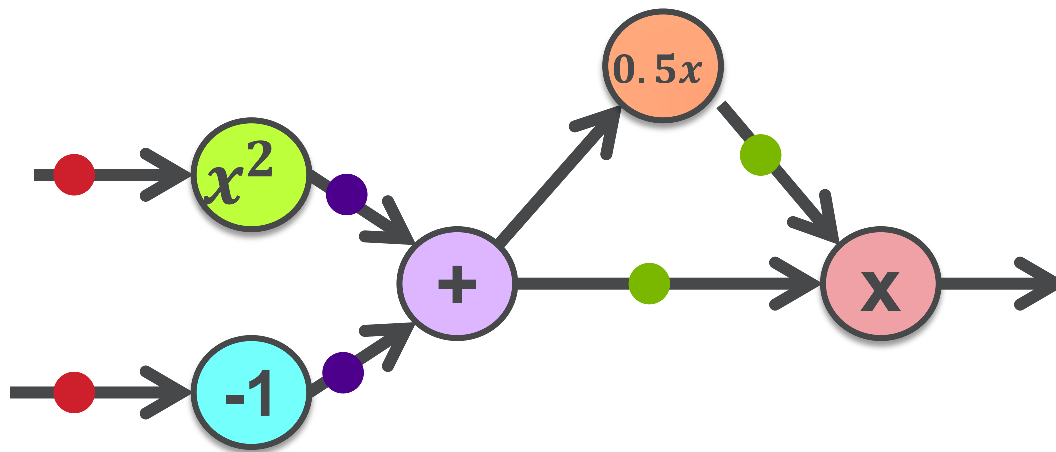
DATAFLOW PIPELINING

Allow multiple executions of the same dataflow graph



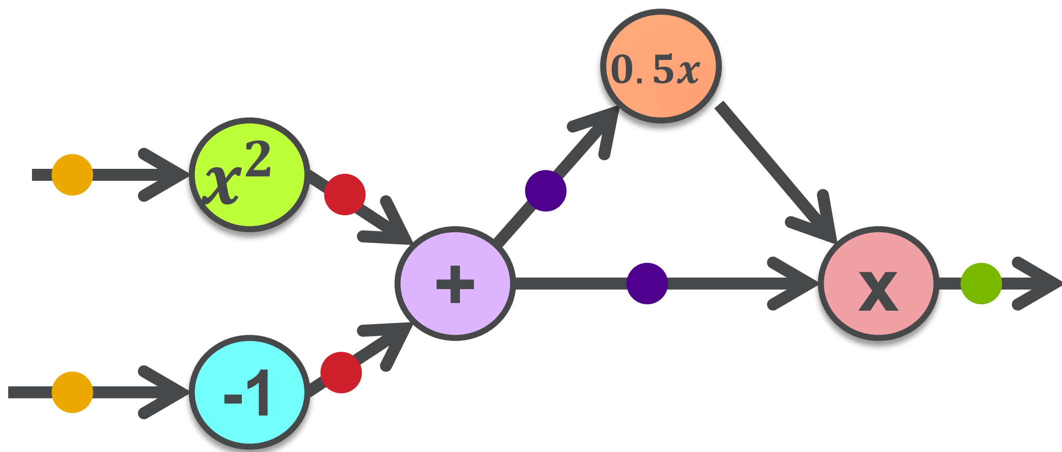
DATAFLOW PIPELINING

Allow multiple executions of the same dataflow graph



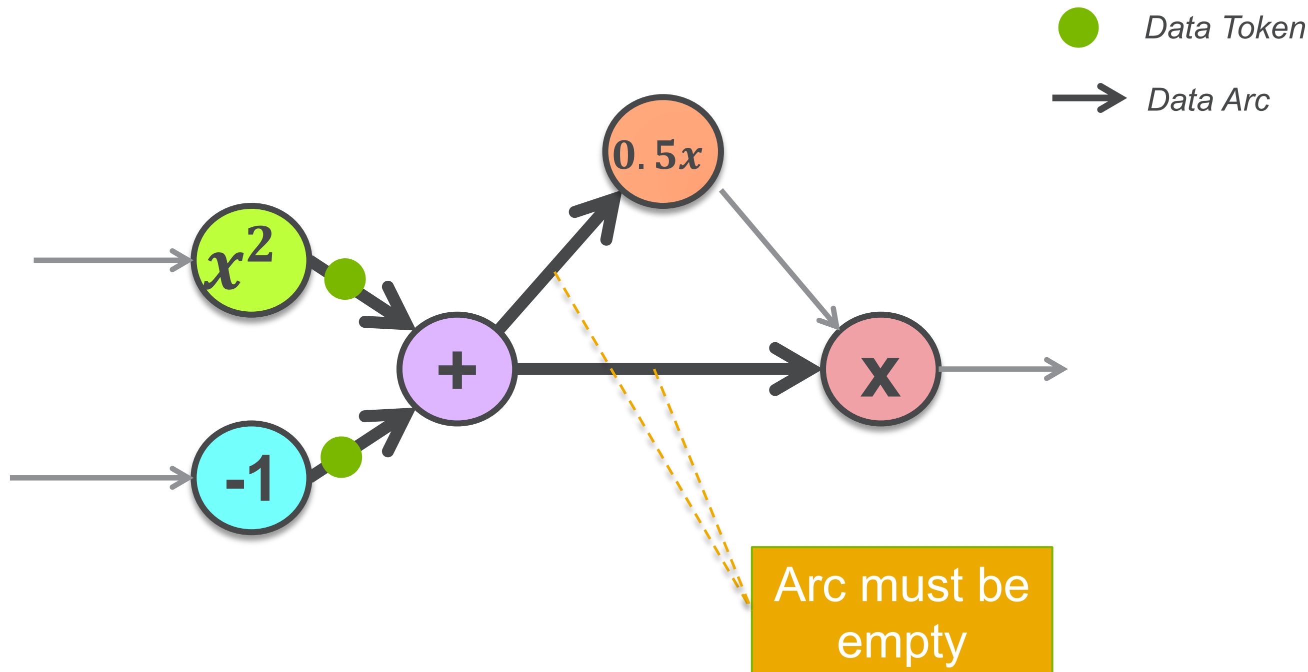
DATAFLOW PIPELINING

Allow multiple executions of the same dataflow graph



DATAFLOW PIPELINING

The back-pressure problem



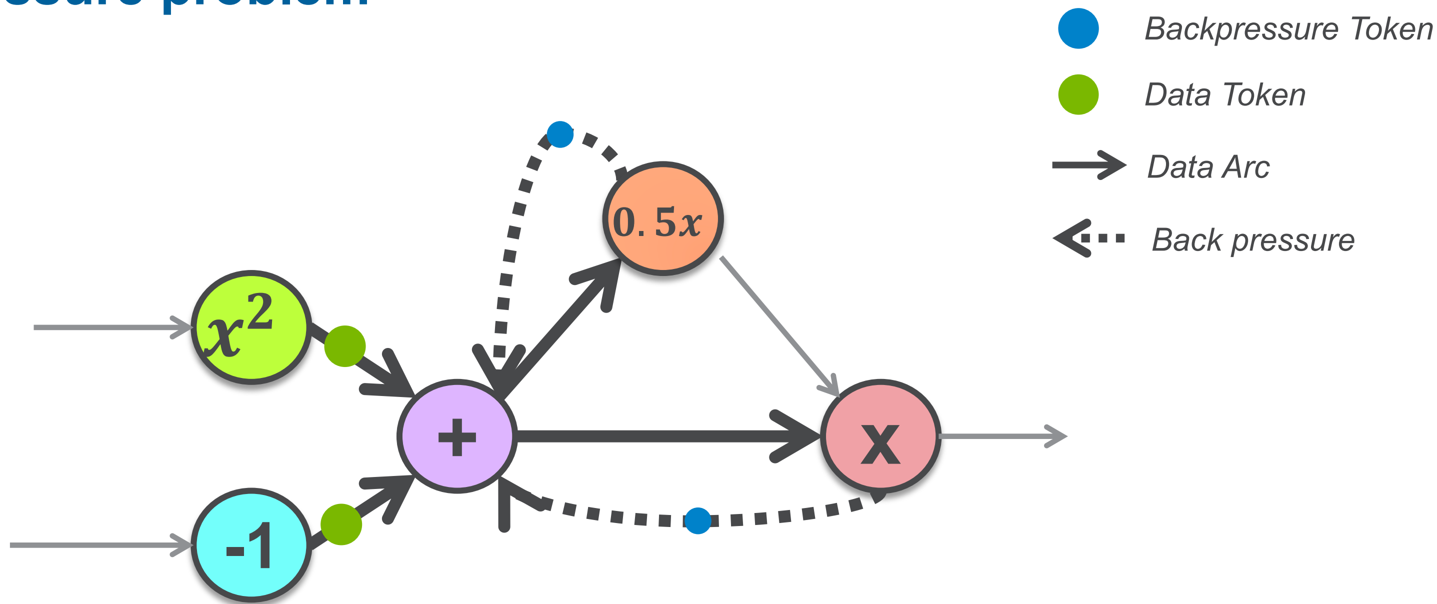
DATAFLOW PIPELINING: THE BACK-PRESSURE PROBLEM

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

Argonne
NATIONAL LABORATORY

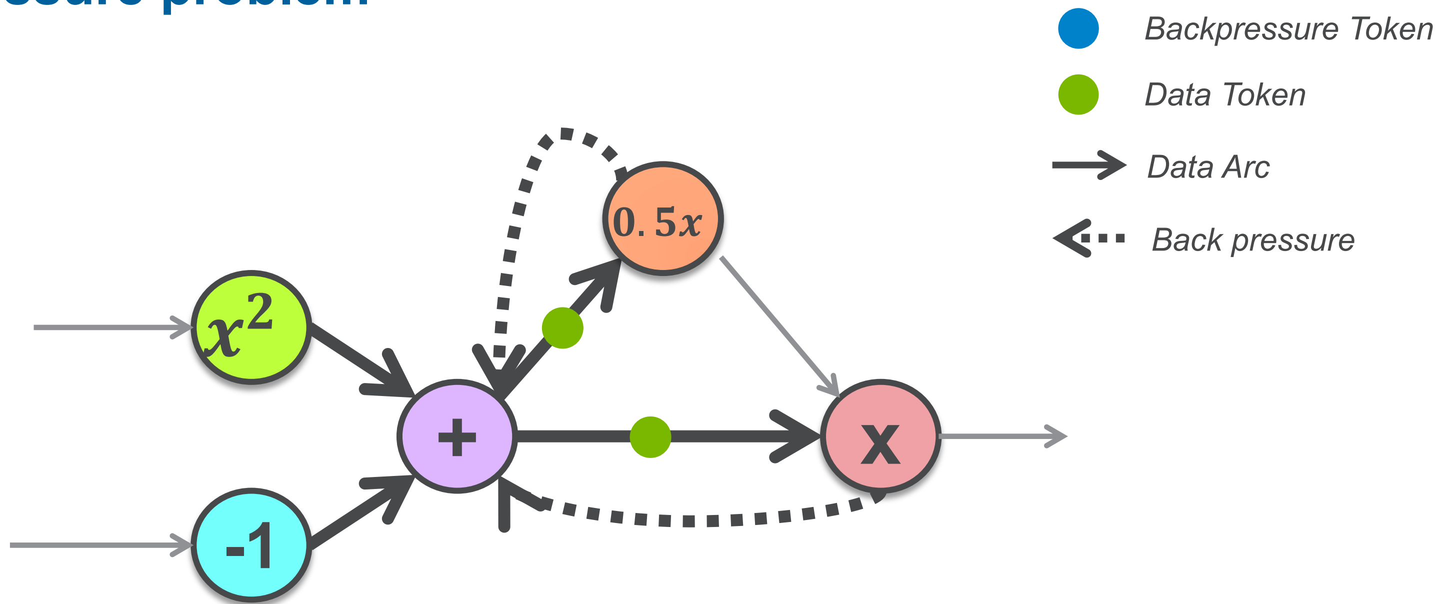
DATAFLOW PIPELINING

The back-pressure problem



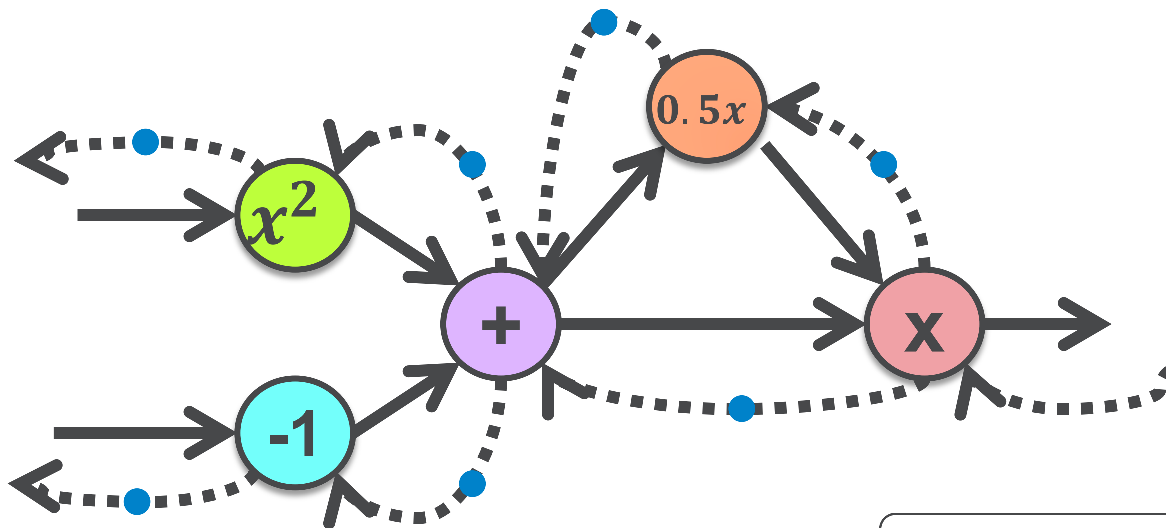
DATAFLOW PIPELINING

The back-pressure problem



DATAFLOW PIPELINING

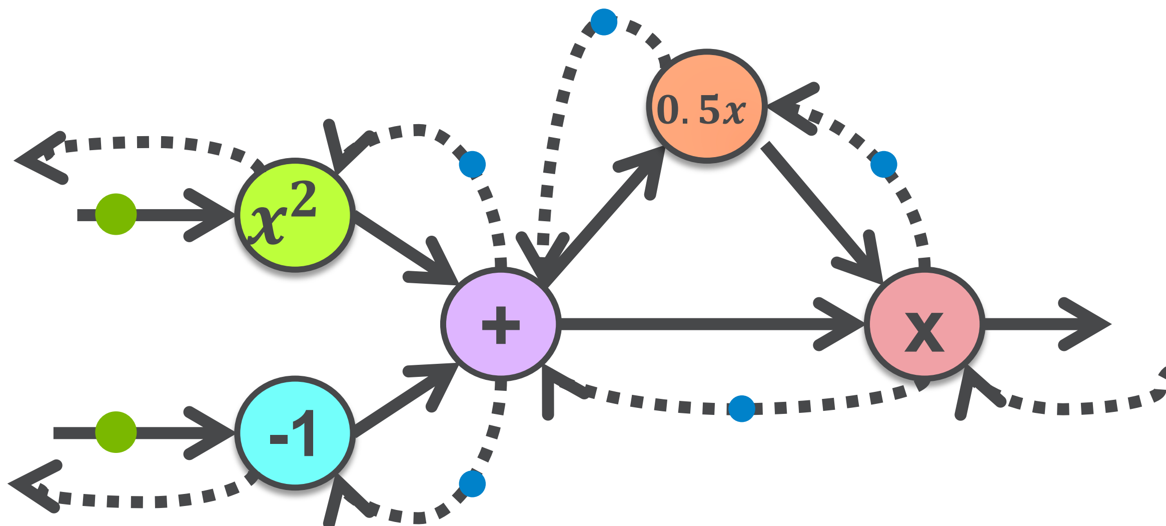
The back-pressure token



Initial State

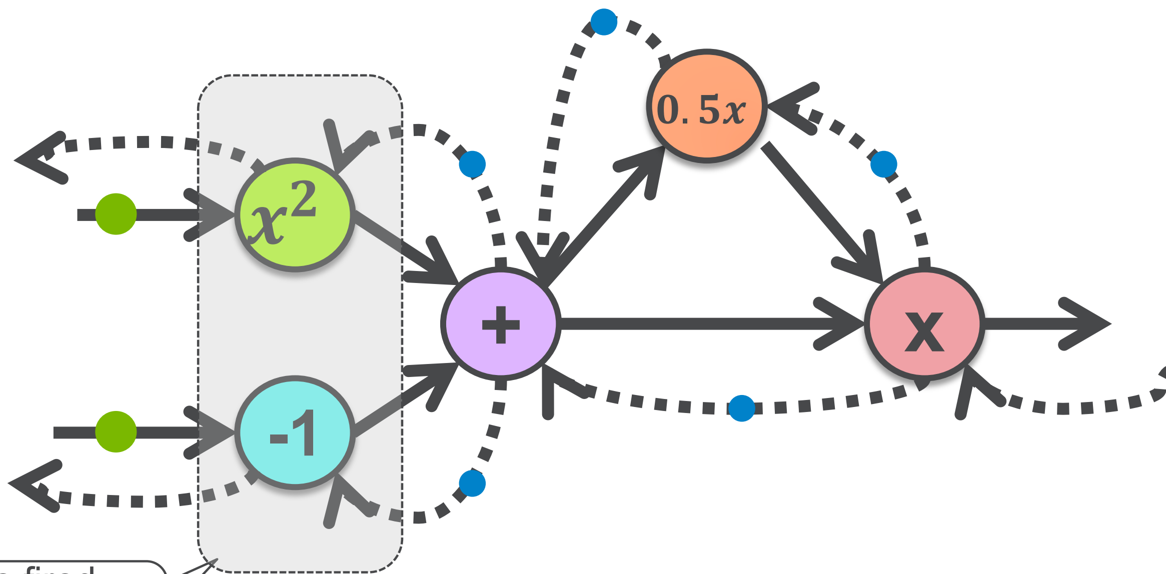
DATAFLOW PIPELINING

The back-pressure token



DATAFLOW PIPELINING

The back-pressure token

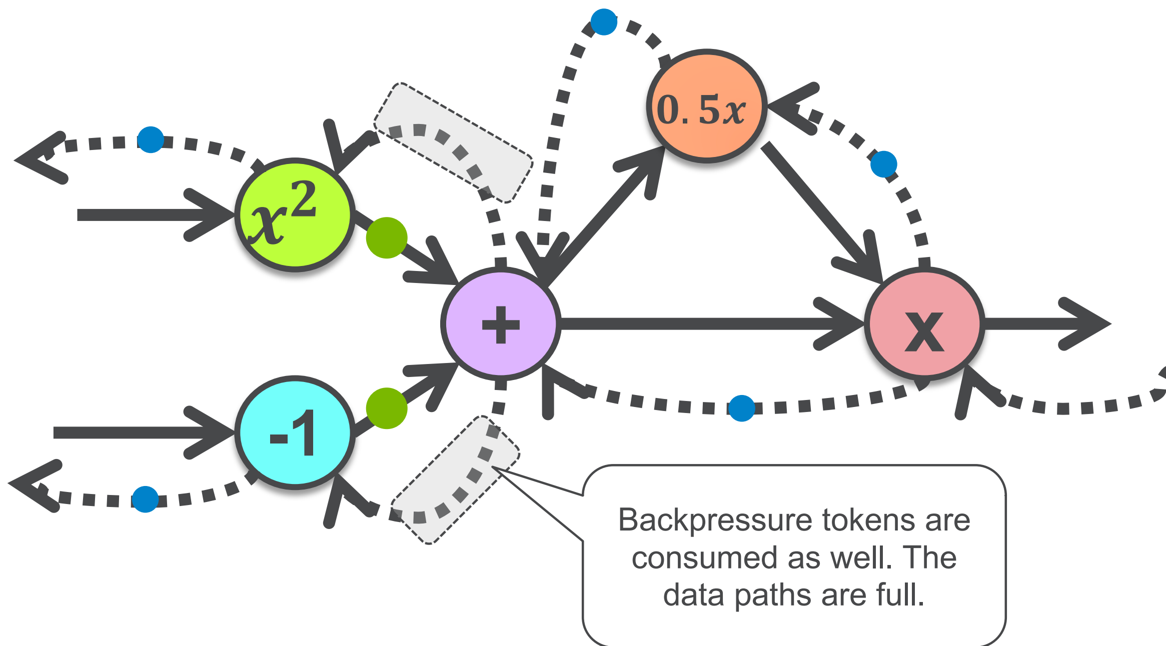


Actors can be fired since ACK tokens are present in their outputs

Argonne NATIONAL LABORATORY

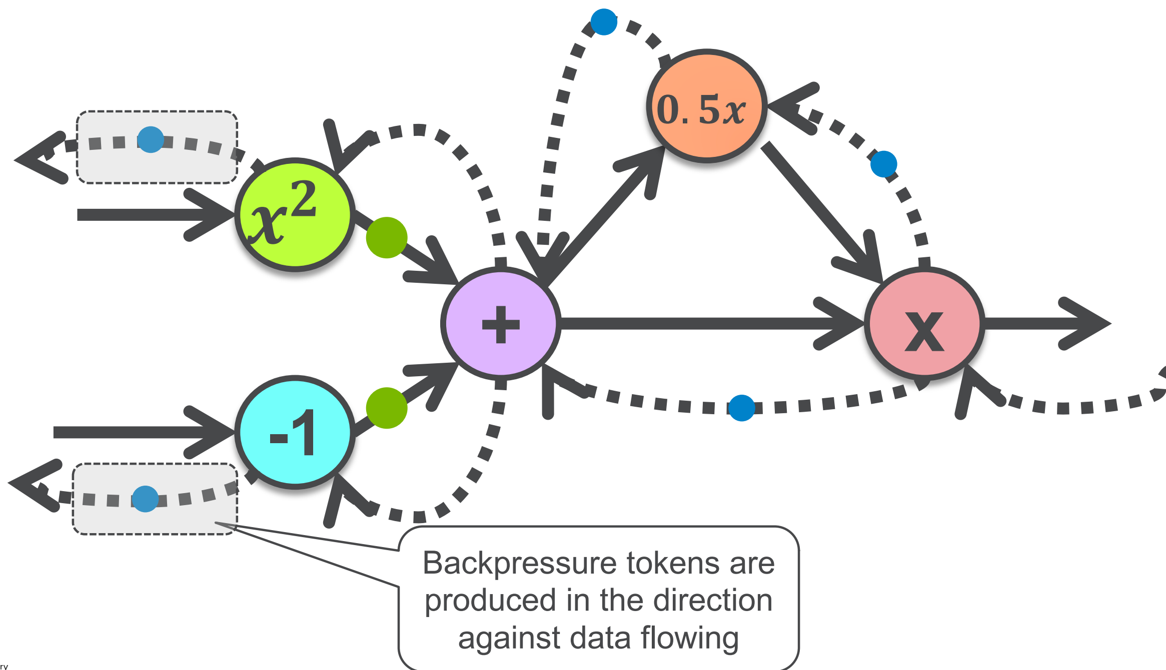
DATAFLOW PIPELINING

The back-pressure token



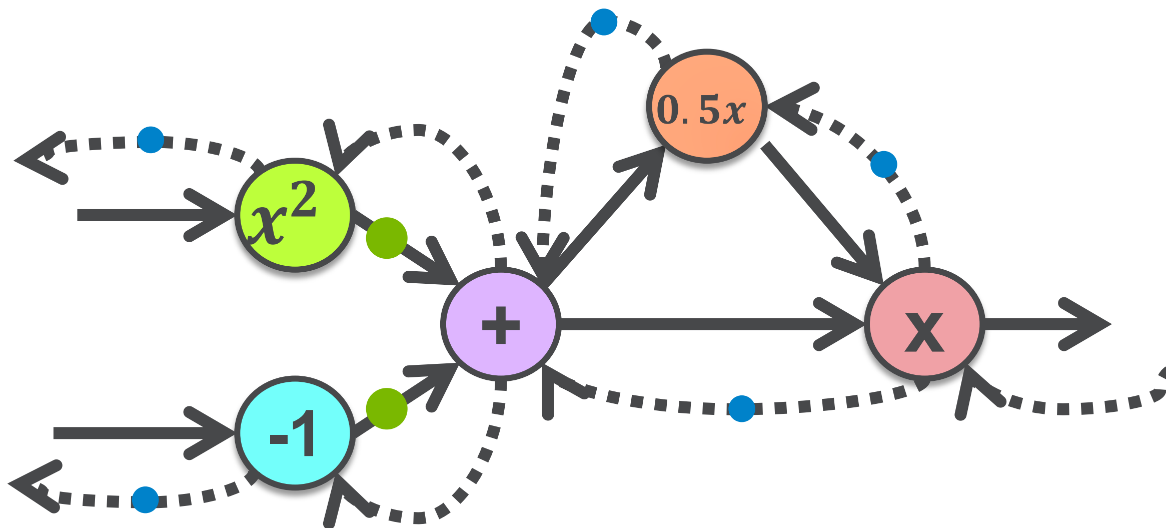
DATAFLOW PIPELINING

The back-pressure token



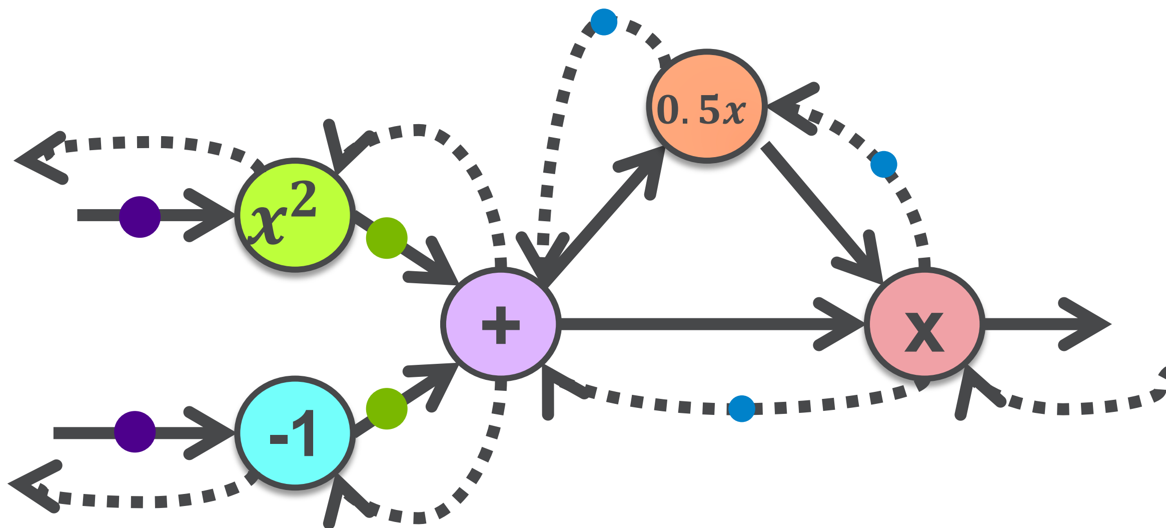
DATAFLOW PIPELINING

The back-pressure token



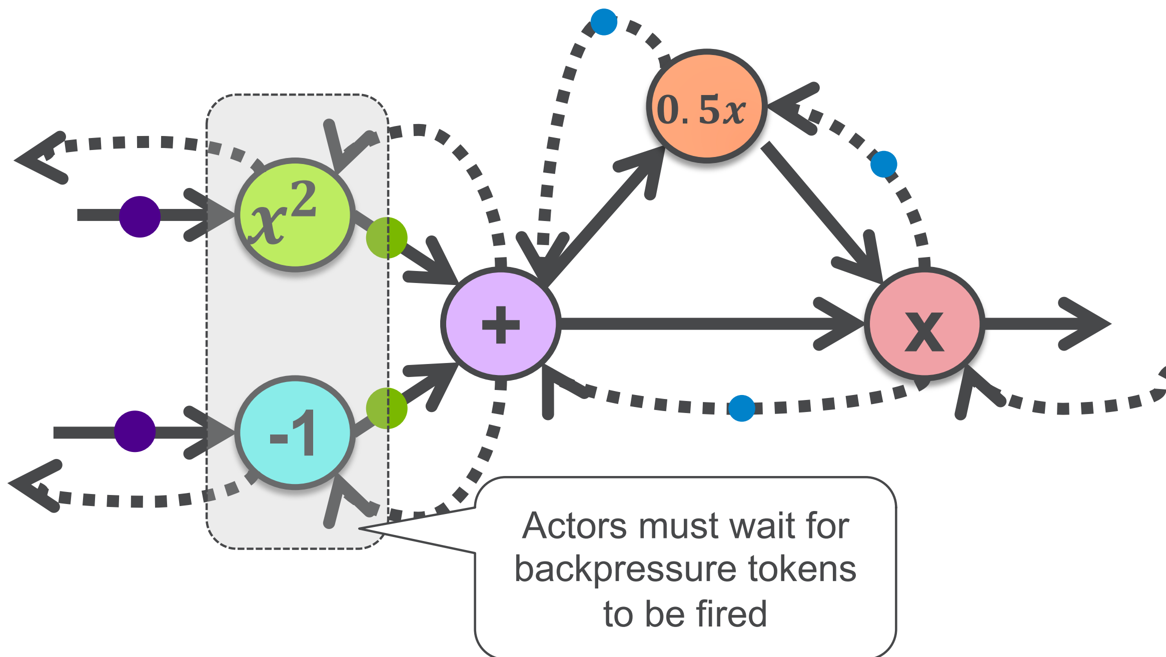
DATAFLOW PIPELINING

The back-pressure token



DATAFLOW PIPELINING

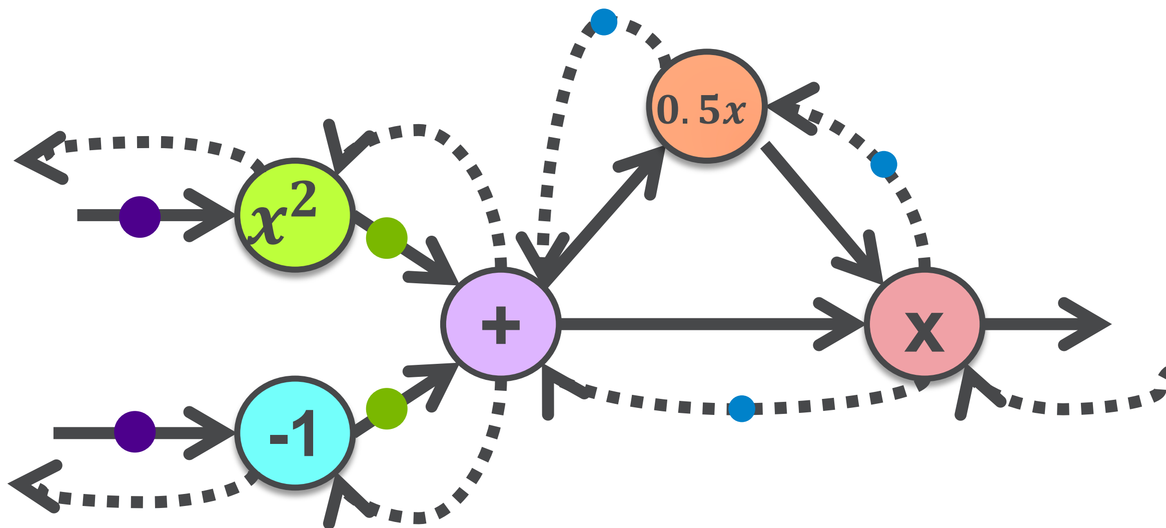
The back-pressure token



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

DATAFLOW PIPELINING

The back-pressure token



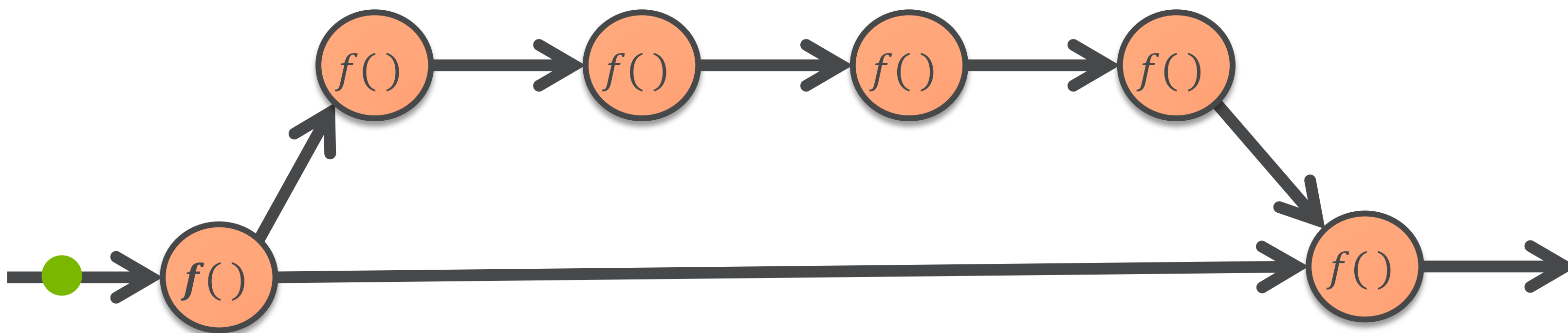
DATAFLOW PERFORMANCE: THE UNBALANCE PROBLEM

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

Argonne
NATIONAL LABORATORY

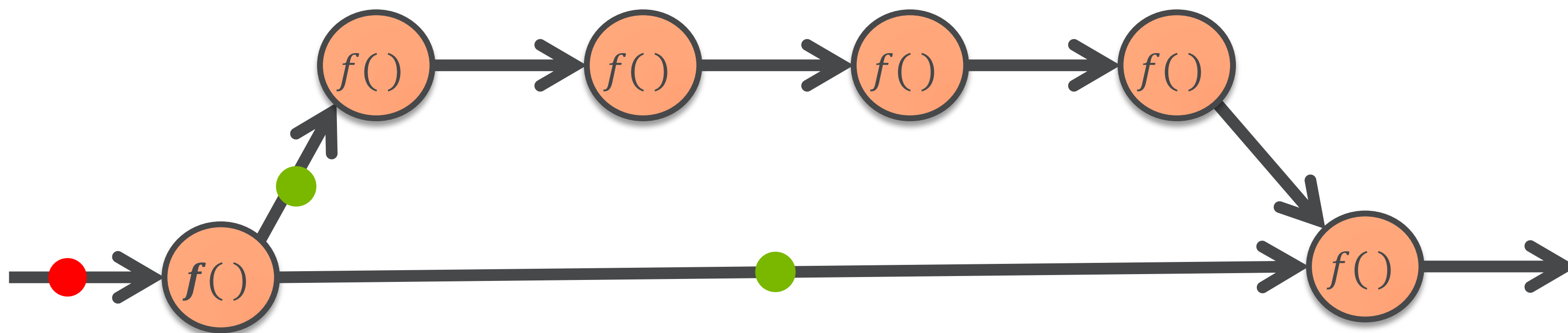
DATAFLOW PERFORMANCE

Path unbalance



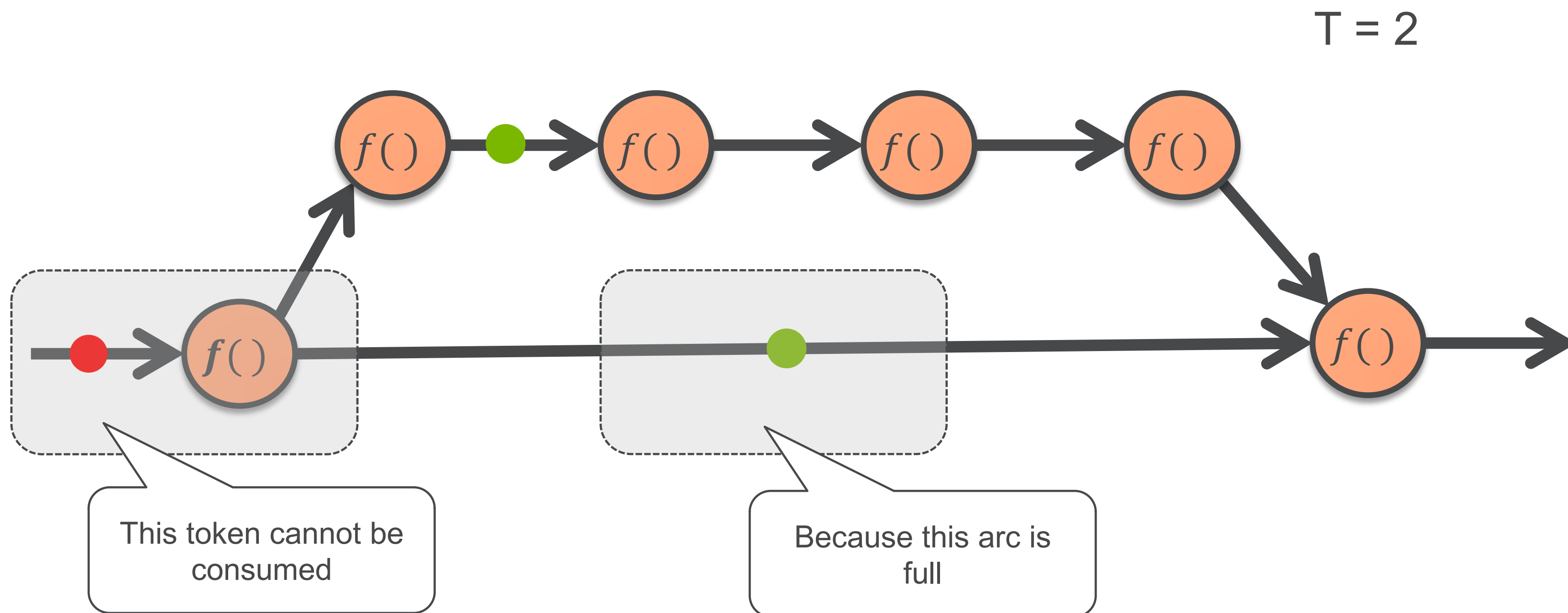
DATAFLOW PERFORMANCE

Path unbalance



DATAFLOW PERFORMANCE

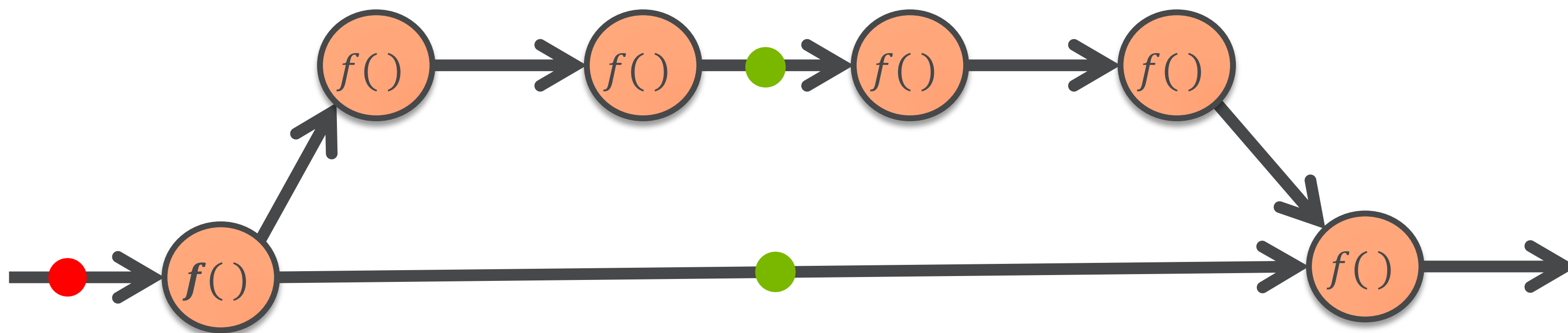
Path unbalance



DATAFLOW PERFORMANCE

Path unbalance

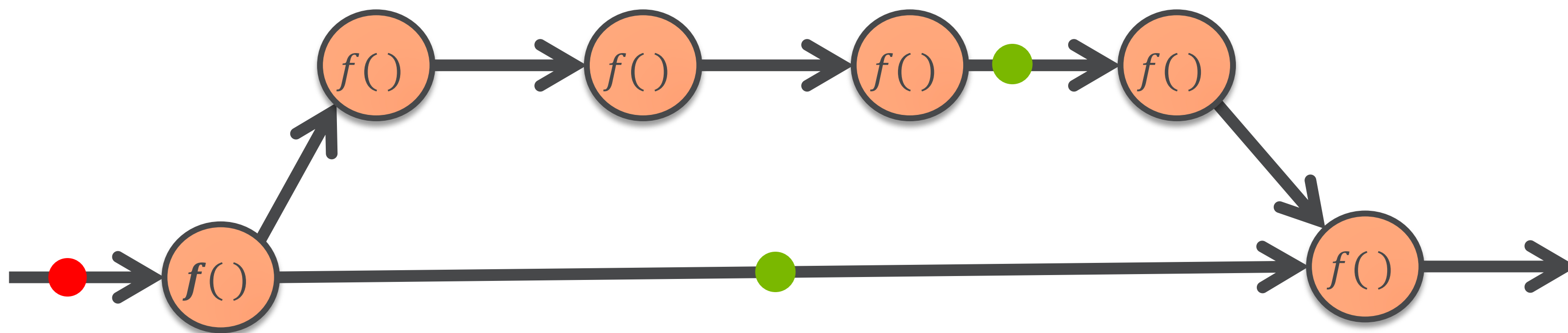
T = 3



DATAFLOW PERFORMANCE

Path unbalance

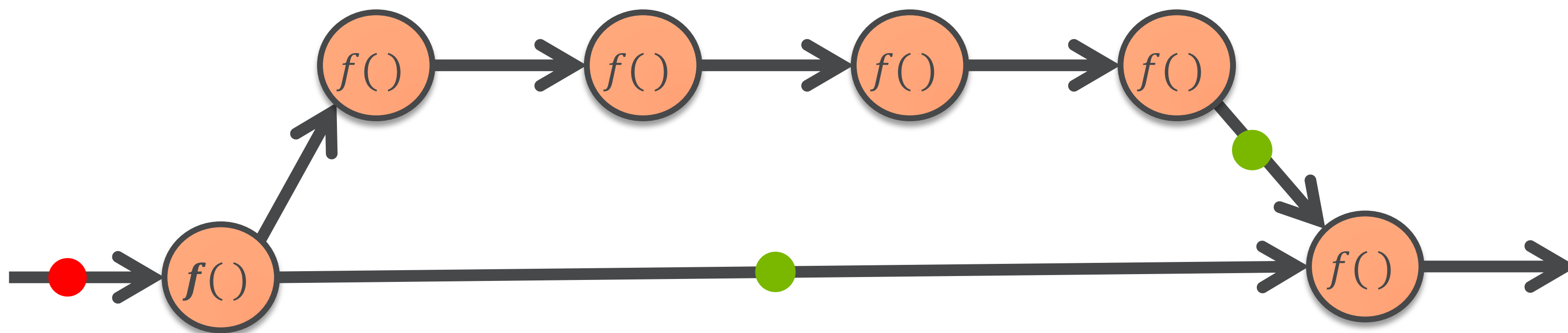
T = 4



DATAFLOW PERFORMANCE

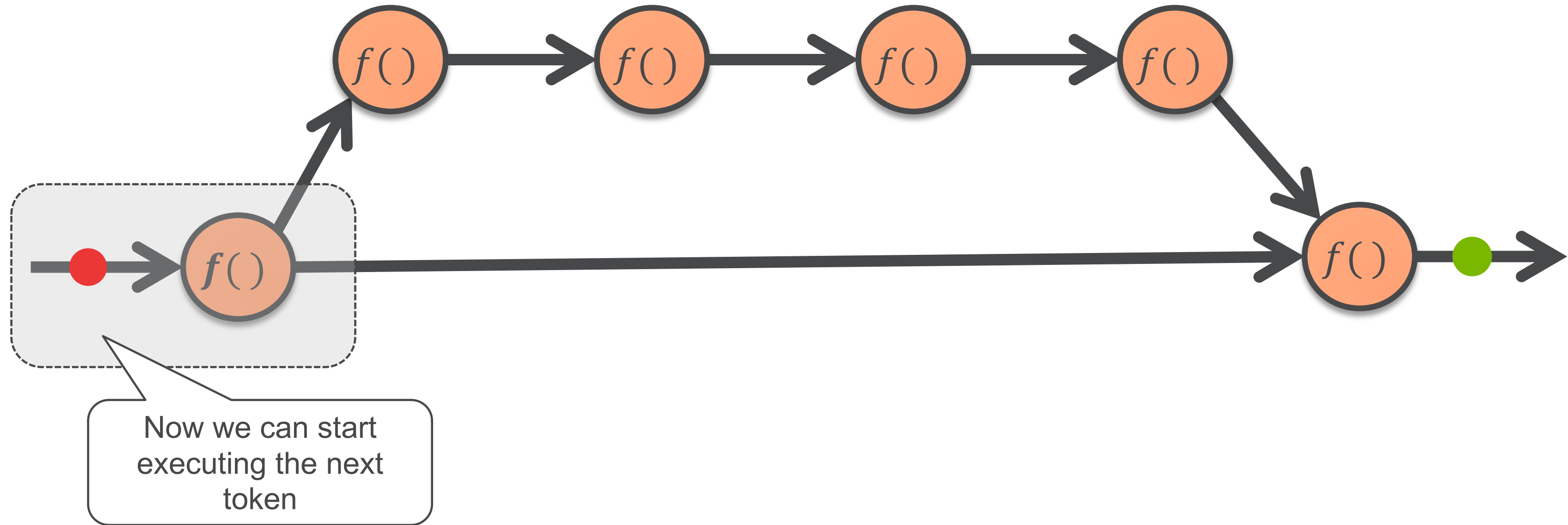
Path unbalance

T = 5



DATAFLOW PERFORMANCE

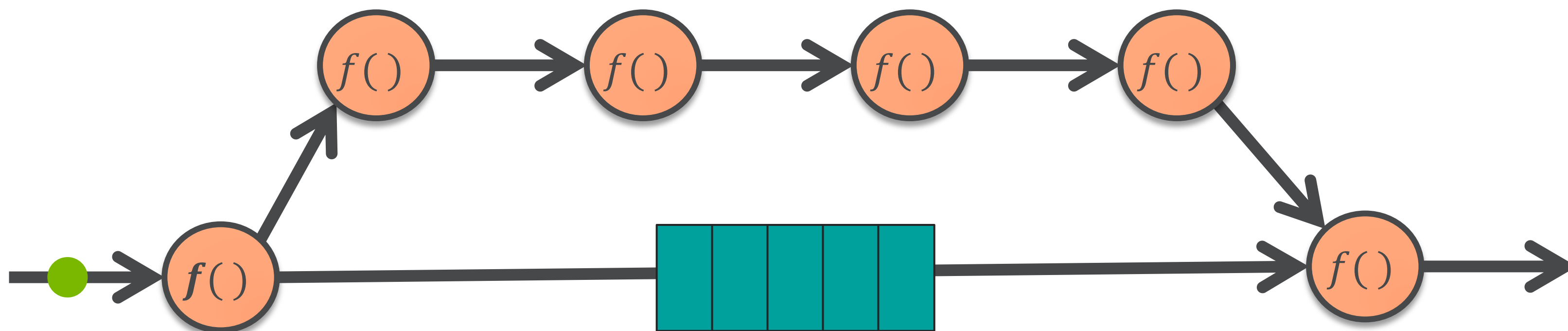
Path unbalance



DATAFLOW PERFORMANCE: BALANCING

DATAFLOW PERFORMANCE

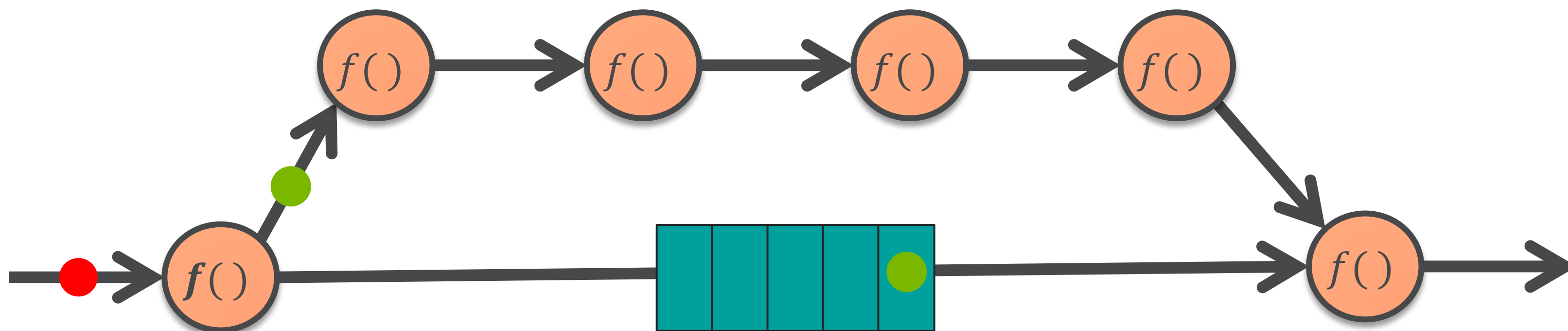
Queue insertion



DATAFLOW PERFORMANCE

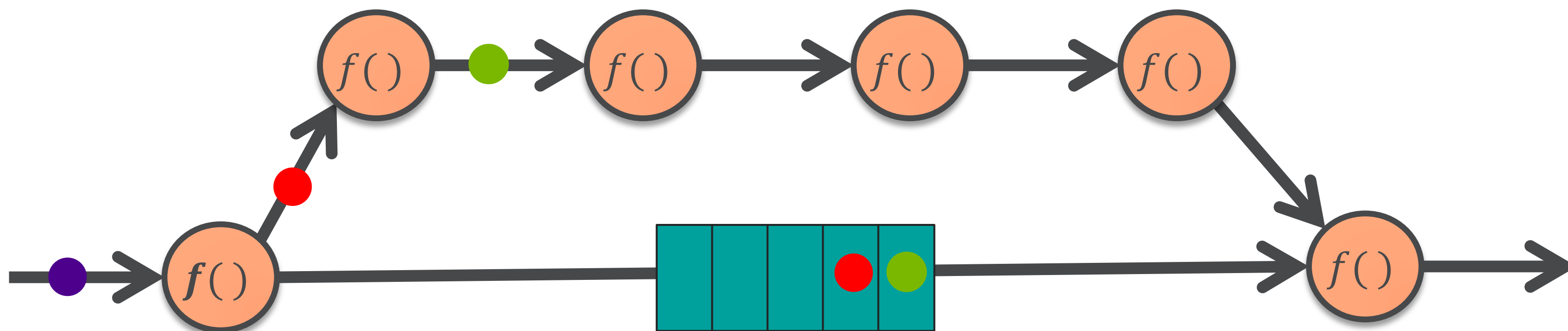
Queue insertion

T = 1



DATAFLOW PERFORMANCE

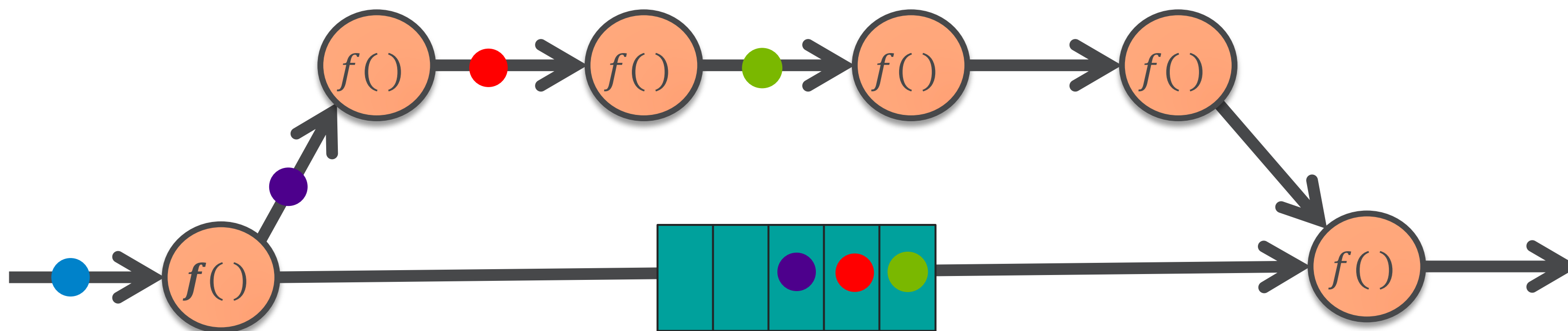
Queue insertion



DATAFLOW PERFORMANCE

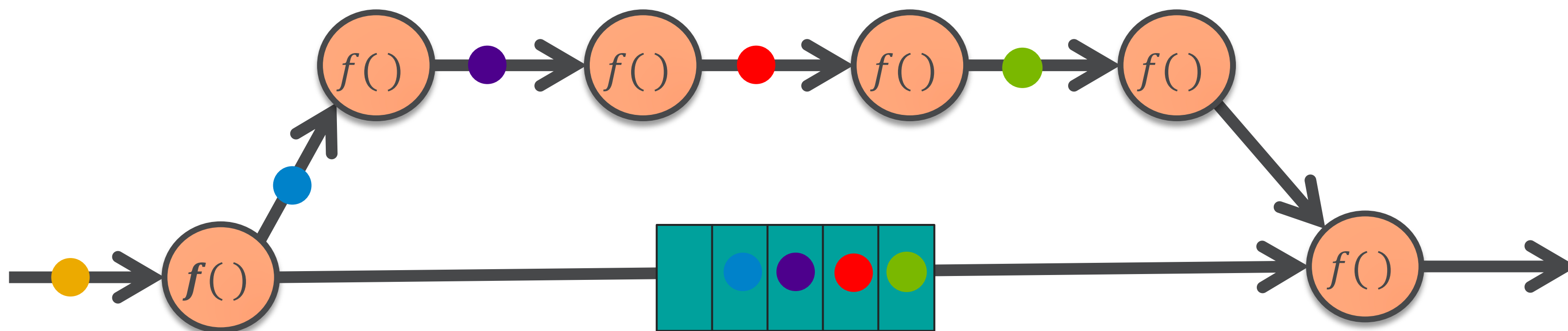
Queue insertion

T = 3



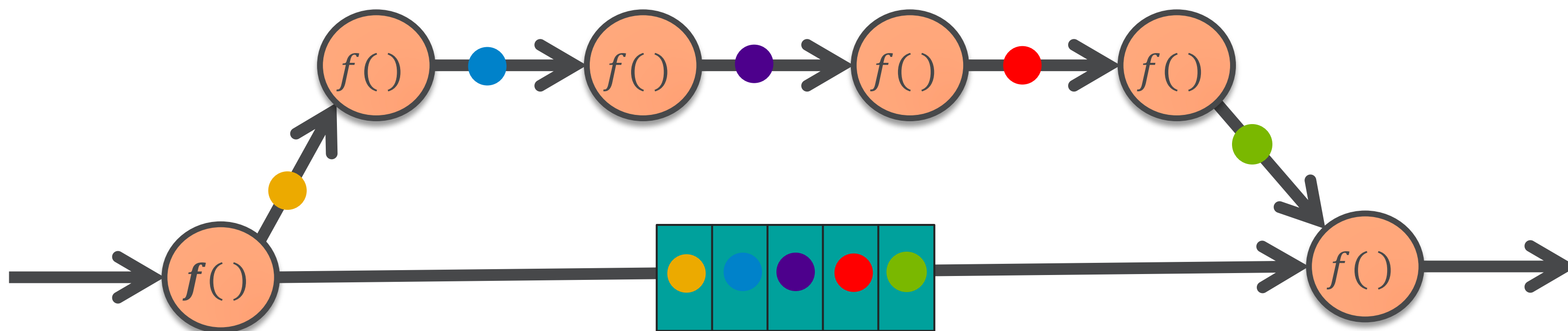
DATAFLOW PERFORMANCE

Queue insertion



DATAFLOW PERFORMANCE

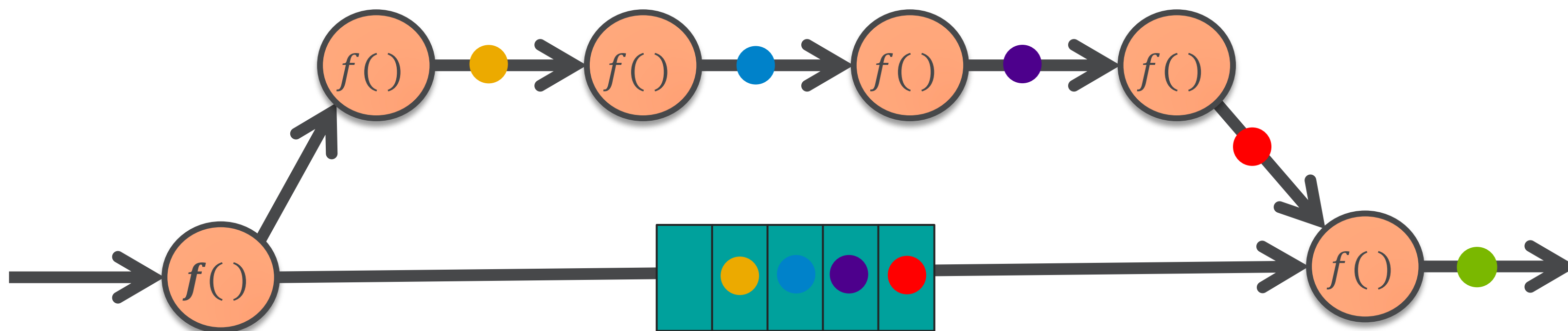
Queue insertion



DATAFLOW PERFORMANCE

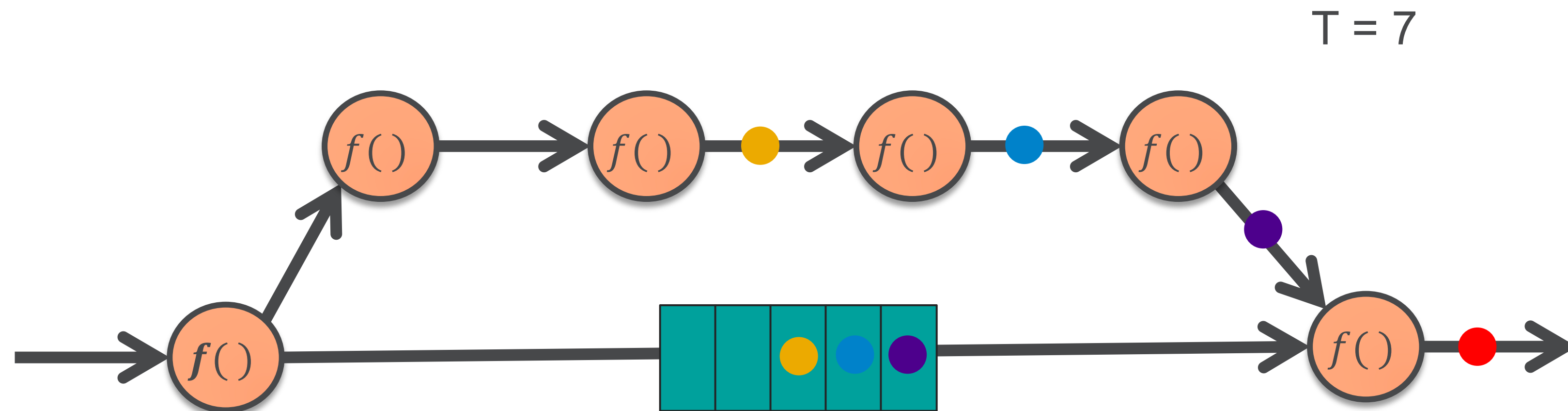
Queue insertion

T = 6



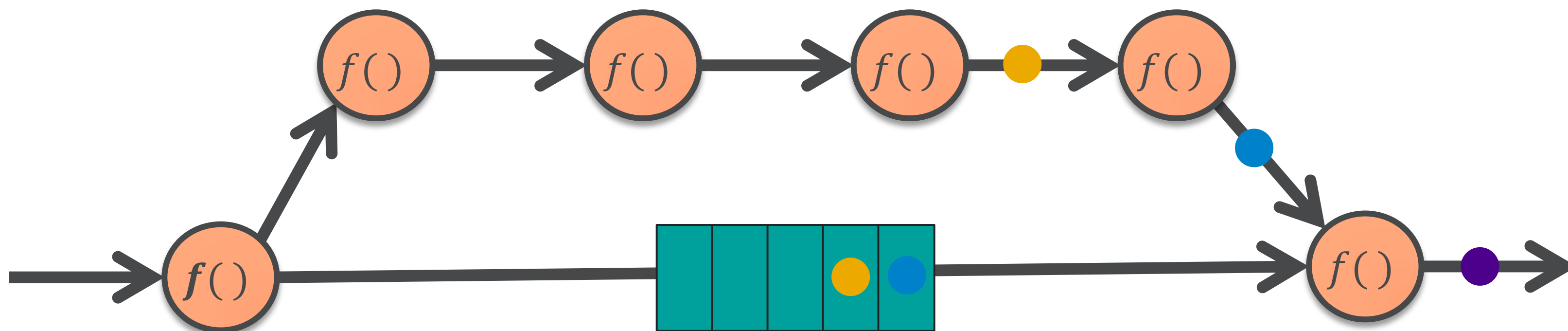
DATAFLOW PERFORMANCE

Queue insertion



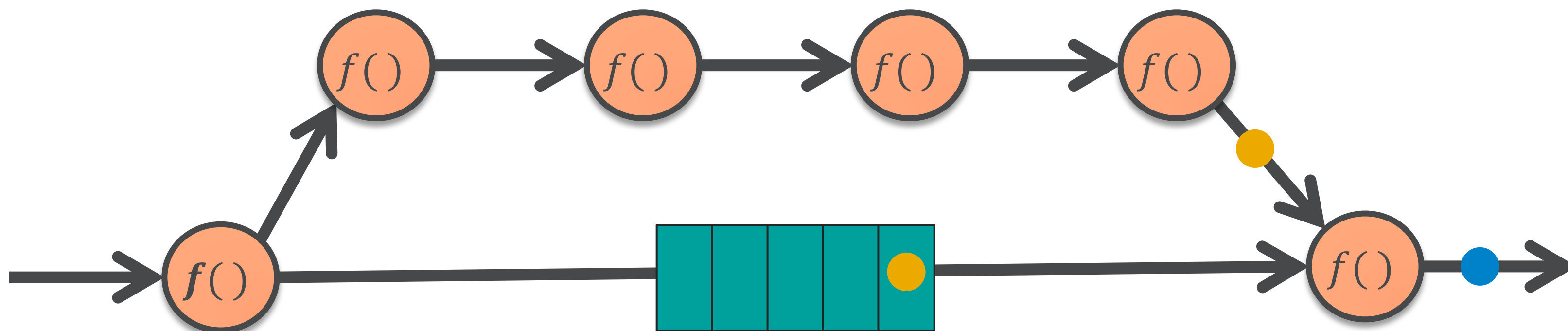
DATAFLOW PERFORMANCE

Queue insertion



DATAFLOW PERFORMANCE

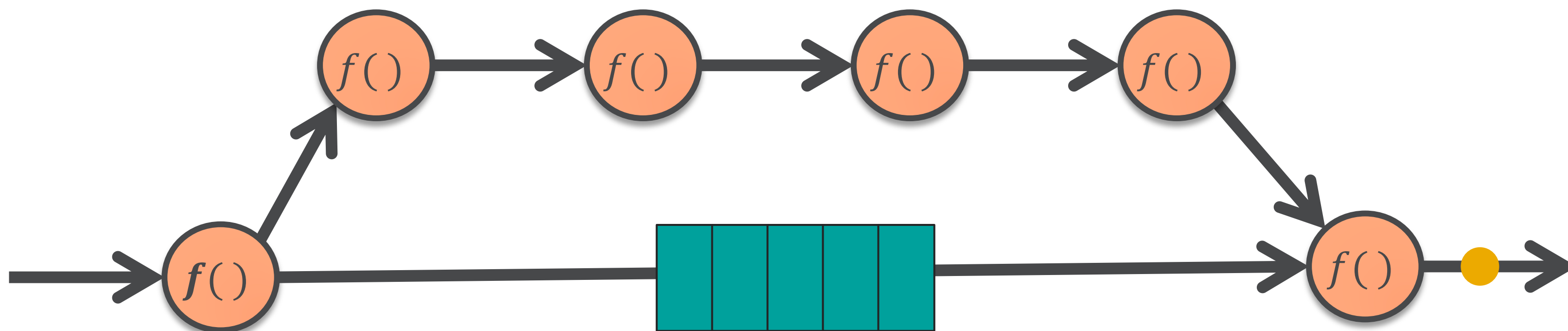
Queue insertion



DATAFLOW PERFORMANCE

Queue insertion

T = 10



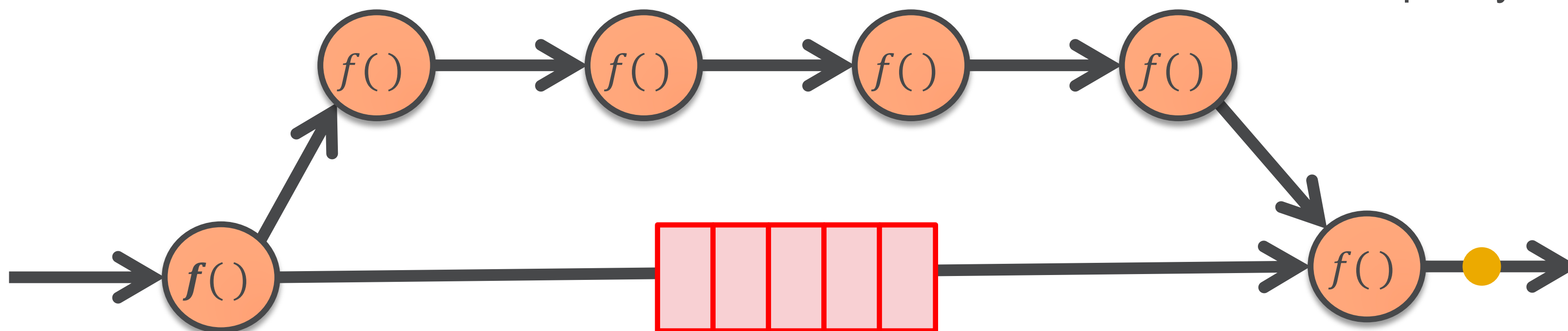
DATAFLOW PERFORMANCE

Queue insertion – Load balancing

10 tokens

T = 10

After warm up (6 cycles)
1 token per cycle

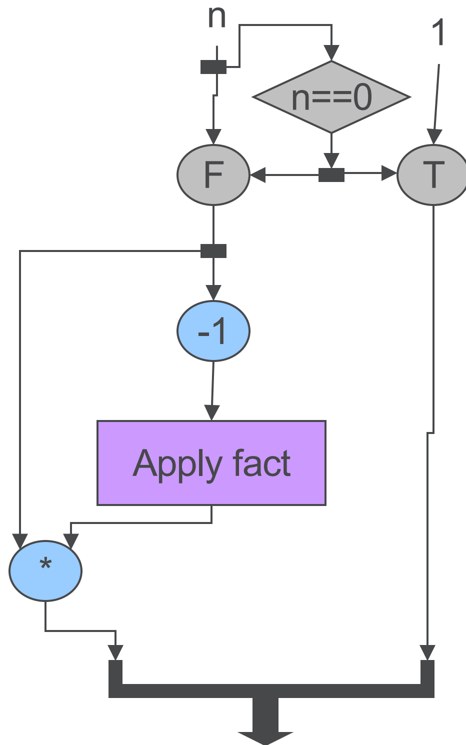


The queue allows balancing the length of the paths

Guang R. Gao, Algorithmic aspects of balancing techniques for pipelined data flow code generation, Journal of Parallel and Distributed Computing, Volume 6, Issue 1, 1989, Pages 39-61, ISSN 0743-7315, https://doi.org/10.1016/0743-7315(89)90041-5.

STATIC VS DYNAMIC DATAFLOW
THE RE-ENTRY PROBLEM

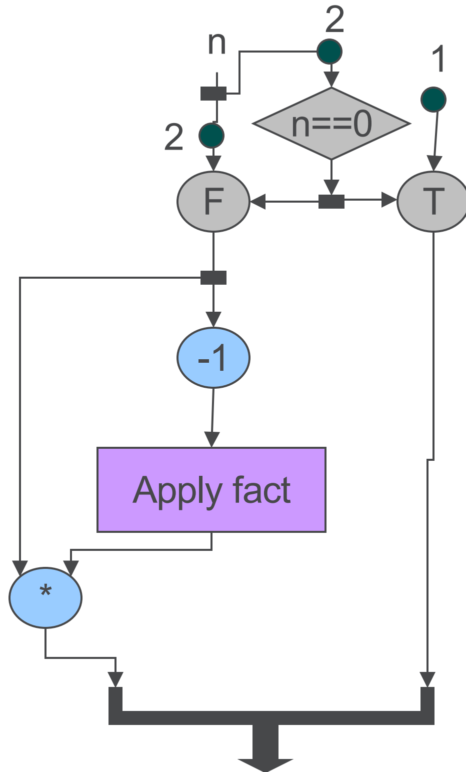
FACTORIAL




```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

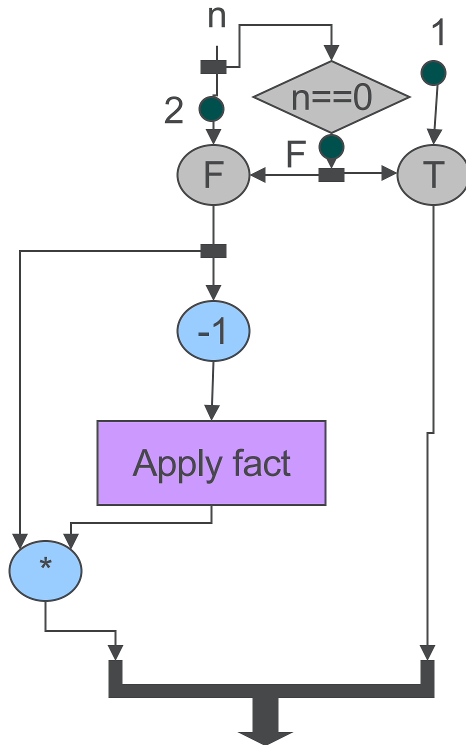


fact(2)

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

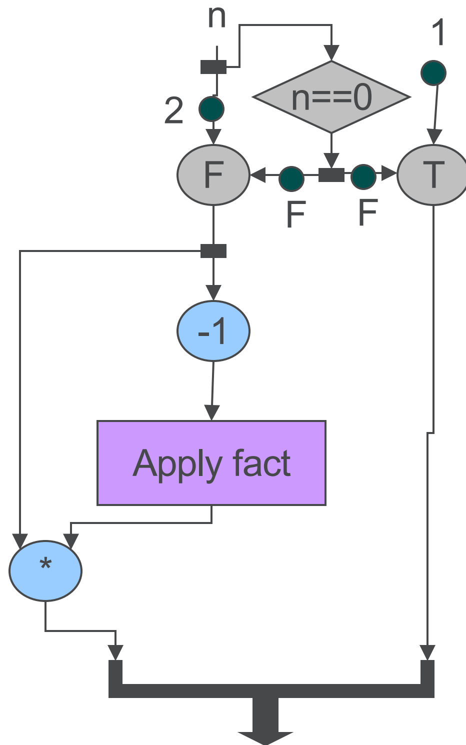


fact(2)

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

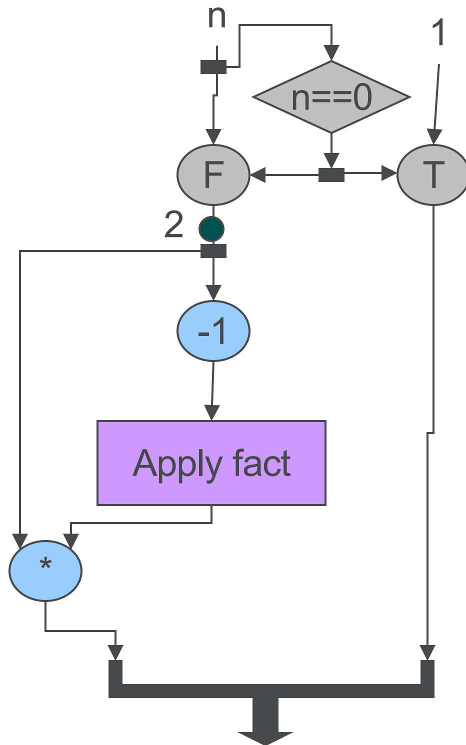


fact(2)

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

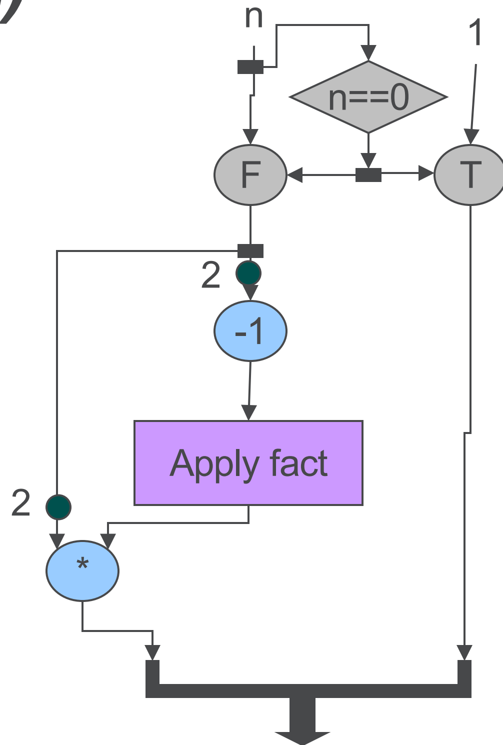


fact(2)

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

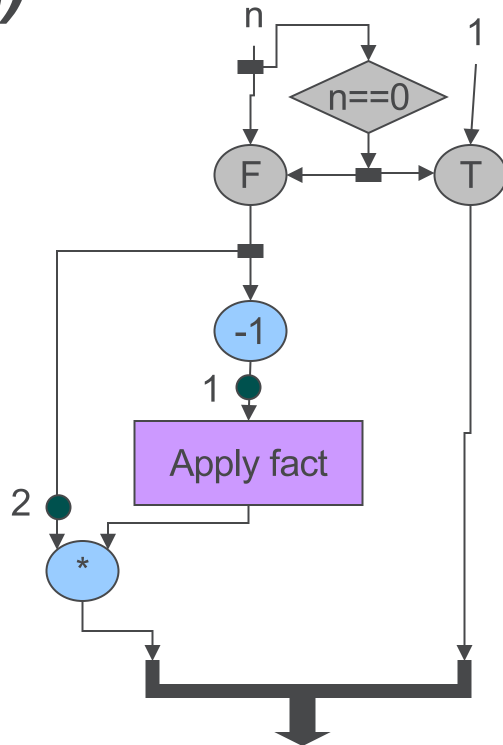


fact(2)

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

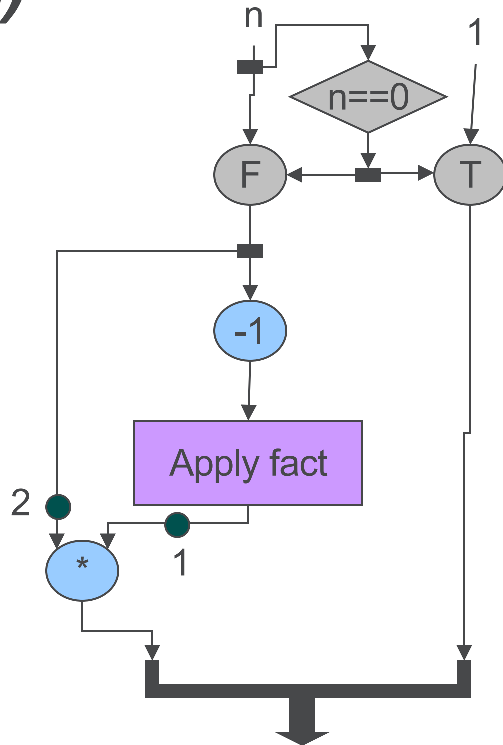


fact(2)

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

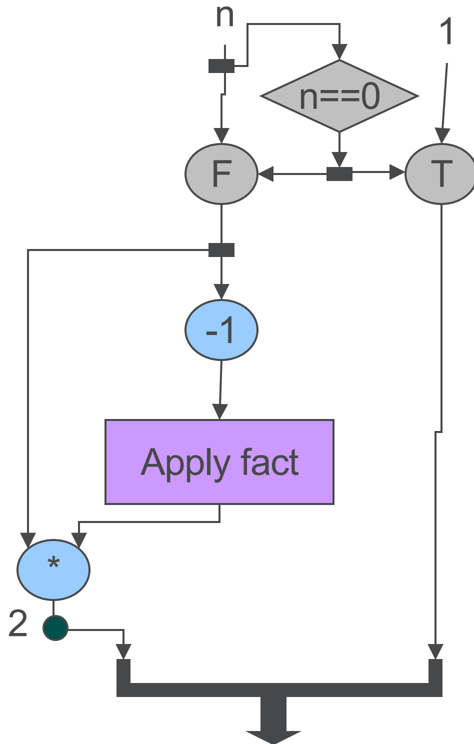


fact(2 * 1)

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

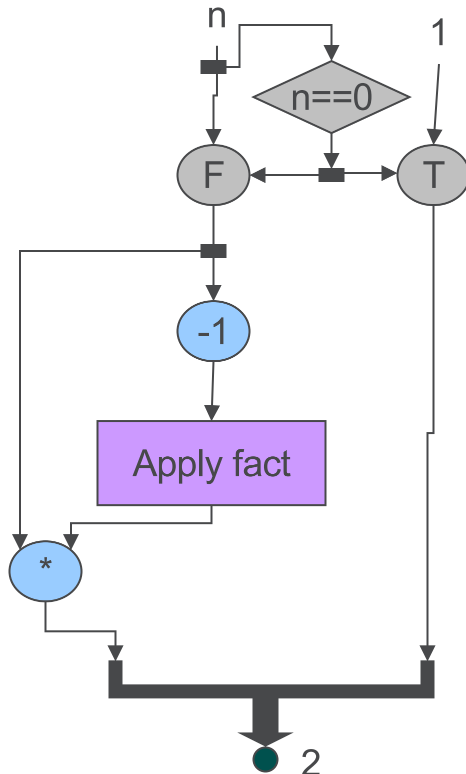


*fact(2 * 1)*

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

fact(2)

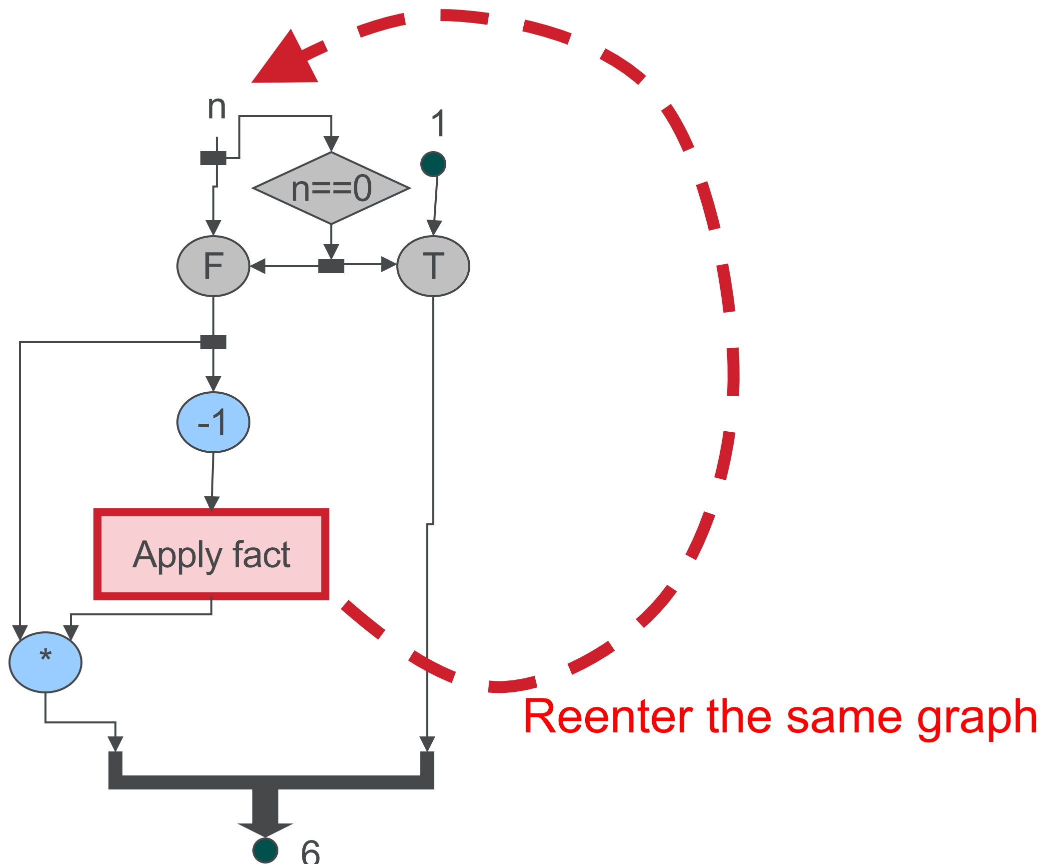


6

```
long fact(n) {
  if(n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

FACTORIAL

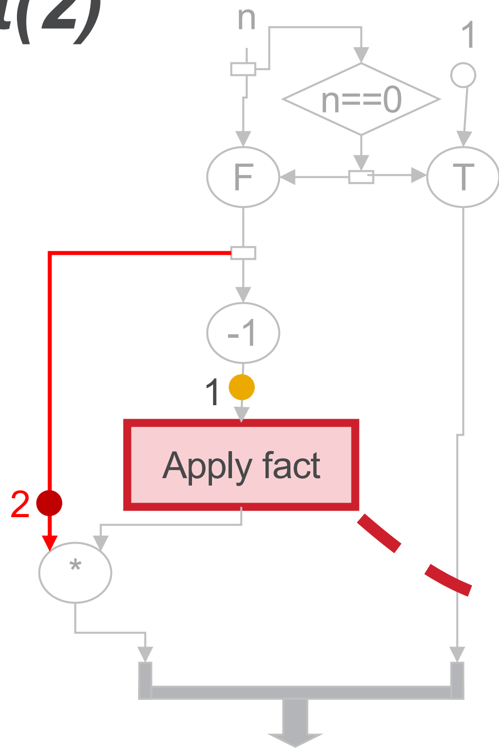
fact(n)



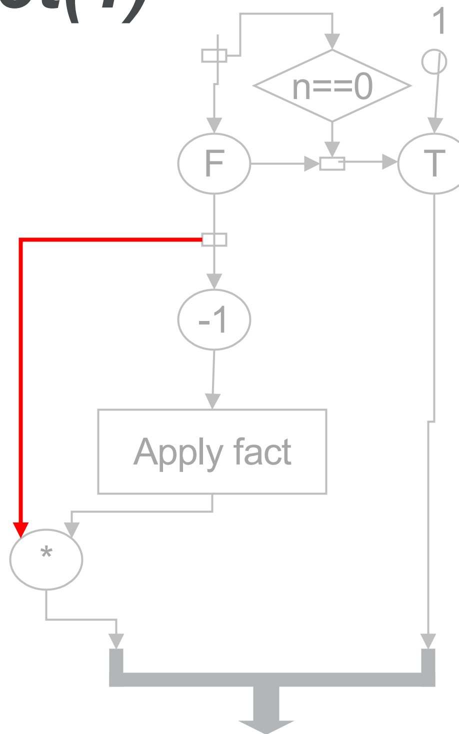
Reenter the same graph

FACTORIAL REENTRY PROBLEM

fact(2)

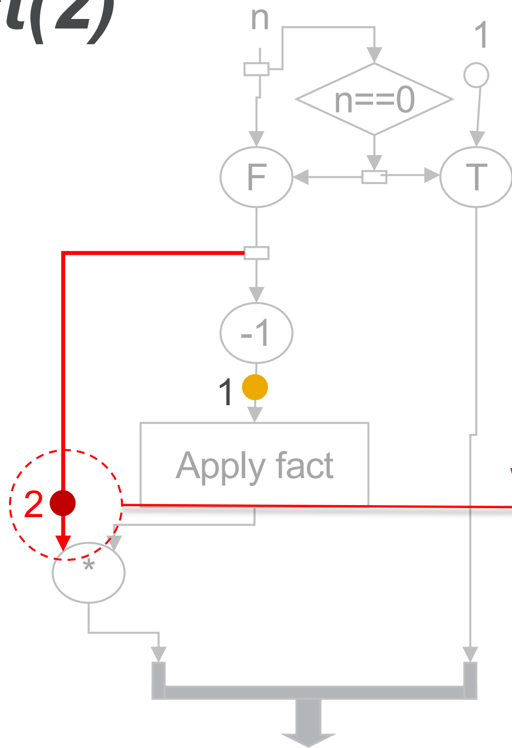


fact(1)



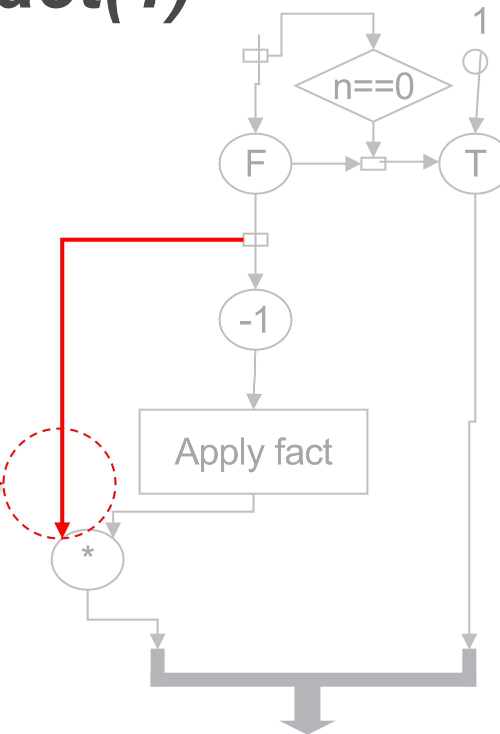
FACTORIAL REENTRY PROBLEM

fact(2)



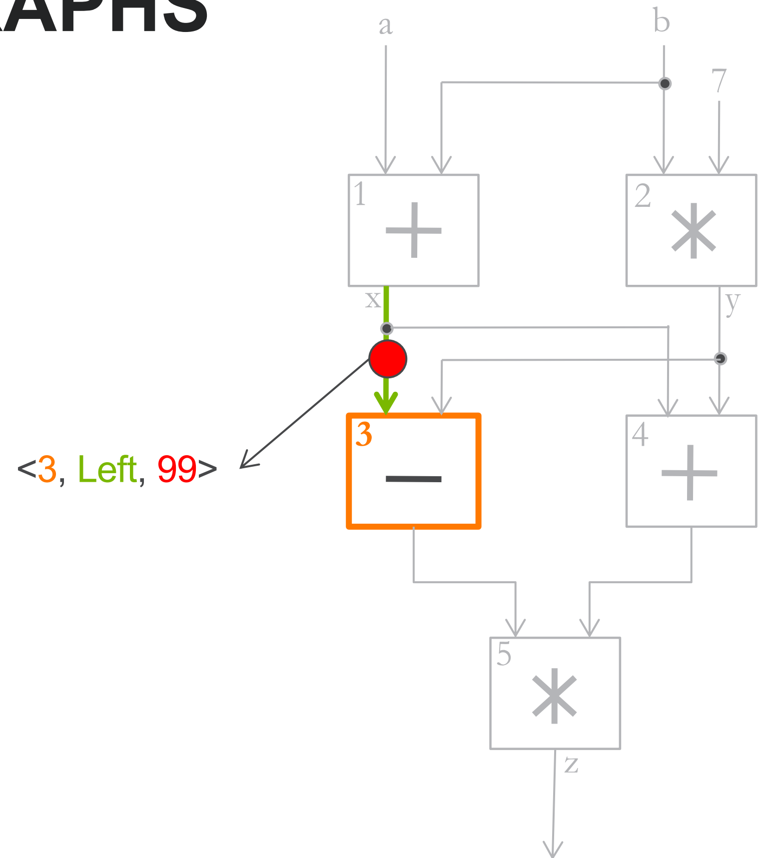
Where did it go

fact(1)



DYNAMIC DATAFLOW

STATIC DATAFLOW GRAPHS



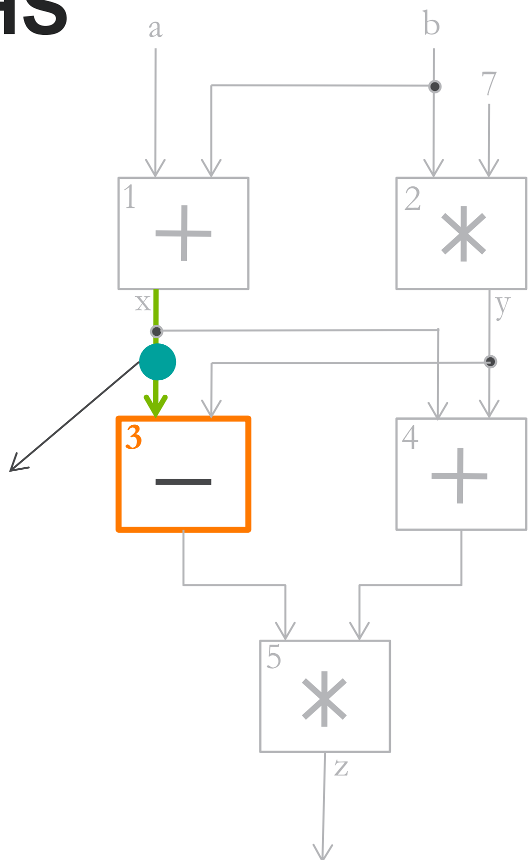
Values in **static** dataflow graphs represented as tokens

<*instruction_ptr*, *port*, *value*>

DYNAMIC DATAFLOW GRAPHS

a.k.a. Color token or tagged token dataflow

<turquoise, 3, Left, 99>



Values in **dynamic** dataflow graphs represented as tokens

<color, instruction_ptr, port, value>

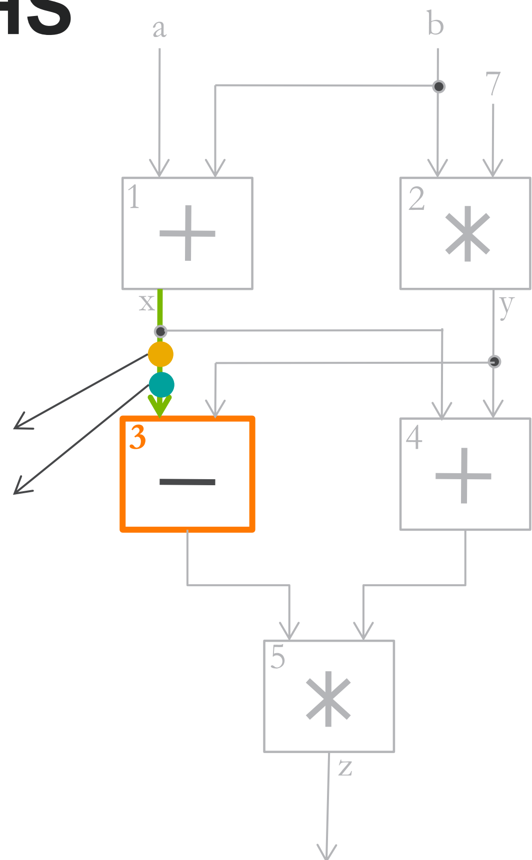
DYNAMIC DATAFLOW GRAPHS

a.k.a. Color token or tagged token dataflow

There may be multiple tokens per arc, as long as they are of different “color”

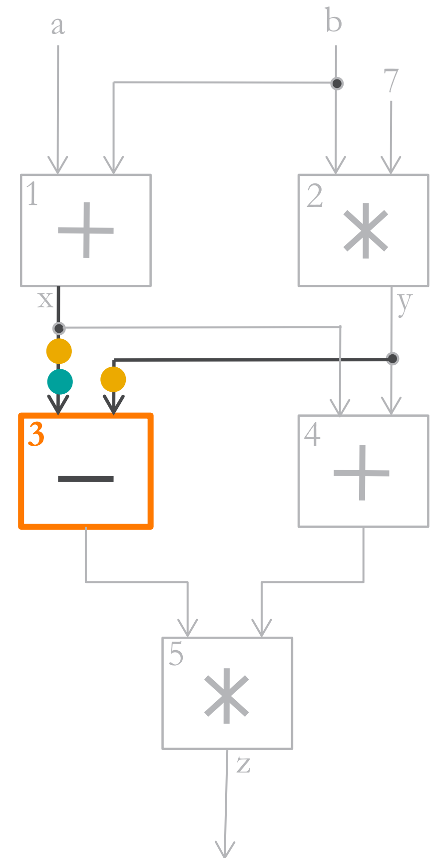
<yellow, 3, Left, 99>

<turquoise, 3, Left, 99>



DYNAMIC DATAFLOW GRAPHS

a.k.a. Color token or tagged token dataflow

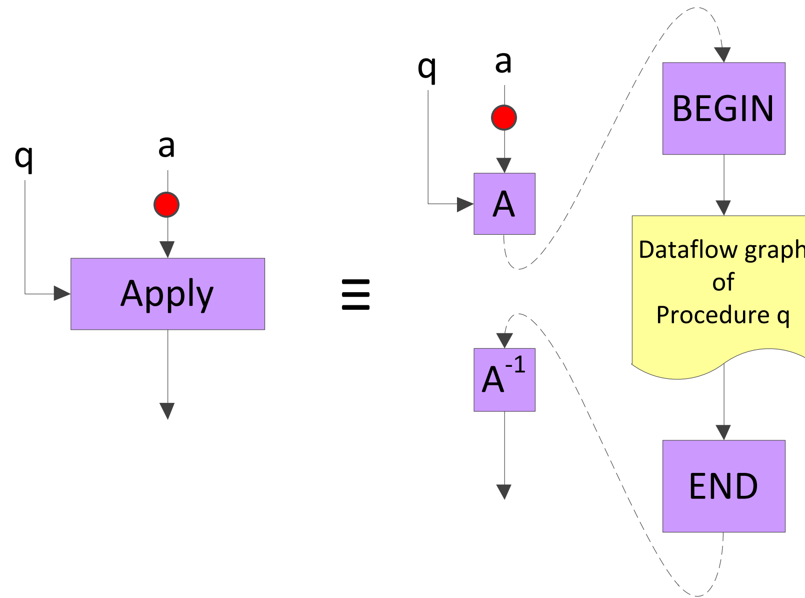


Operational semantics also change:

- Firing Rules ➔ All tokens **of the same color** are present in the input arcs.

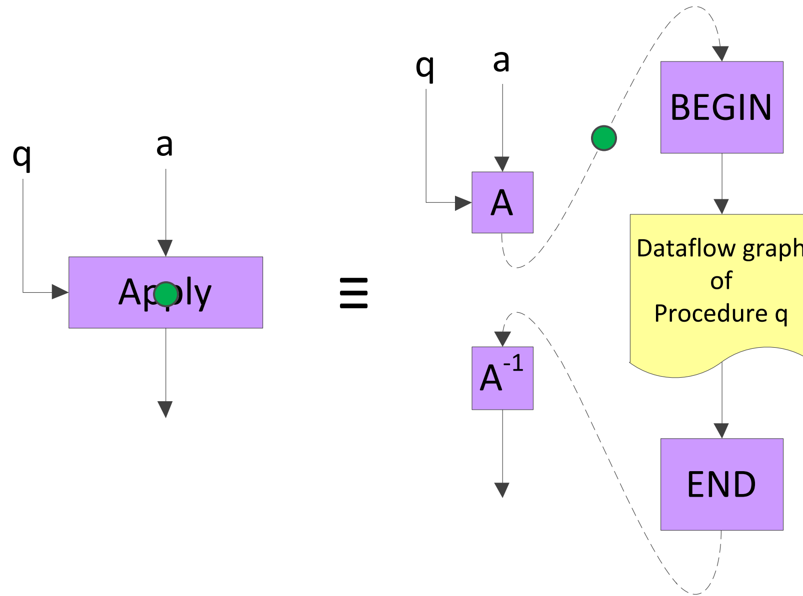
DYNAMIC DATAFLOW

THE APPLY OPERATOR



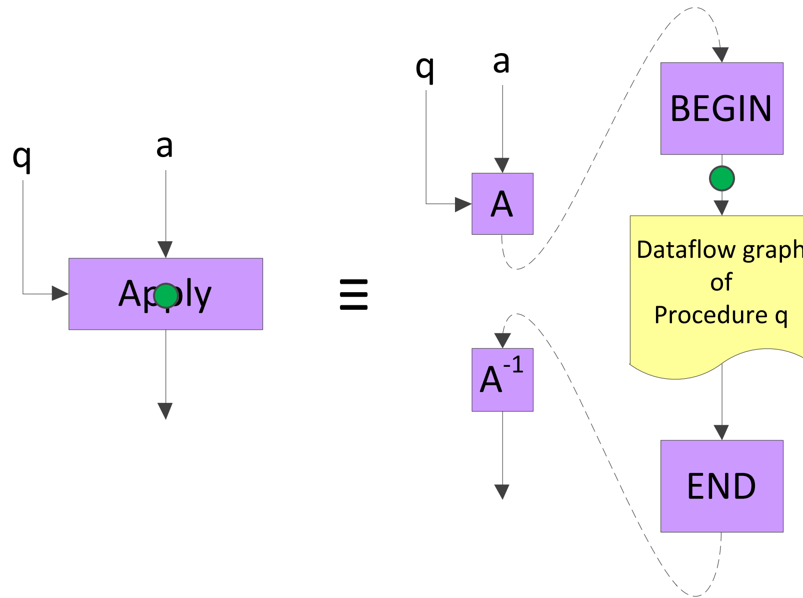
DYNAMIC DATAFLOW

THE APPLY OPERATOR



DYNAMIC DATAFLOW

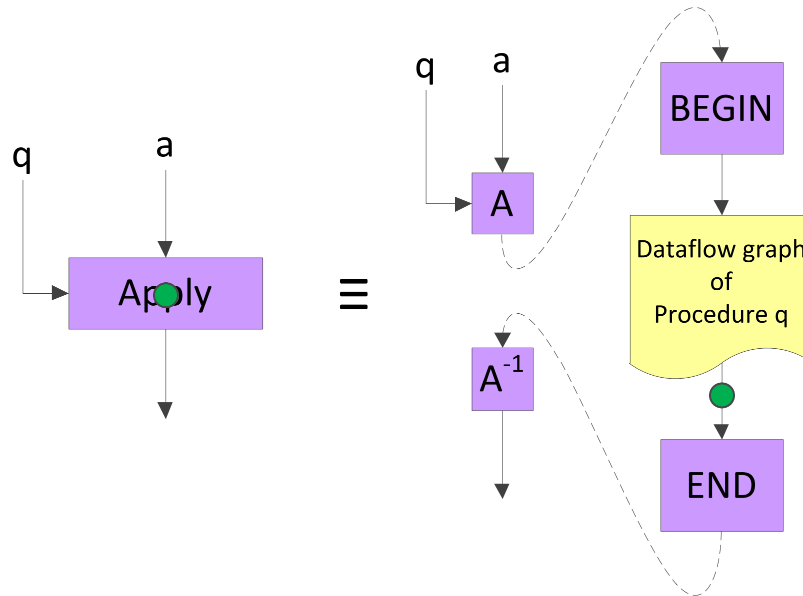
THE APPLY OPERATOR



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

DYNAMIC DATAFLOW

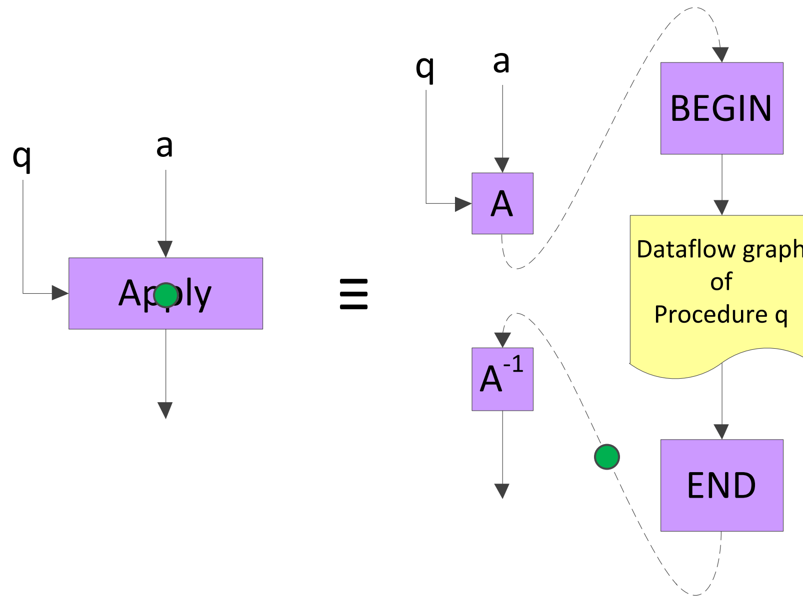
THE APPLY OPERATOR



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

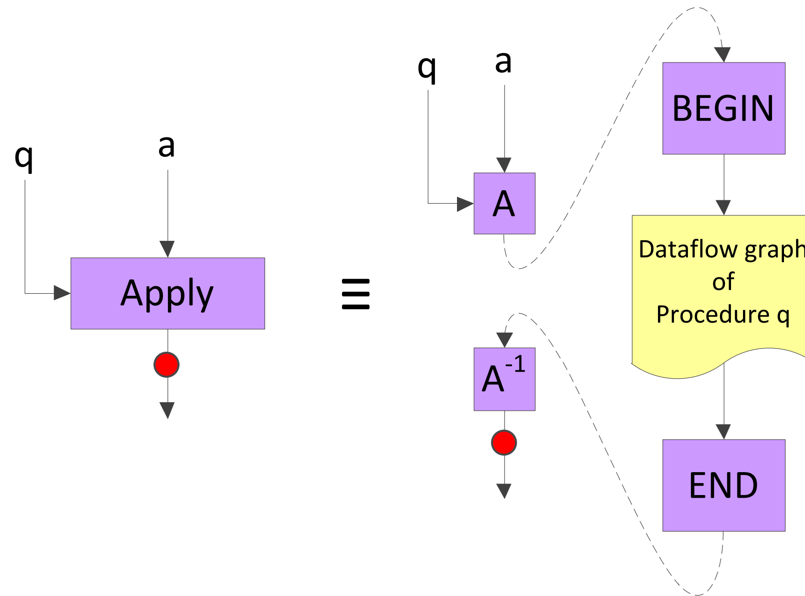
DYNAMIC DATAFLOW

THE APPLY OPERATOR



DYNAMIC DATAFLOW

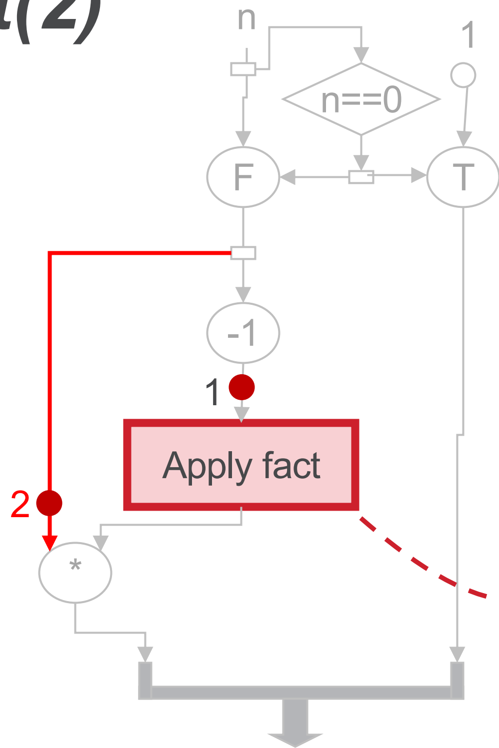
THE APPLY OPERATOR



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

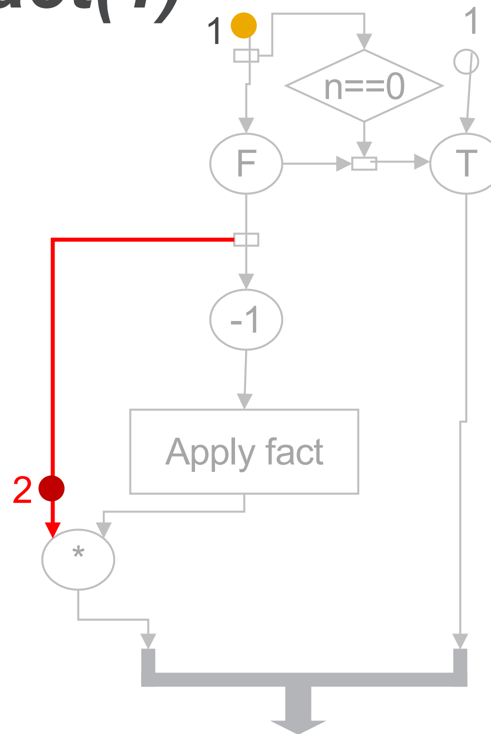
FACTORIAL REENTRY PROBLEM

fact(2)



Color change

fact(1)



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

MIT TAGGED TOKEN

 IEEE TRANSACTIONS ON COMPUTERS, VOL. 39, NO. 3, MARCH 1990

Executing a Program on the MIT Tagged-Token Dataflow Architecture

ARVIND, SENIOR MEMBER, IEEE, AND RISHIYUR S. NIKHIL, MEMBER, IEEE

Abstract—**The MIT Tagged-Token Dataflow project has an unconventional, but integrated approach to general-purpose high-performance parallel computing. Rather than extending conventional sequential languages, we use Id, a high-level language with fine-grained parallelism and determinacy implicit in its operational semantics. Id programs are compiled to dynamic dataflow graphs, a parallel machine language. Dataflow graphs are directly executed on the MIT Tagged-Token Dataflow Architecture (TTDA), a novel multiprocessor architecture. Dataflow research has advanced significantly in the last few years; in this paper, we provide an overview of our current thinking, by describing example Id programs, their compilation to dataflow graphs, and their execution on the TTDA. Finally, we describe related work and the status of our project.**

Index Terms—**Dataflow architectures, dataflow graphs, functional languages, implicit parallelism, *I*-structures, MIMD machines.**

I. Introduction

THERE are several commercial and research efforts currently underway to build parallel computers with performance far beyond what is possible today. Among those approaches that can be classified as general-purpose, "multiple instruction multiple data" (MIMD) machines, most are evolutionary in nature. For architectures, they employ interconnections of conventional von Neumann machines. For programming, they rely upon conventional sequential languages (such as Fortran, C, or Lisp) extended with some parallel primitives, often implemented using operating system calls. These extensions are necessary because the automatic detection of adequate parallelism remains a difficult problem, in spite of recent advances in compiler technology [28], [2], [35].

Unfortunately, a traditional von Neumann processor has fundamental characteristics that reduce its effectiveness in a parallel machine. First, its performance suffers in the presence of long memory and communication latencies, and these are unavoidable in a parallel machine. Second, they do not provide good synchronization mechanisms for frequent task switching between parallel activities, again inevitable in a parallel machine. Our detailed technical examination of these issues may be found in [11]. In [25], Iannucci explores architectural changes to remedy these problems, inspired by dataflow architectures.

Furthermore, traditional programming languages are not easily extended to incorporate parallelism. First, loss of determinacy adds significant complexity to establishing correctness (this includes debugging). Second, it is a significant added complication for the programmer to manage parallelism explicitly—to identify and schedule parallel tasks small enough to utilize the machine effectively but large enough to keep the resource-management overheads reasonable.

In contrast, our dataflow approach is quite unconventional. We begin with *Id*, a high-level language with fine-grained parallelism *implicit* in its operational semantics. Despite this potential for enormous parallelism, the semantics are also *determinate*. Programs in Id are compiled into *dataflow graphs*, which constitute a parallel machine language. Finally, dataflow graphs are executed directly on the *Tagged-Token Dataflow Architecture* (TTDA), a machine with purely data-driven instruction scheduling, unlike the sequential program counter-based scheduling of von Neumann machines.

Dataflow research has made great strides since the seminal paper on dataflow graphs by Dennis [18]. Major milestones have been: the *U*-Interpreter for dynamic dataflow graphs [9], the first version of Id [10], the Manchester Dataflow machine [22] and, most recently, the ETL Sigma-1 in Japan [48], [23]. But much has happened since then at all levels—language, compiling, and architecture—and dataflow, not being a mainstream approach, requires some demystification. In this paper, we provide an accurate snapshot as of early 1987, by providing a fairly detailed explanation of the compilation and execution of an Id program. Because of the expanse of topics, our coverage of neither the language and compiler nor the architecture can be comprehensive; we provide pointers to relevant literature for the interested reader.

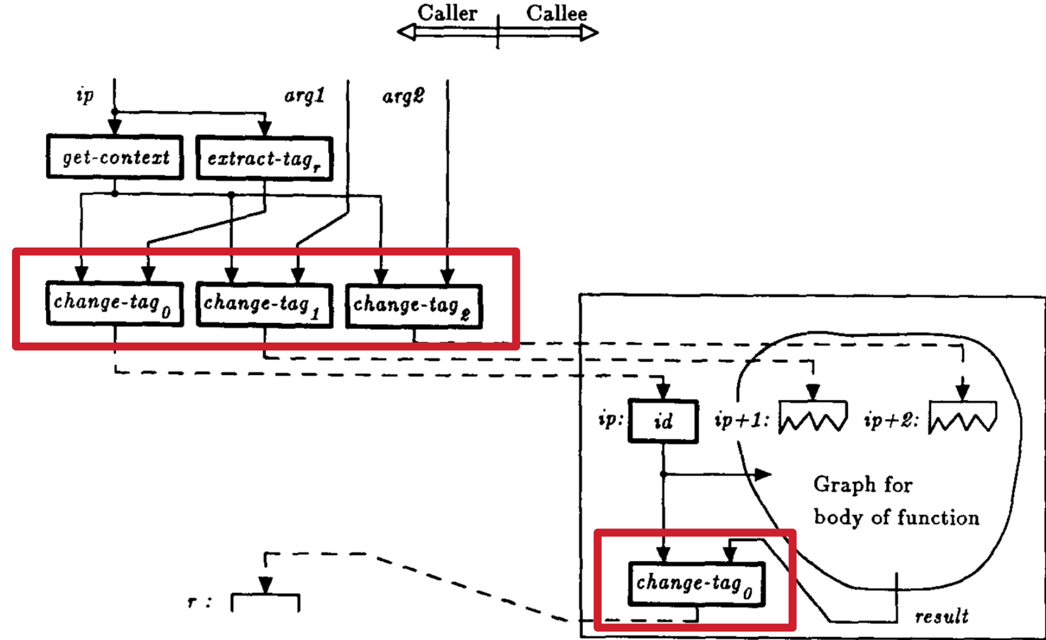
In Section II, we present example programs expressed in Id, our high-level parallel language. We take the opportunity to explain the parallelism in Id, and to state our philosophy about parallel languages in general. In Section III, we explain dataflow graphs as a parallel machine language and show how to compile the example programs. In Section IV, we describe the MIT Tagged-Token Dataflow Architecture and show how to encode and execute dataflow graphs. Finally, in Section V

Manuscript received August 5, 1987; revised February 3, 1989. This work was done at MIT Laboratory for Computer Science. This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research Contract N00014-84-K-0099. An early version of this paper appeared in the *Proceeding of the PARLE Conference*, Eindhoven, The Netherland, Springer-Verlag LNCS Vol. 259, June 1987.

The authors are with the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

IEEE Log Number 8932907.

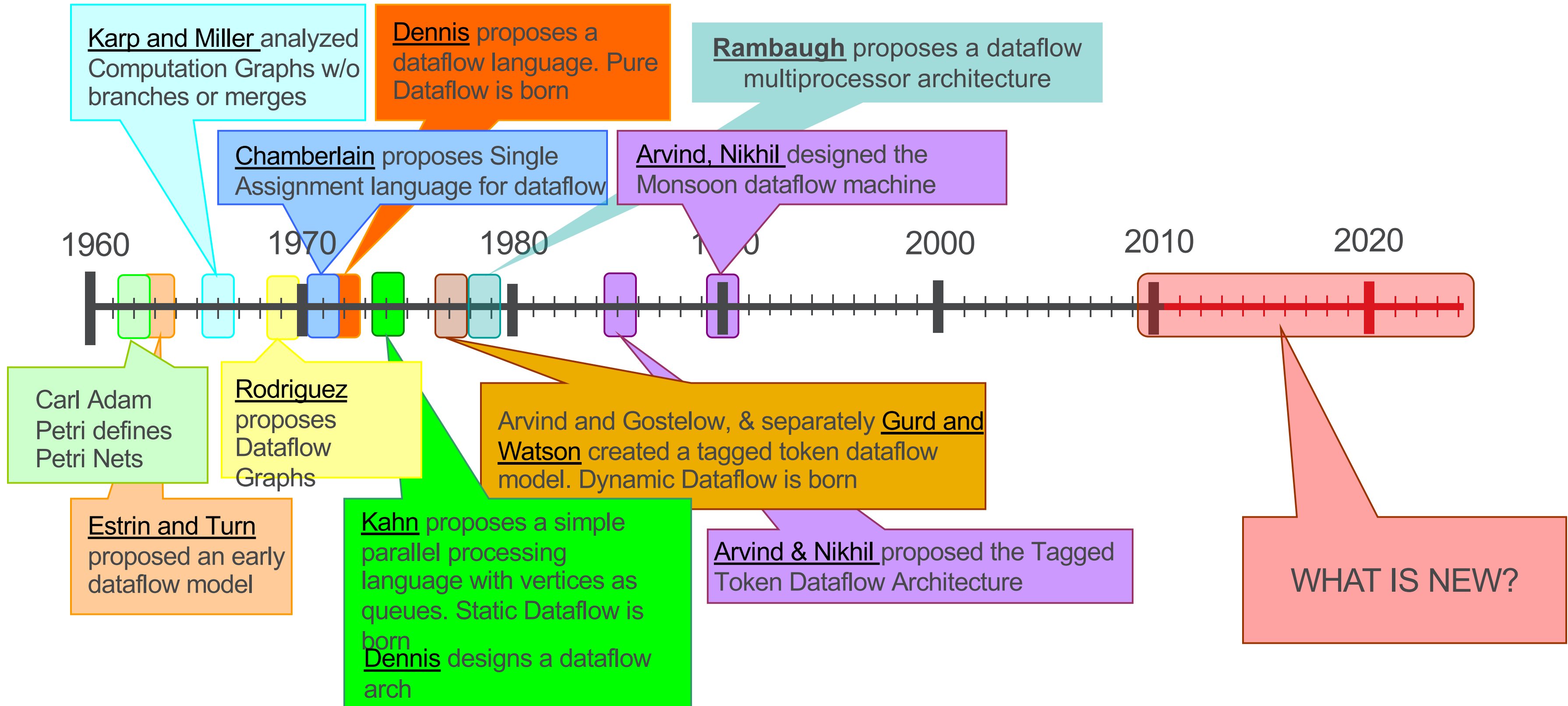
0018-9340/90/0300-0300$01.00 © 1990 IEEE



U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

MODERN DATAFLOW ARCHITECTURES

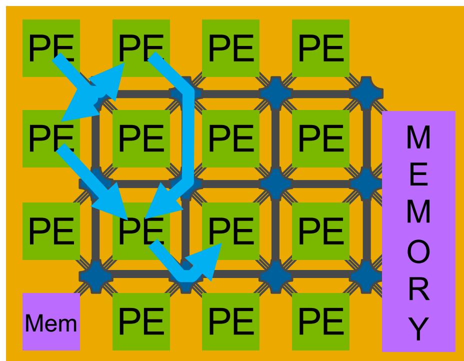
BRIEF HISTORY OF DATAFLOW

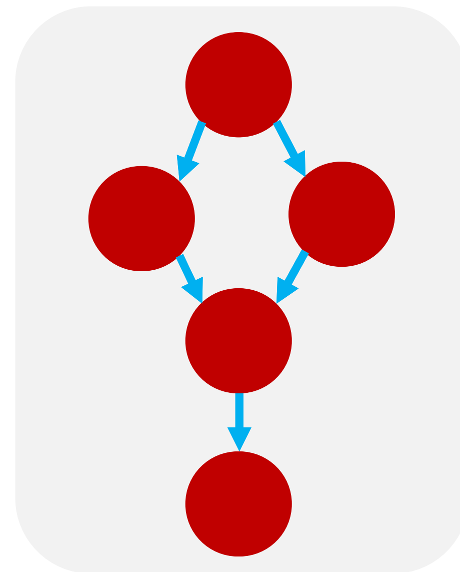


U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

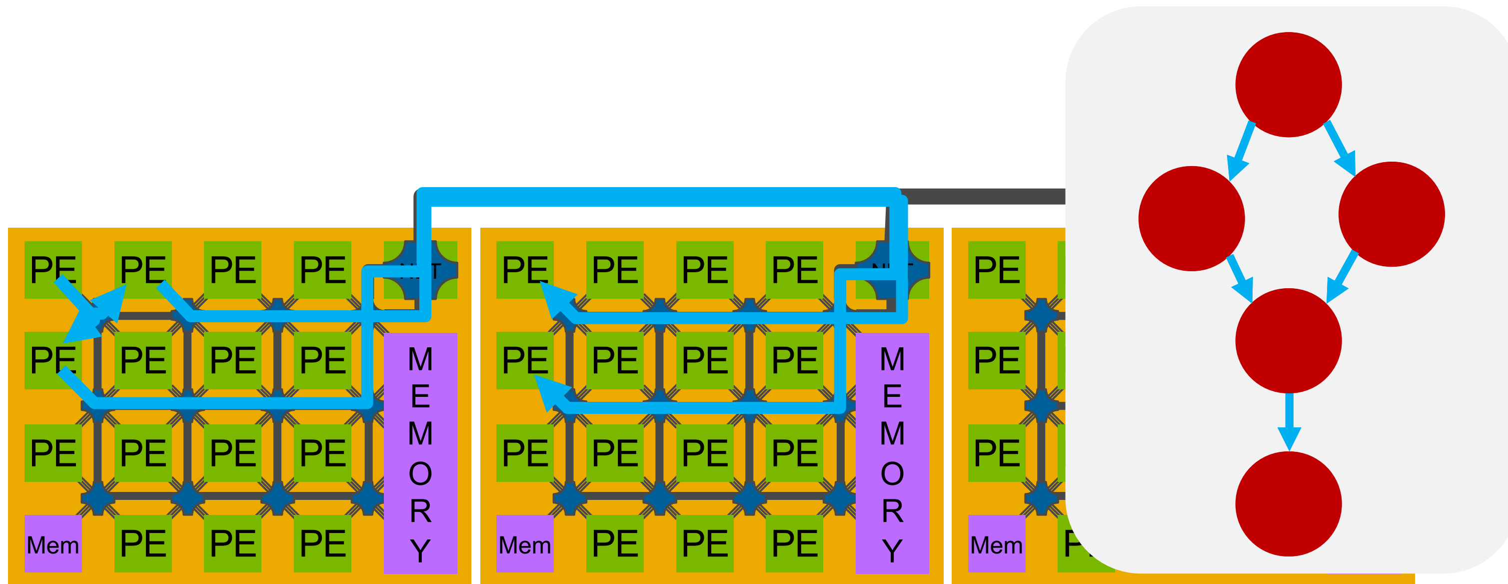
SPATIAL RECONFIGURABLE ARCHITECTURES

1. Granularity of dataflow operations
2. Spatial reconfigurable architectures





SPATIAL RECONFIGURABLE ARCHITECTURES

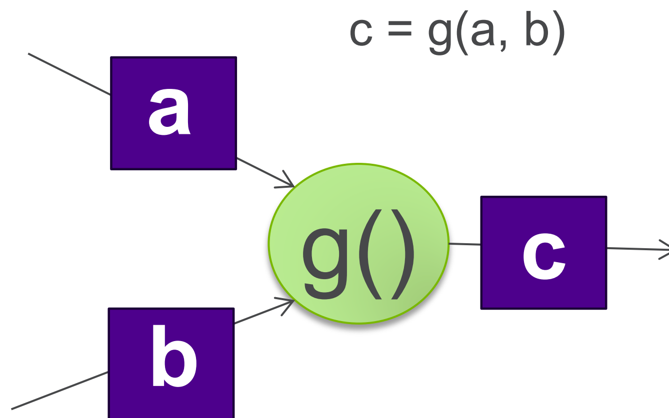


COARSE GRAIN DATAFLOW

Dataflow graph node size

Dataflow graph node executes more complex mathematical operations

- Hybrid execution models
  - Von Neumann + Dataflow
- Node g() is defined in terms of multiple instructions
- Data locations (i.e., tokens) a, b, and c have a larger memory footprint



SOME EXAMPLES OF CURRENT DATAFLOW ARCHITECTURES

Overview of Modern Spatial Architetures

Cerebras CS-2

SambaNova DataScale SN30

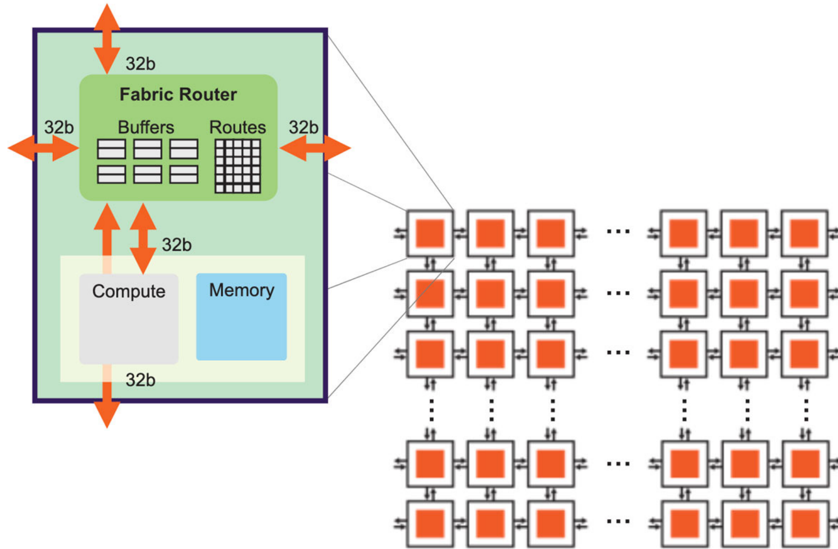
Habana Gaudi1

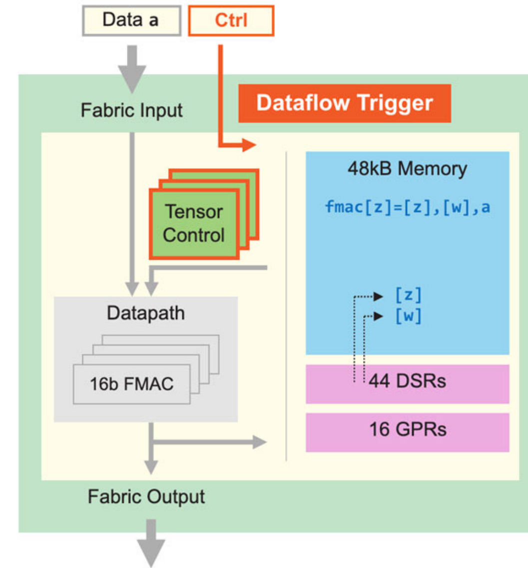
GroqRack

	Cerebras CS2	SambaNova Cardinal SN30	Groq GroqRack	Habana Gaudi1
Compute Units	850,000 Cores	640 PCUs	5120 vector ALUs	8 TPC + GEMM engine
On-Chip Memory	40 GB L1, 1TB+ MemoryX	>300MB L1 1TB	230MB L1	24 MB L1 32GB
Process	7nm	7nm	7 nm	7nm
System Size	2 Nodes including Memory-X and Swarm-X	8 nodes (8 cards per node)	9 nodes (8 cards per node)	2 nodes (8 cards per node)
Estimated Performance of a card (TFlops)	>5780 (FP16)	>660 (BF16)	>250 (FP16) >1000 (INT8)	>150 (FP16)
Software Stack Support	Tensorflow, Pytorch, CSLang	SambaFlow, Pytorch, C++ SDK	GroqAPI, ONNX, C/C++ Groq Runtime	Synapse AI, TensorFlow and PyTorch, TPC
Interconnect	Ethernet-based	Ethernet-based	RealScale TM	Ethernet-based

https://www.alcf.anl.gov/alcf-ai-testbed

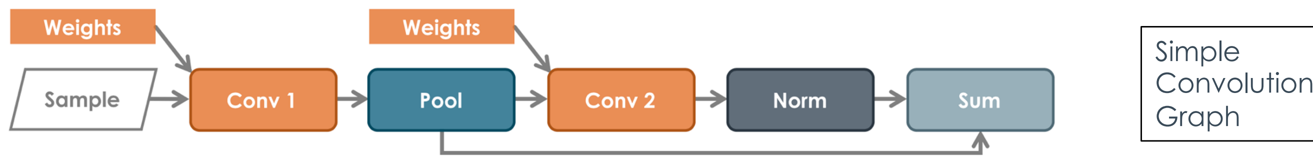
CEREBRAS



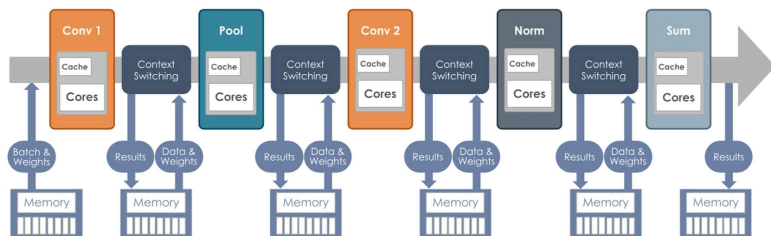


S. Lie, "Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning," in IEEE Micro, vol. 43, no. 3, pp. 18-30, May-June 2023, doi: 10.1109/MM.2023.3256384.

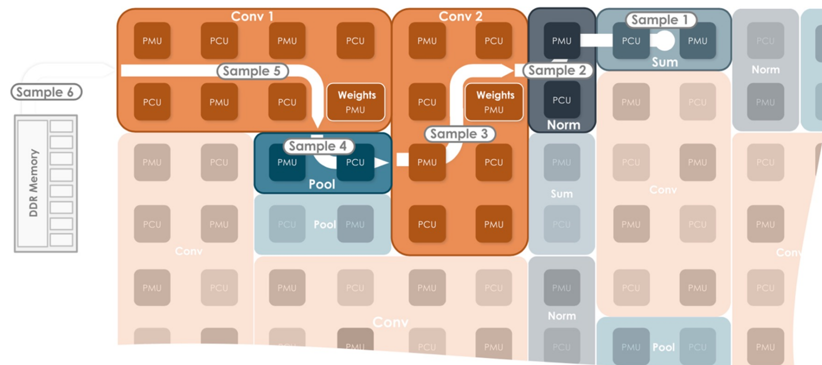
SAMBANOVA



Simple Convolution Graph



GPU accelerators: Each kernel is launched onto the device and bottlenecks include memory bandwidth and kernel-launch latencies



Dataflow: Kernels are spatially mapped onto the accelerator and data flows on-chip between them reducing memory traffic

Image Courtesy: Sumti Jairath, SambaNova

HANDS ON WITH AI ACCELERATORS

Cerebras CS-2 Wafer-Scale Cluster WSE-2

SambaNova DataScale SN30

GRAPHCORE

Graphcore Bow Pod64

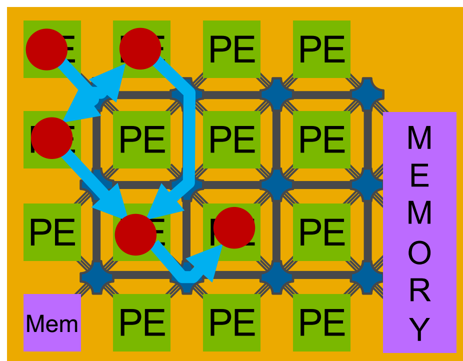
Track 8 – Machine Learning
Introductions to AI Testbed at ALCF and Hands-on
3.30-5.00 PM, August 11
Siddhisanket (Sid) Raskar

DATAFLOW CHALLENGES

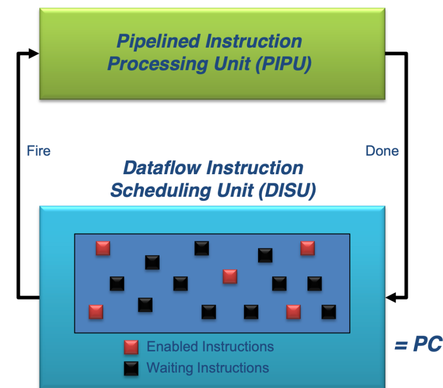
Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

Argonne
NATIONAL LABORATORY

SPATIAL DATAFLOW SCHEDULING VS ARGUMENT FETCHING



NP-Hard problem
(Factorial Complexity)



Runtime scheduling
decision overhead

RECOMMENDATION

Two different points of view

1985

Two Fundamental Issues in Multiprocessing

Arvind
Robert A. Iannucci

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139 - USA

Abstract

A general purpose multiprocessor should be scalable, *i.e.* show higher performance when more hardware resources are added to the machine. Architects of such multiprocessors must address the loss in processor efficiency due to two fundamental issues: long memory latencies and waits due to synchronization events. It is argued that a well designed processor can overcome these losses provided there is sufficient parallelism in the program being executed. The detrimental effect of long latency can be reduced by instruction pipelining, however, the restriction of a single thread of computation in von Neumann processors severely limits their ability to have more than a few instructions in the pipeline. Furthermore, techniques to reduce the memory latency tend to increase the cost of task switching. The cost of synchronization events in von Neumann machines makes decomposing a program into very small tasks counter-productive. Dataflow machines, on the other hand, treat each instruction as a task, and by paying a small synchronization cost for each instruction executed, offer the ultimate flexibility in scheduling instructions to reduce processor idle time.

Key words and phrases: caches, cache coherence, dataflow architectures, hazard resolution, instruction pipelining, LOAD/STORE architectures, memory latency, multiprocessors, multi-thread architectures, semaphores, synchronization, von Neumann architecture.

1993

Two Fundamental Limits on Dataflow Multiprocessing*

David E. Culler
Klaus Erik Schauser
Thorsten von Eicken

Report No. UCB/CSD 92/716
Computer Science Division
University of California, Berkeley

Abstract: This paper examines the argument for dataflow architectures in "Two Fundamental Issues in Multiprocessing[5]." We observe two key problems. First, the justification of extensive multithreading is based on an overly simplistic view of the storage hierarchy. Second, the local greedy scheduling policy embodied in dataflow is inadequate in many circumstances. A more realistic model of the storage hierarchy imposes significant constraints on the scheduling of computation and requires a degree of parsimony in the scheduling policy. In particular, it is important to establish a scheduling hierarchy that reflects the underlying storage hierarchy. However, even with this improvement, simple local scheduling policies are unlikely to be adequate.

Keywords: dataflow, multiprocessing, multithreading, latency tolerance, storage hierarchy, scheduling hierarchy.

1 Introduction

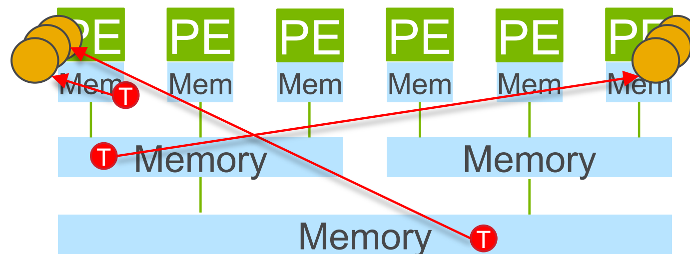
The advantages of dataflow architectures were argued persuasively in a seminal 1983 paper by Arvind and Iannucci[4] and in a 1987 revision entitled "Two Fundamental Issues in Multiprocessing"[5]. However, reality has proved less favorable to this approach than their arguments would suggest. This motivates us to examine the line of reasoning that has driven dataflow architectures and fine-grain multithreading to understand where the argument went awry. We observe two key problems. First, the justification of extensive multithreading is based on an overly simplistic view of the storage hierarchy. Second, the local greedy scheduling policy embodied in dataflow, "execute whenever data is available" is inadequate in many circumstances. A more realistic model of the storage hierarchy imposes significant constraints on the scheduling of computation and requires a degree of parsimony in the scheduling policy. In particular, it is important to establish a scheduling hierarchy that reflects the underlying storage hierarchy. However, even with this improvement, simple local scheduling policies are unlikely to be adequate.

TWO FUNDAMENTAL LIMITS ON DATAFLOW MULTIPROCESSING

David E. Culler et. al.

- Limit 1: Storage Hierarchy

"Dataflow architectures essentially replace the small register number with a large tag that serves to "name" the value. A realistic view of the storage hierarchy requires that only a small number of such name/value pairs can be resident at a time. Once the number of VPs exceeds the capacity of the top level matching store, the synchronization cost increases dramatically, since some form of overflow store must be used."



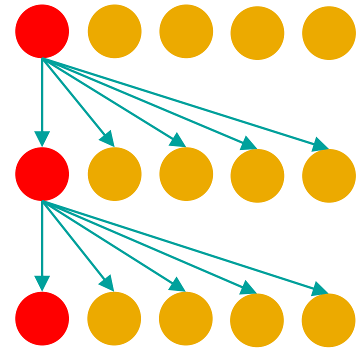
U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

TWO FUNDAMENTAL LIMITS ON DATAFLOW MULTIPROCESSING

David E. Culler et. al.

- Limit 2: Local Dynamic Scheduling

"…any naïve local scheduling policy exhibits unnecessarily low machine efficiency or high resource requirements on some programs. Thus, it would seem unwise to rely solely on low-level hardware mechanisms or runtime system support to determine the scheduling of computation."

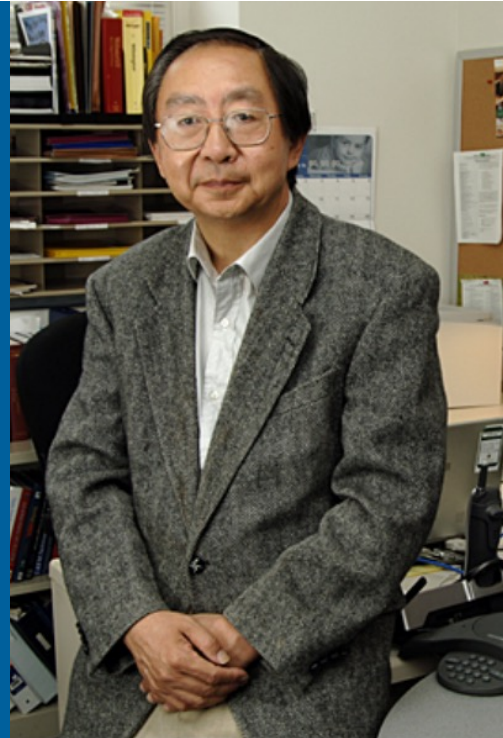


CHALLENGES

- Memory management
- Resource allocation and scheduling
- Parallelism control vs locality
- Balancing "size" of each dataflow node
- Programmability
- …

Argonne NATIONAL LABORATORY

THANK YOU



IN LOVING MEMORY OF

Professor Guang R Gao 1945-2021

Let his legacy be remembered and his impact persist forever