Dataflow asynchronous design and pipeline performance

Benjamin Hill

benjamin.hill@intel.com

ASYNC Summer School 2022

Legal Information

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Your costs and results may vary.

Results have been estimated or simulated

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Asynchronous circuits

- At a behavioral level, only dipping down to explain motivation for important concepts (we’ll get to lower-level details next week)
- Token = [DATA] + VALIDITY + FLOW CONTROL
- Processes communicate on Channels by exchanging Tokens
- Parallel (processes) is free; sequencing is engineered/expensive

Dataflow computation

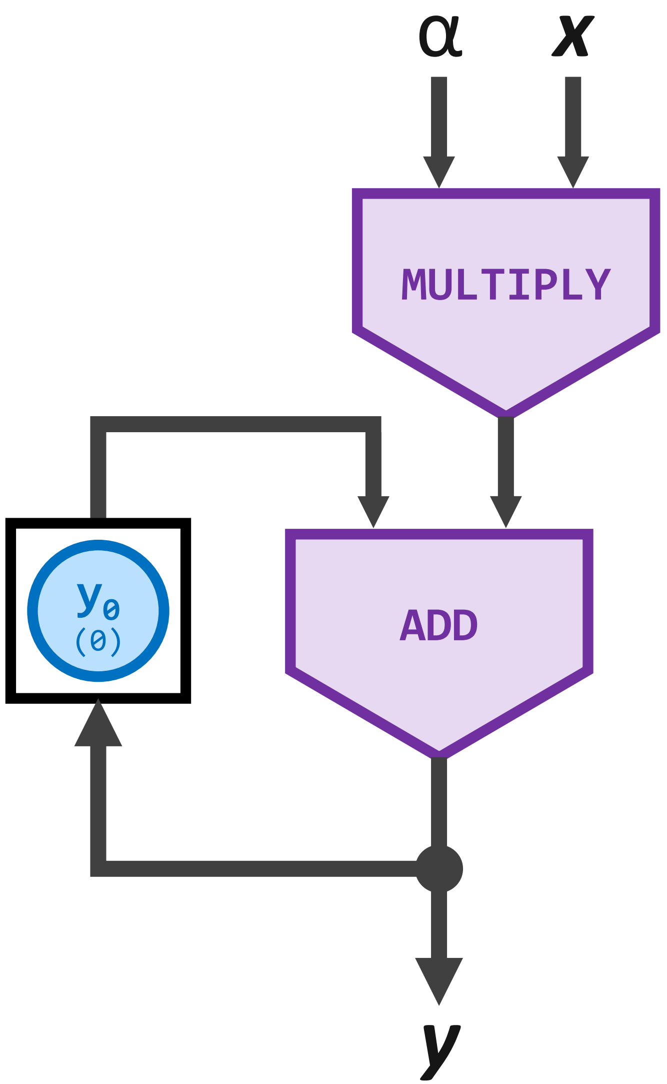
- Structural, graphical way of describing computation
- Useful abstraction, intuitive way to convey design intent
- Not necessarily asynchronous, though some key advantages as a natural mapping

Example: multiply-accumulate

Motivation: linear algebra core operation

$$\boldsymbol{y} \leftarrow \alpha\boldsymbol{x} + \boldsymbol{y} \qquad \text{(SAXPY)}$$

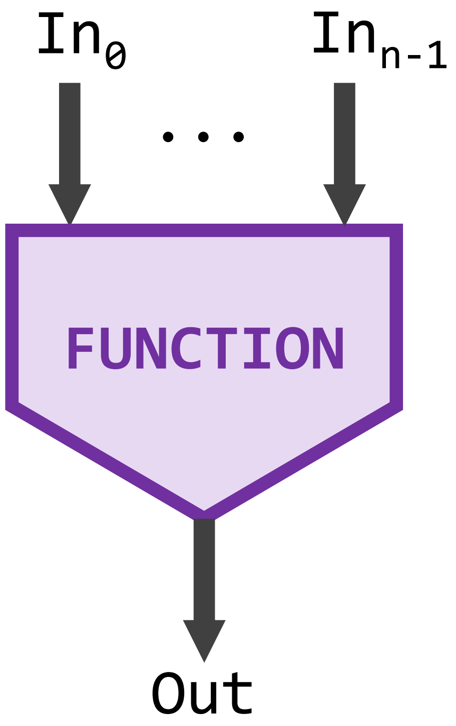
If you care about DSP, HPC, AI/deep learning… this is a useful kernel to implement



FUNCTION

Read values from all inputs, compute result and send on output

Example functions: arithmetic, logic, decoding, etc.



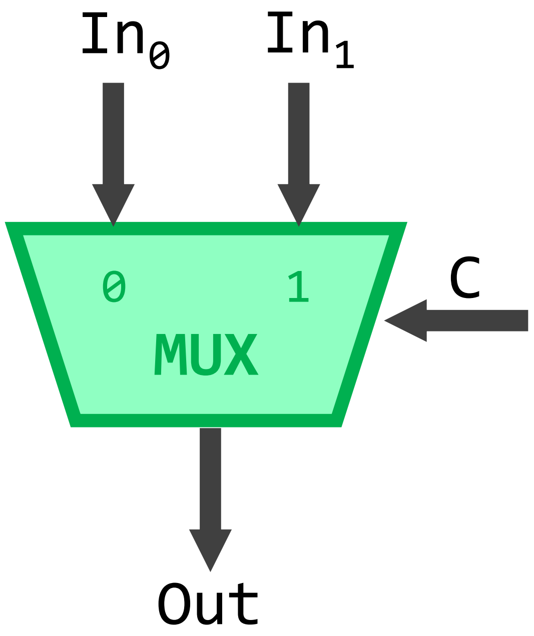

```
*[ In_0?arg_0, In_1?arg_1, … , In_n-1?arg_n-1;
   Out!func(arg_0,arg_1,…,arg_n-1)
 ]
```

Also known as: OPERATOR

Multiplexer (MUX)

Select one input to send to output
based on control signal; ignore other input

Not to be confused with combinational MUX:
same basic behavior, but this is a dataflow operator

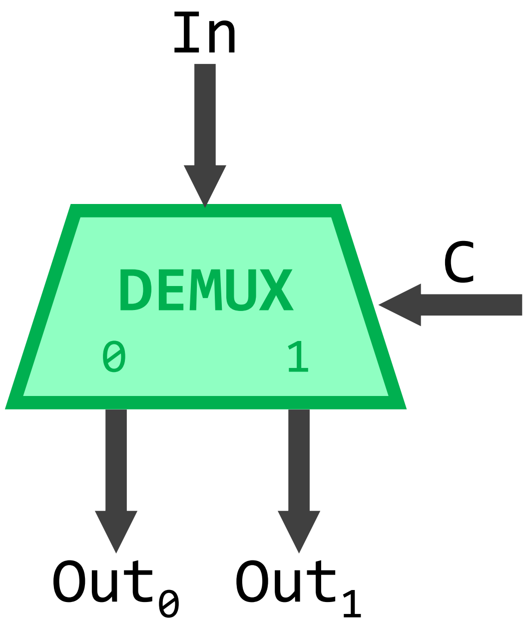



```
*[C?c;
   [  c=0 -> In0?x
   [] c=1 -> In1?x
   ];
  Out!x
 ]
```

Also known as: controlled merge, conditional join

DEMUX

Steer input to one of two outputs, based on value of control signal



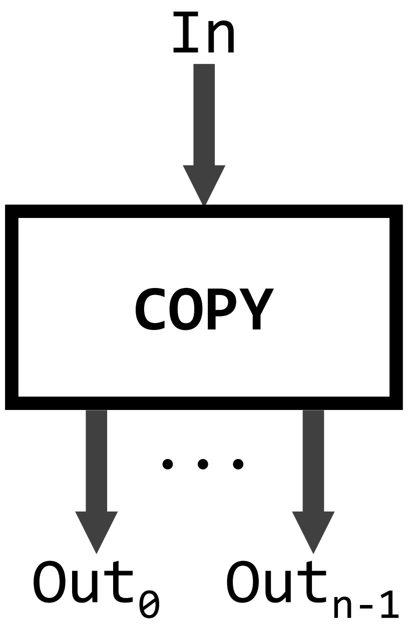

```
*[In?x, C?c;
   [  c=0 -> Out0!x
   [] c=1 -> Out1!x
   ]
 ]
```

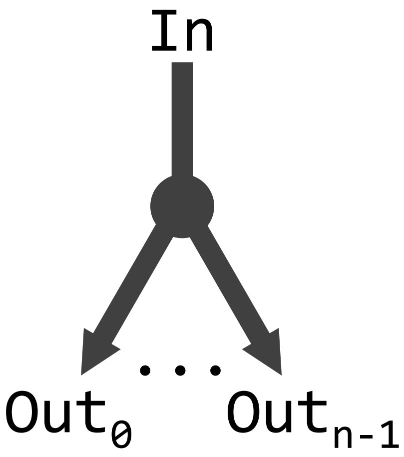
Also known as: SPLIT

COPY

Copy input token to multiple destinations

Often not drawn explicitly; all fan-out in dataflow graph requires a COPY





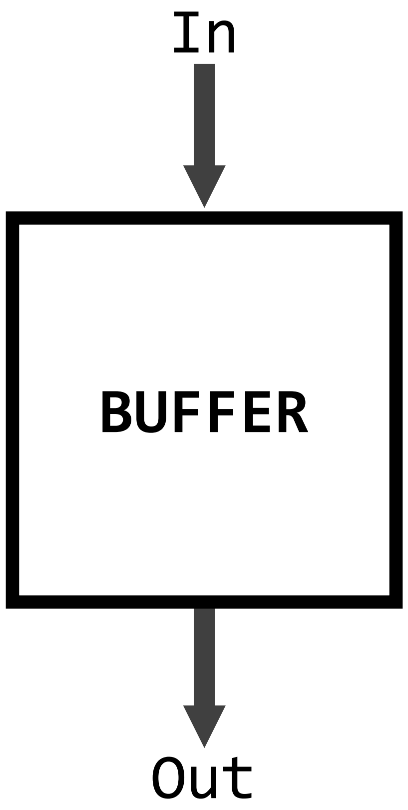
Also known as: FORK, n-way link

`*[In?x; Out_0!x, …, Out_{n-1}!x]`

BUFFER

Transmit token from input to output with storage and handshaking flow control

Important for performance, but often not drawn explicitly in static dataflow diagrams

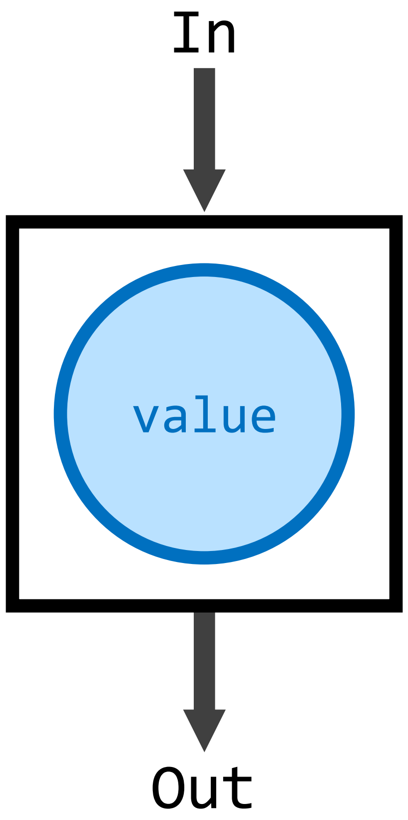


Also known as: slack buffer, one-place FIFO, latch

```
*[In?x; Out!x]
```

Initial token buffer

Send one initial value token,
then behave as a normal buffer

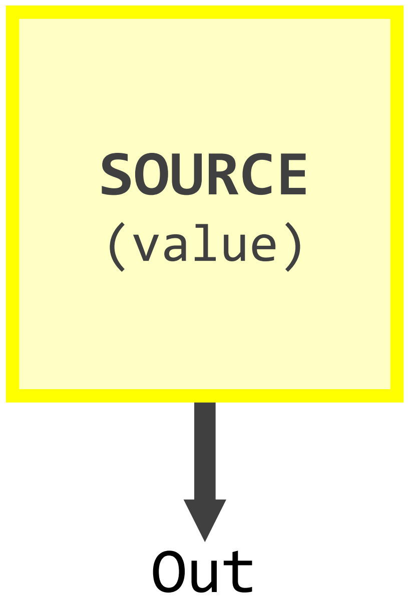


Also known as: INITIALIZER

```
Out!value; *[In?x; Out!x]
```

SOURCE

Repeatedly send tokens with same constant value



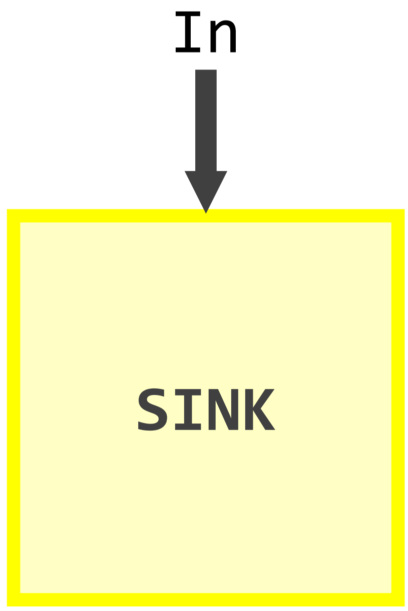
Also known as: bit/token generator

`*[Out!value]`

SINK

Consume and discard input token

Not particularly useful by itself, but in combination with other dataflow primitives



Also known as: (bit) bucket

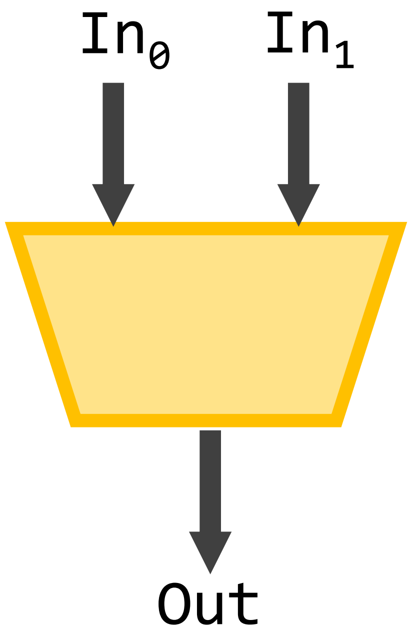
`*[In?value]`

Uncontrolled merge

Combine two input streams to one output

Depending on system design, selection is either:

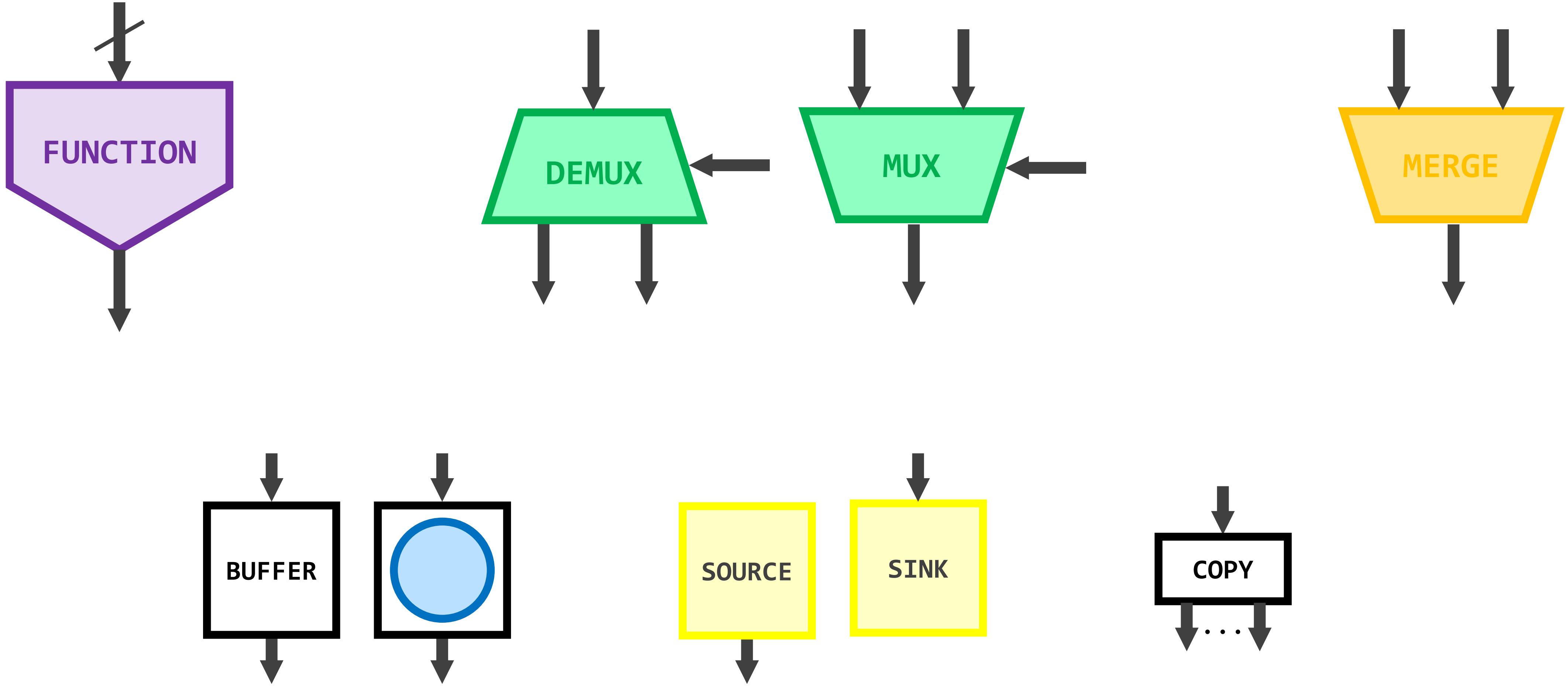
- **deterministic** – only one input will be used at a time
- **non-deterministic** – requires arbitration to choose



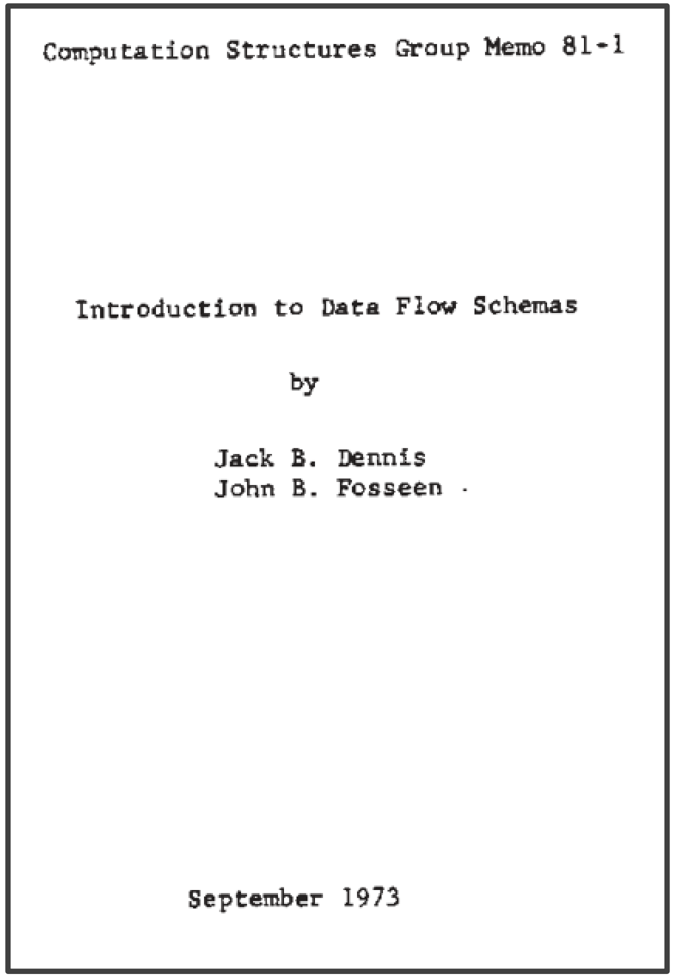

```
*[ [  #In₀ -> In₀?x
   [] #In₁ -> In₁?x
   ];
   Out!x
 ]
```

Also known as: MIXER, JOIN

Dataflow building blocks



Example: T-gate



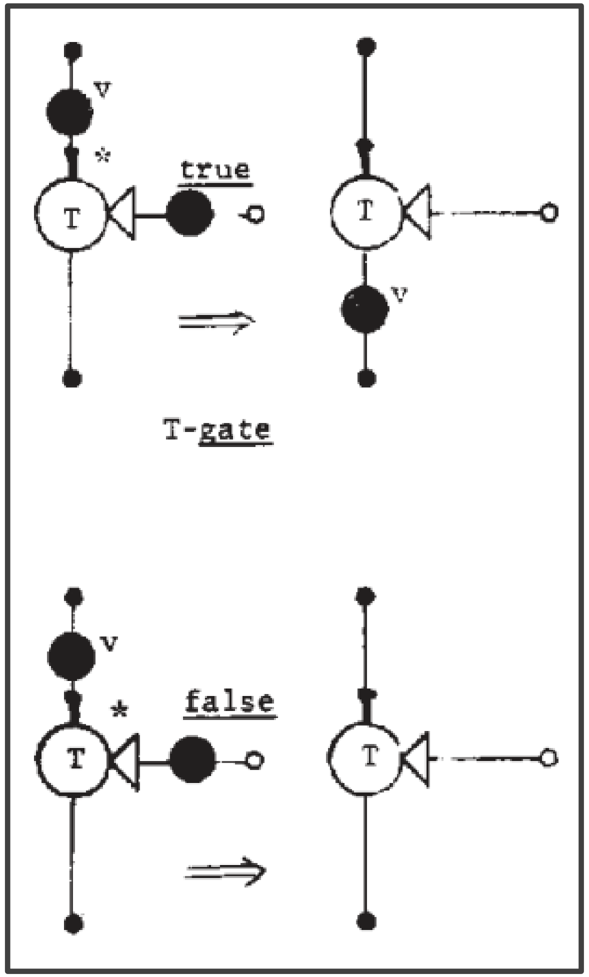
Computation Structures Group Memo 81-1

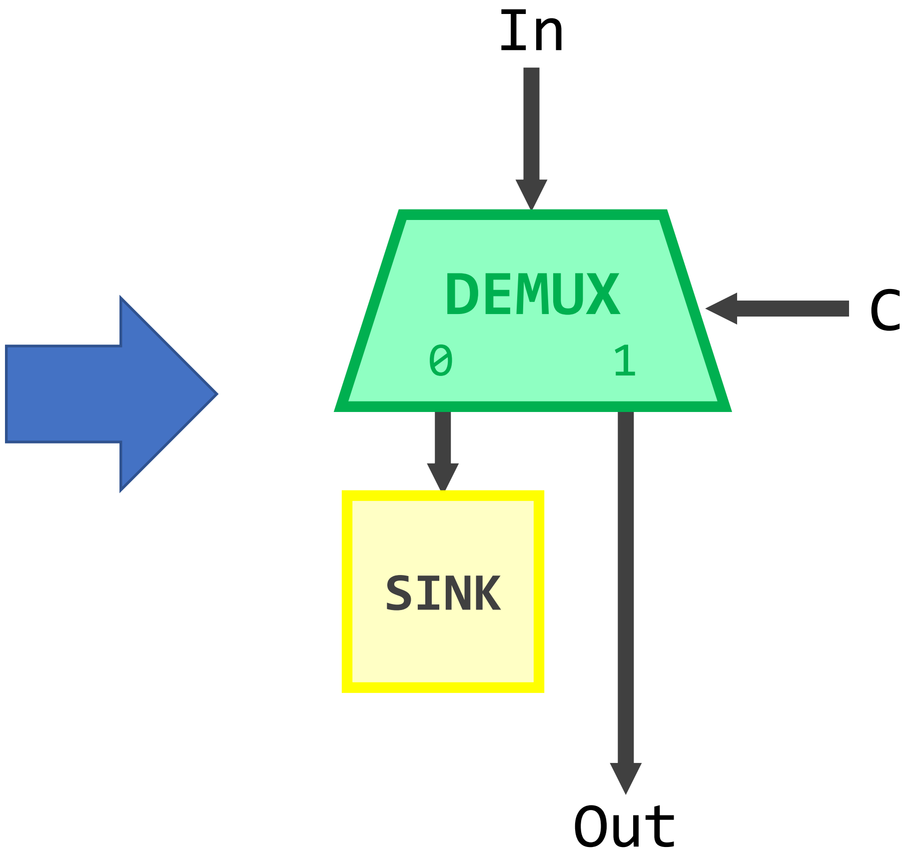
Introduction to Data Flow Schemas

by

Jack B. Dennis
John B. Fosseen

September 1973





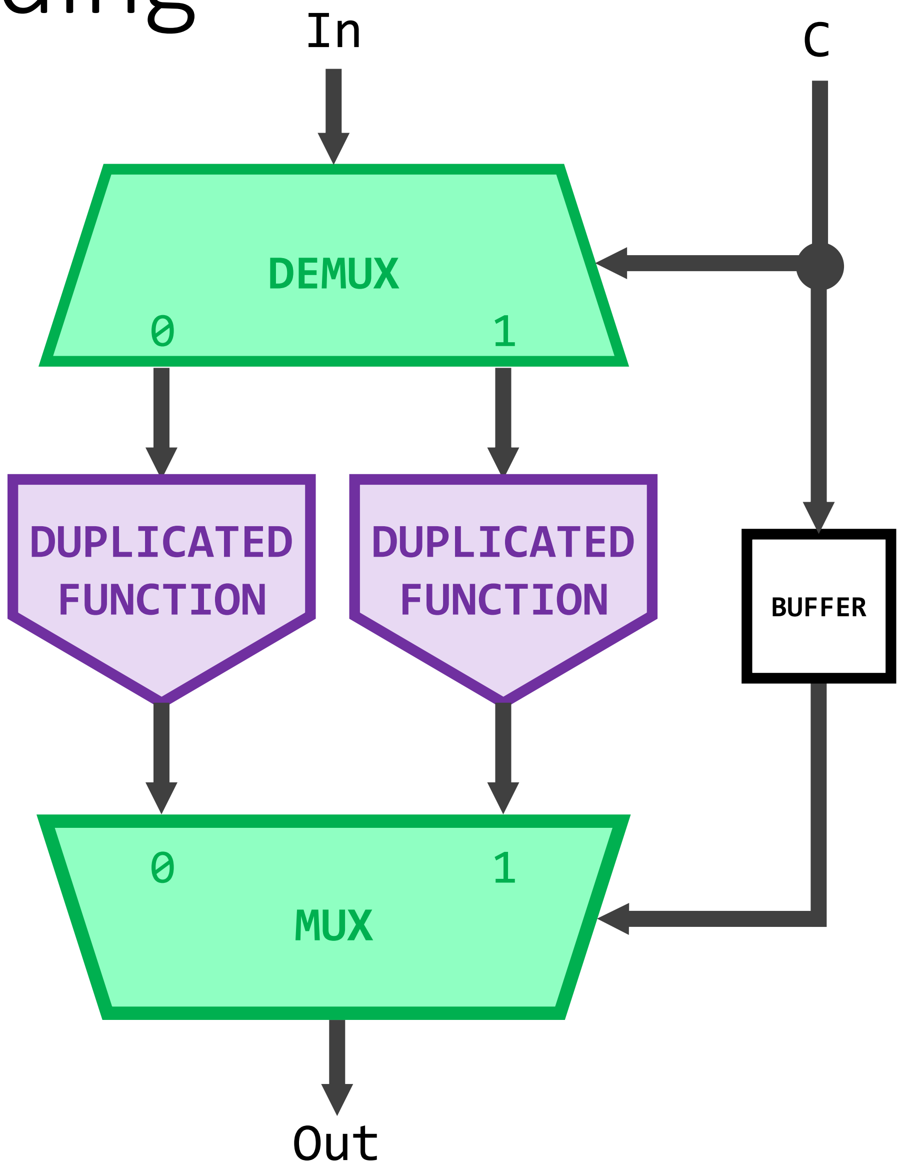
Transformation: "Multithreading"

Idea: replicate dataflow elements and interleave data between them

Improves throughput at the cost of area

Example: large arithmetic block where it is difficult to add internal pipelining

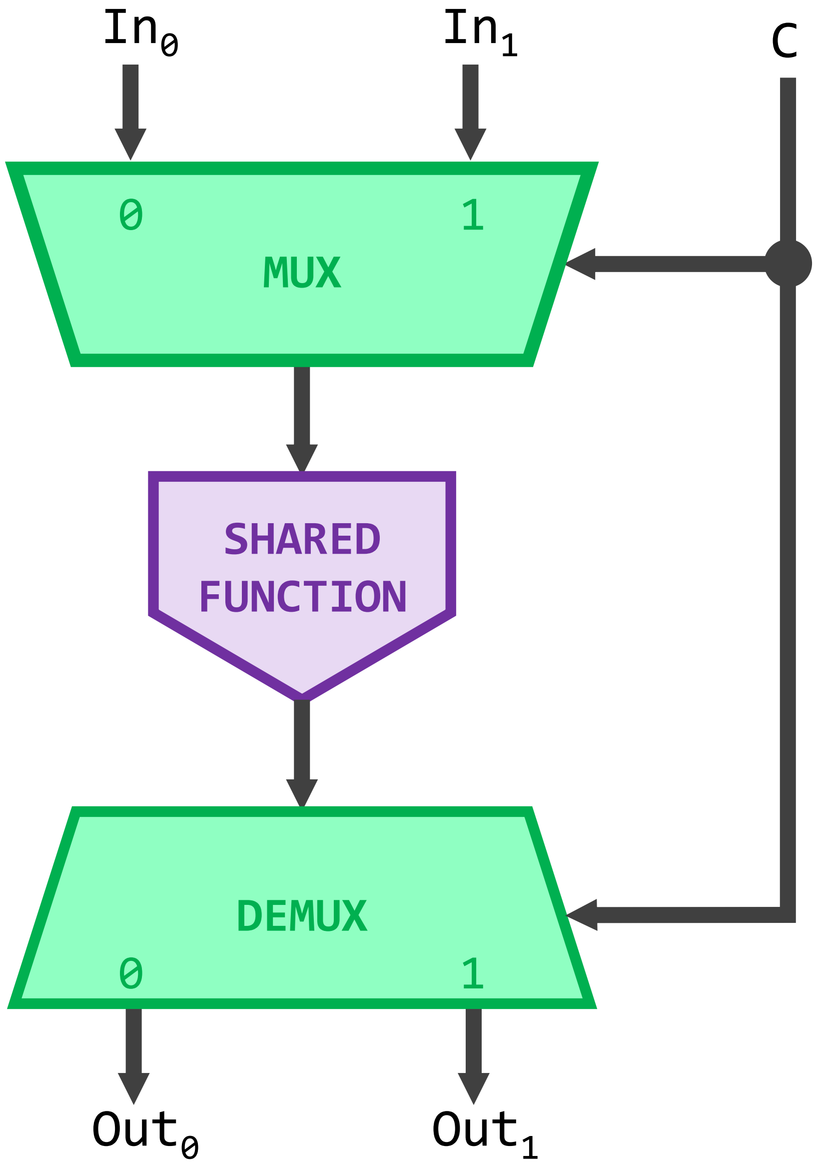
Not just for compute, could also be storage (e.g. tree FIFO)



Transformation: time sharing

Idea: share one expensive or unique resource between multiple users

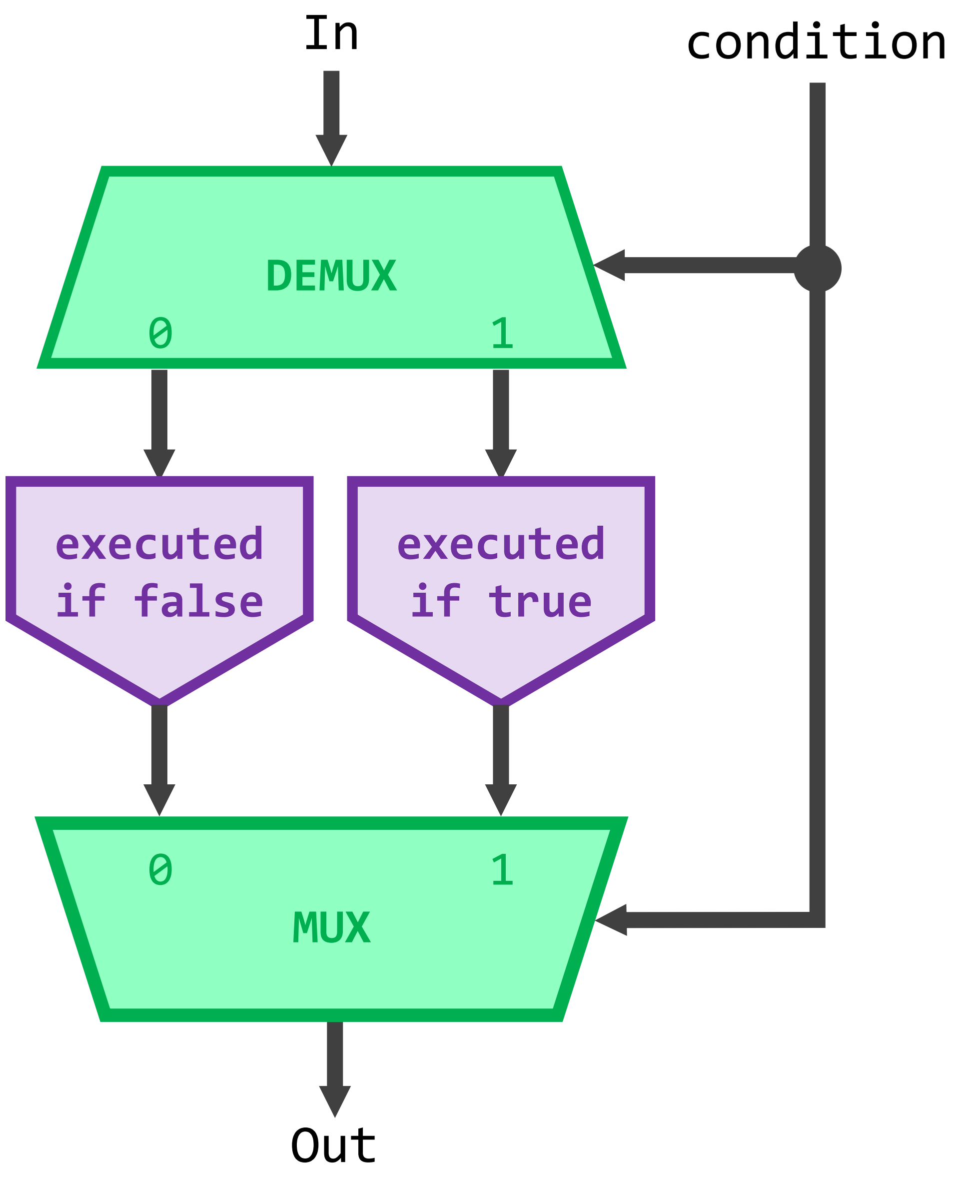
Improves area at the cost of throughput



Building block: IF statement

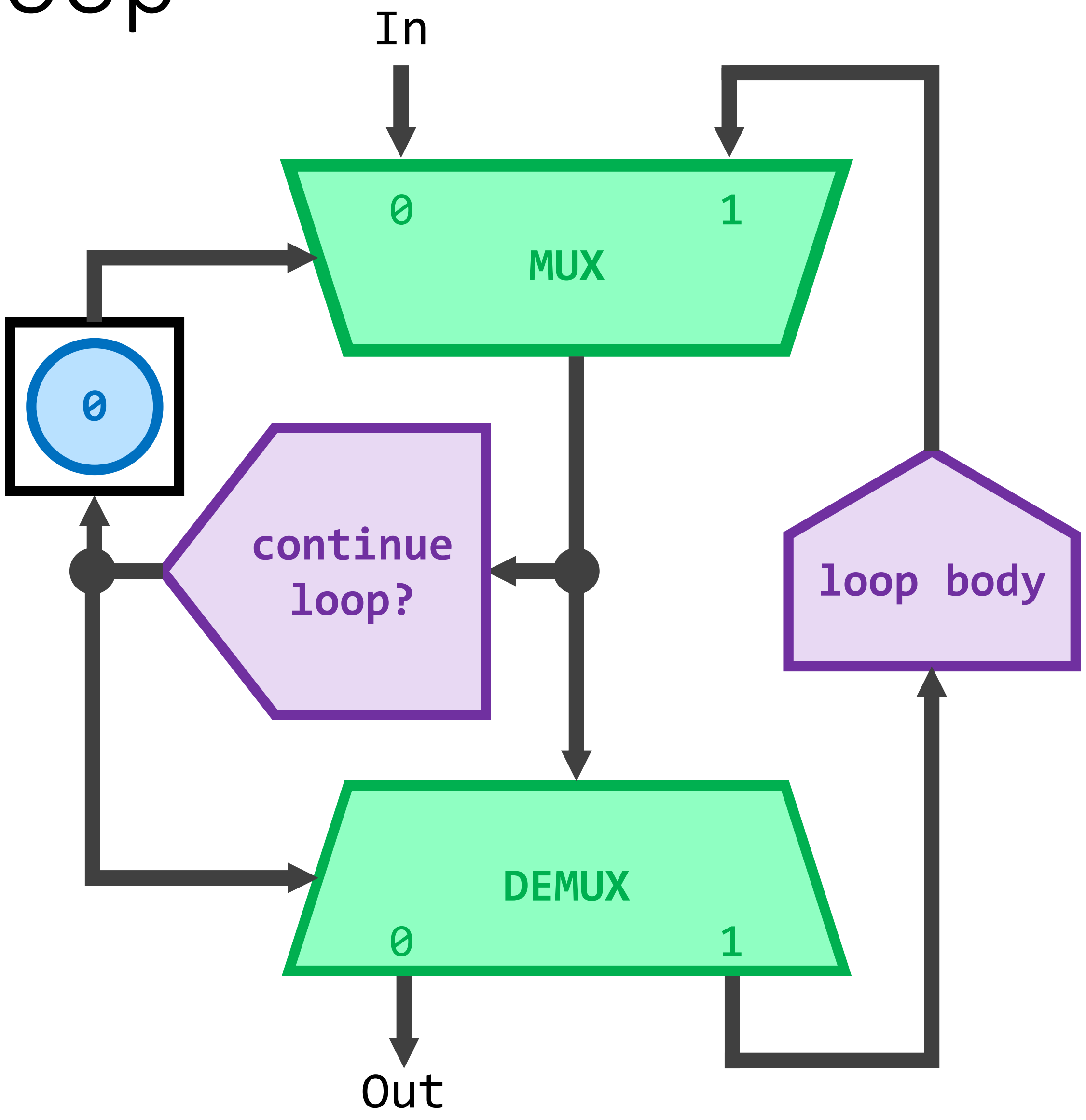
Useful for high-level synthesis

Shown with FUNCTION blocks but can also be other dataflow graphs (e.g. nested IF statements)



Building block: WHILE loop

Can also implement other loop constructs with a similar pattern

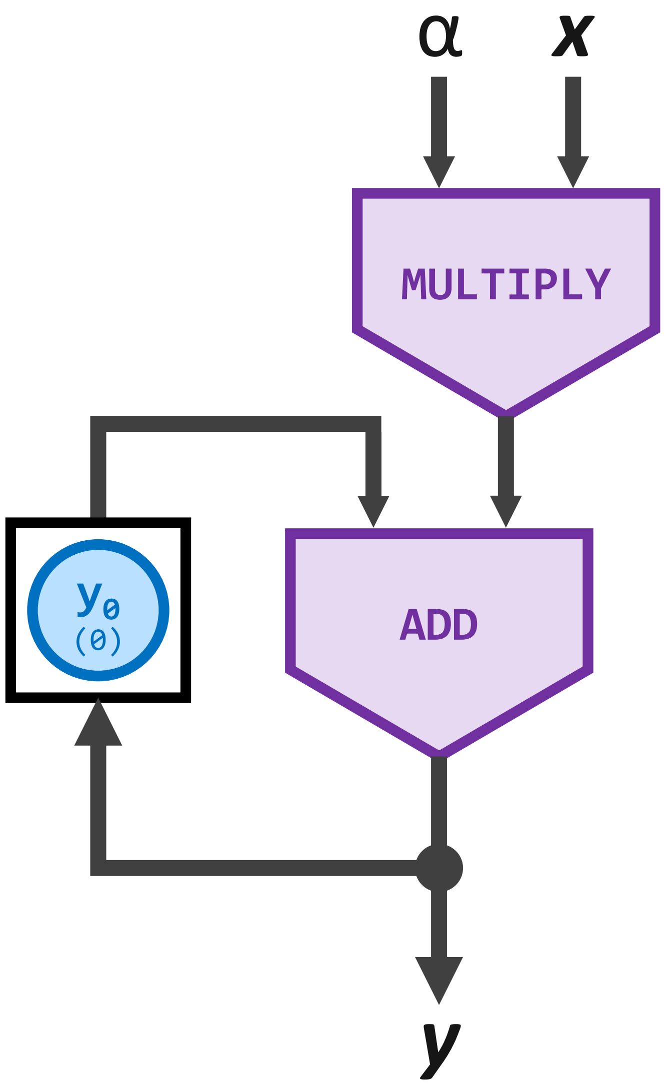


Multiply-accumulate revisited

Motivation: linear algebra core operation

$$\boldsymbol{y} \leftarrow \alpha\boldsymbol{x} + \boldsymbol{y} \qquad \text{(SAXPY)}$$

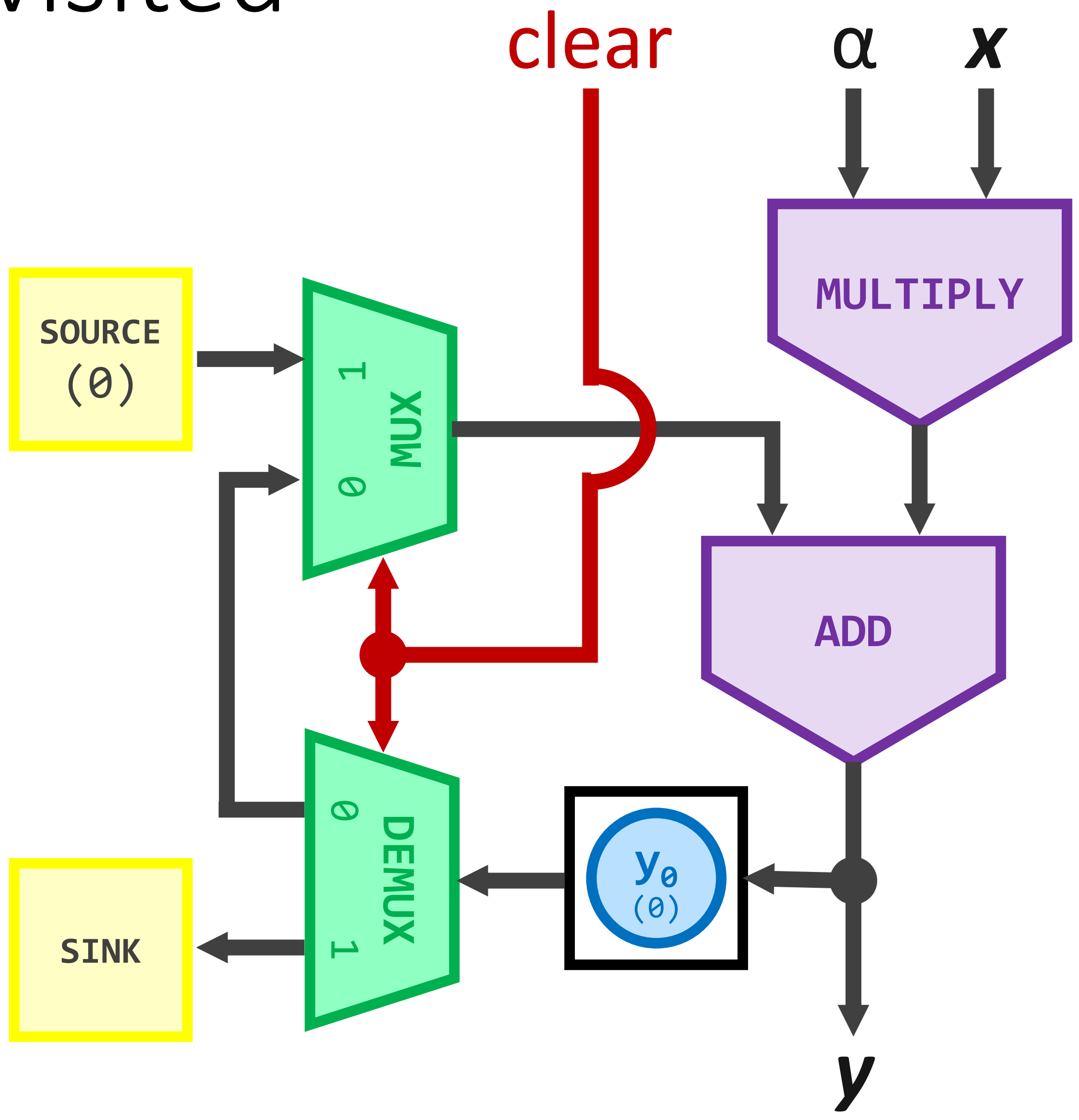
Works well for one vector, but how about the next? Want to reuse this MAC unit without a full system reset



Multiply-accumulate revisited

One solution:

Add “clear” signal to reset the accumulator, send along with each new set of input data



Pipeline performance

Defining asynchronous performance

Latency

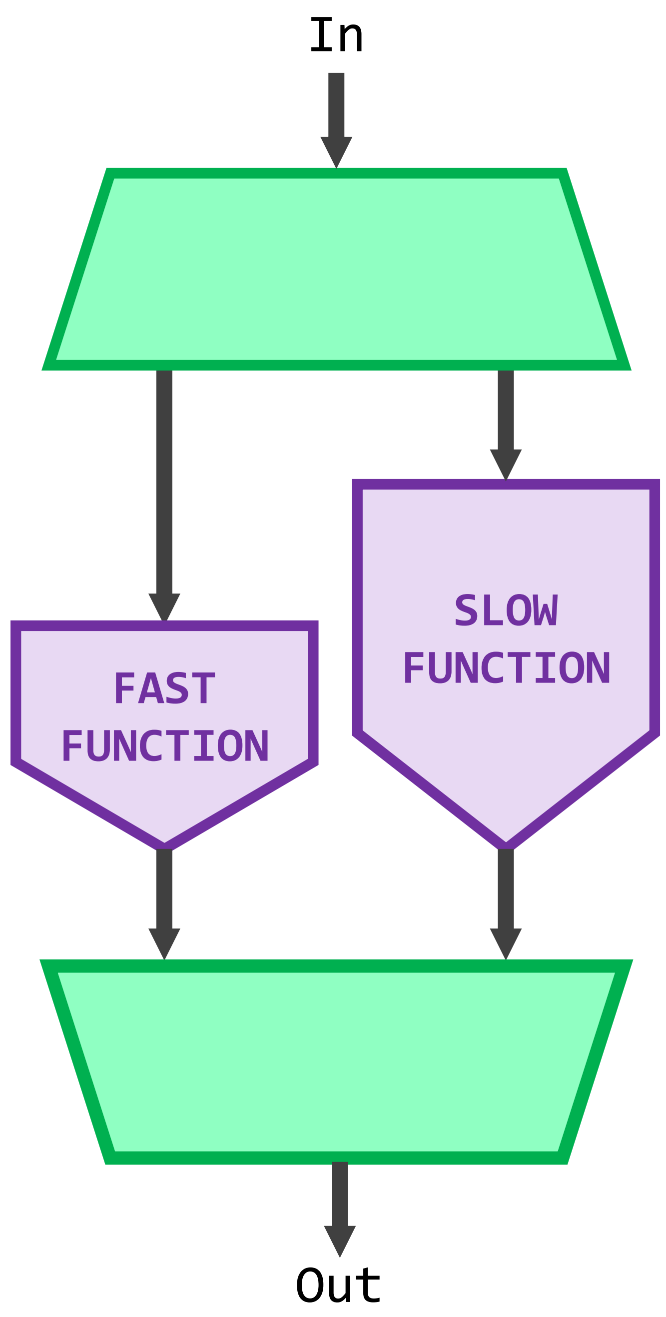
Throughput

Average case performance

Computer architecture principle:
"Make the common case fast"

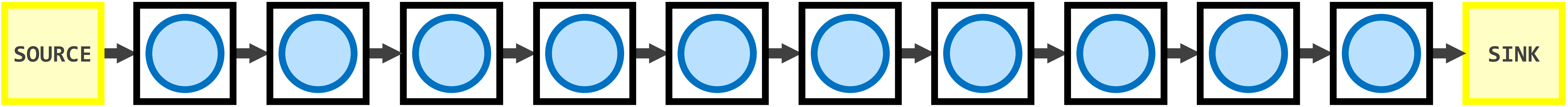
Works especially well in asynchronous design, since performance is only penalized when a given unit is used

Example: divide in a processor ALU

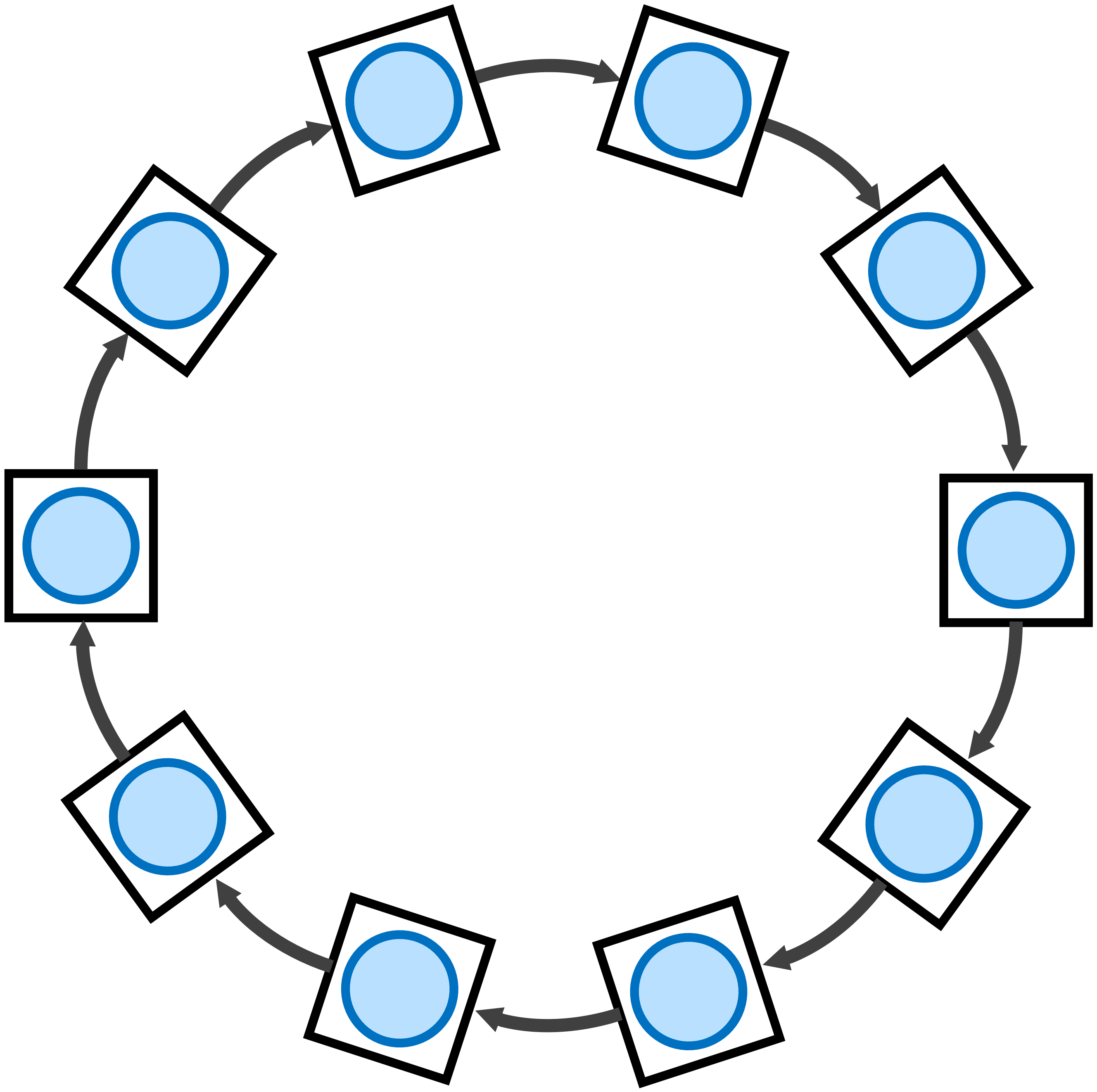


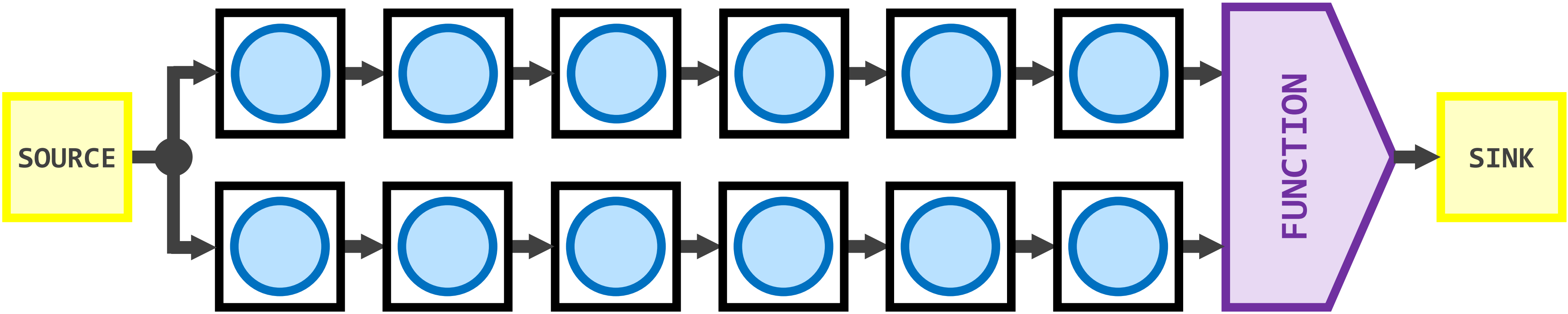
Performance intuition

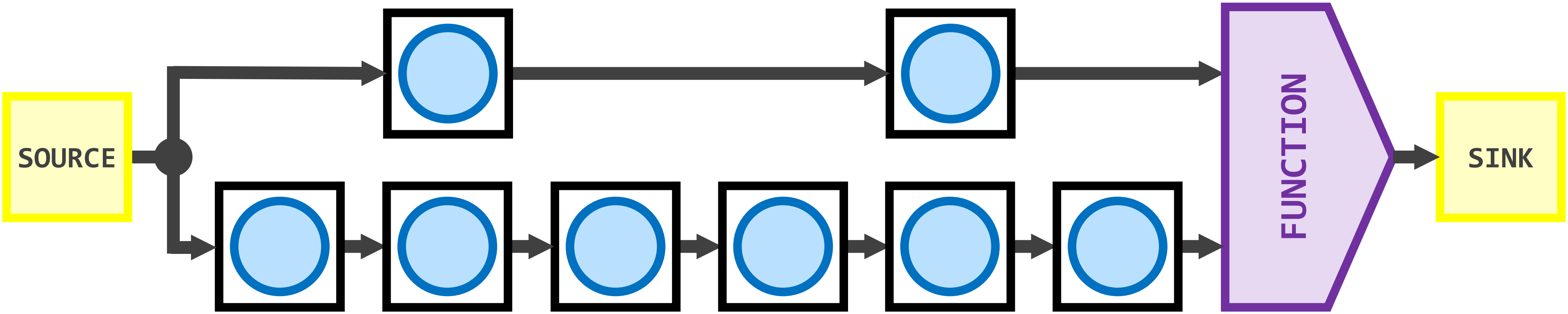
Whiteboard demonstration



SOURCE
SINK

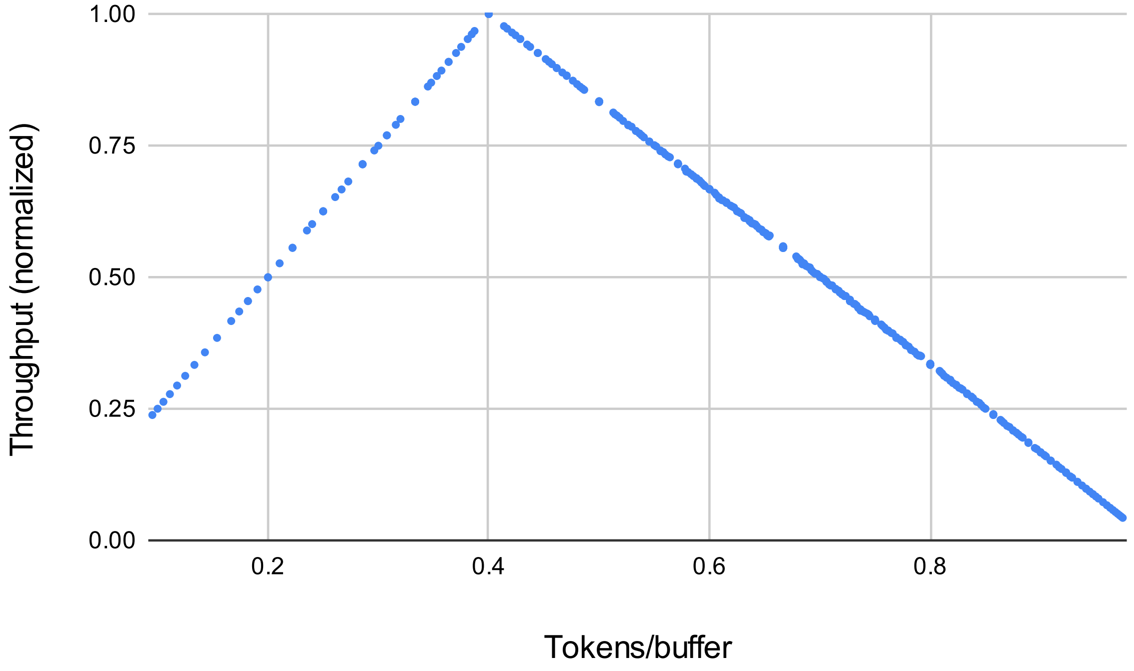


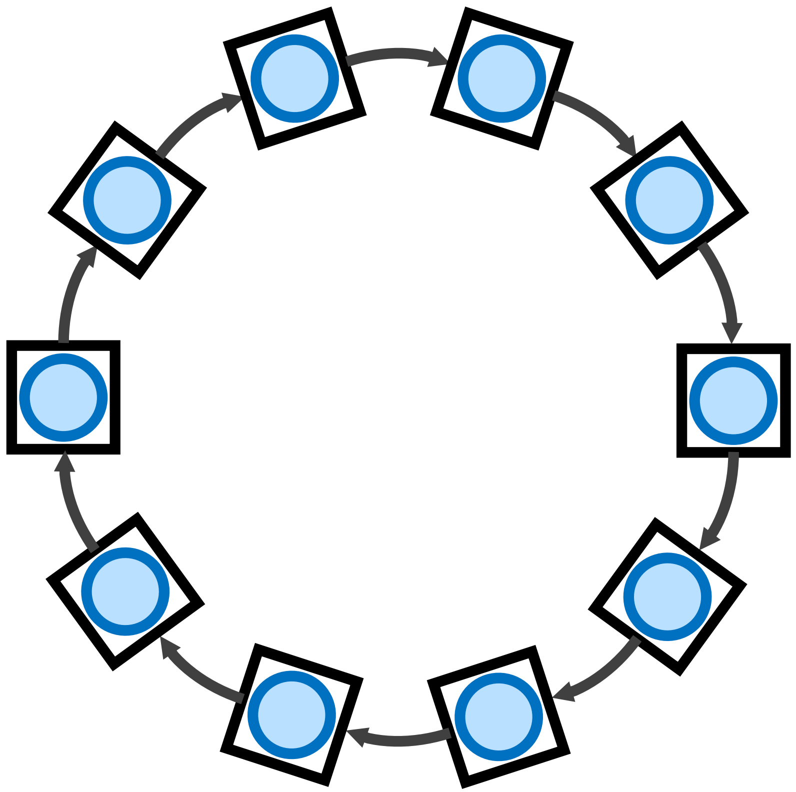

SOURCE
FUNCTION
SINK


SOURCE
FUNCTION
SINK

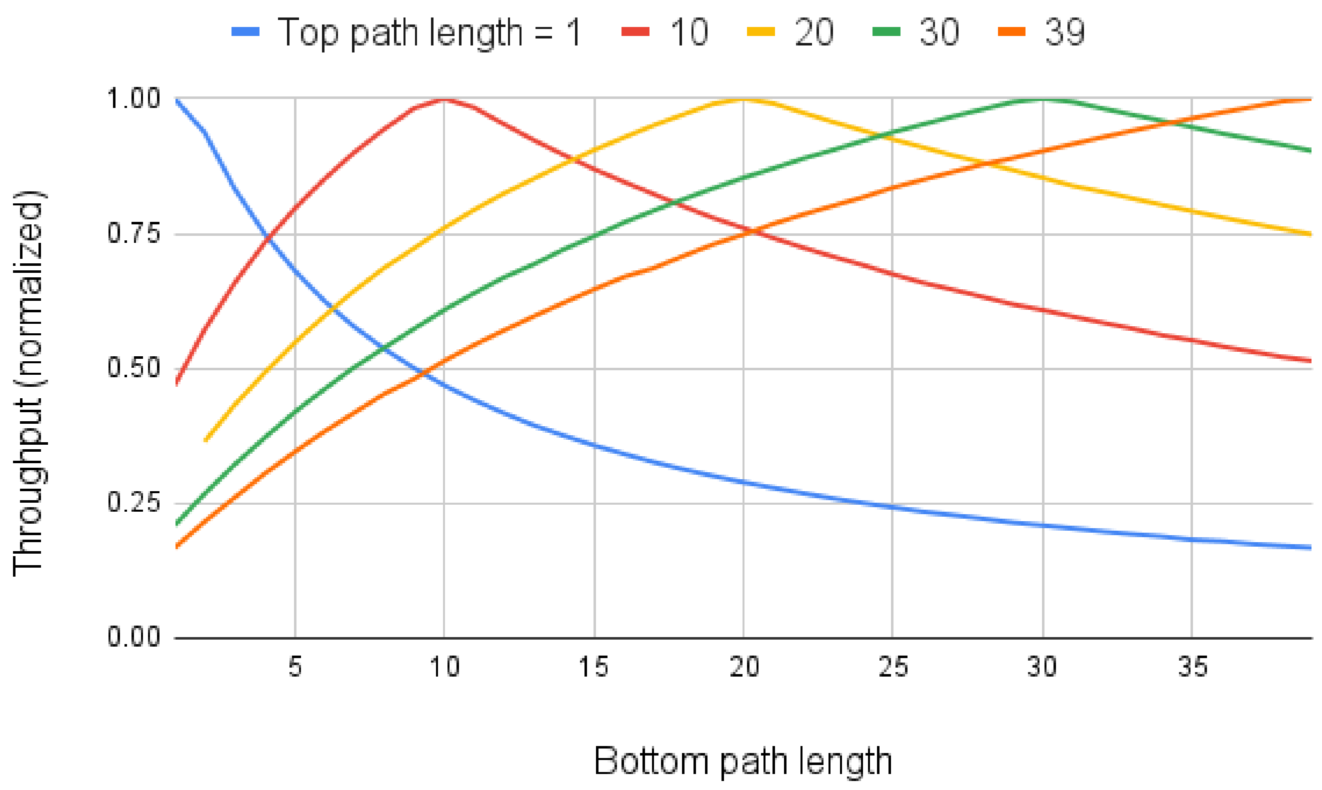
Simulation results

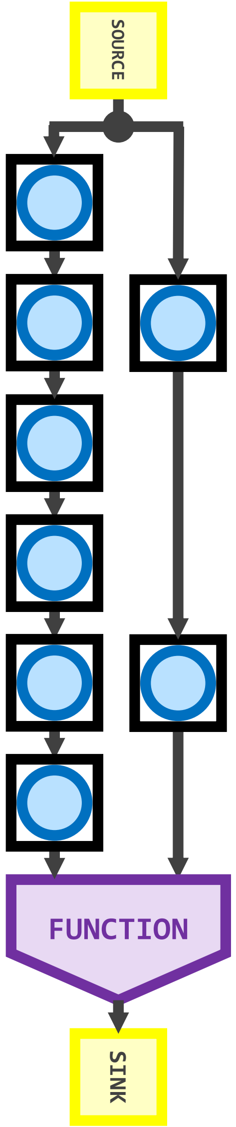
Token ring occupancy vs throughput



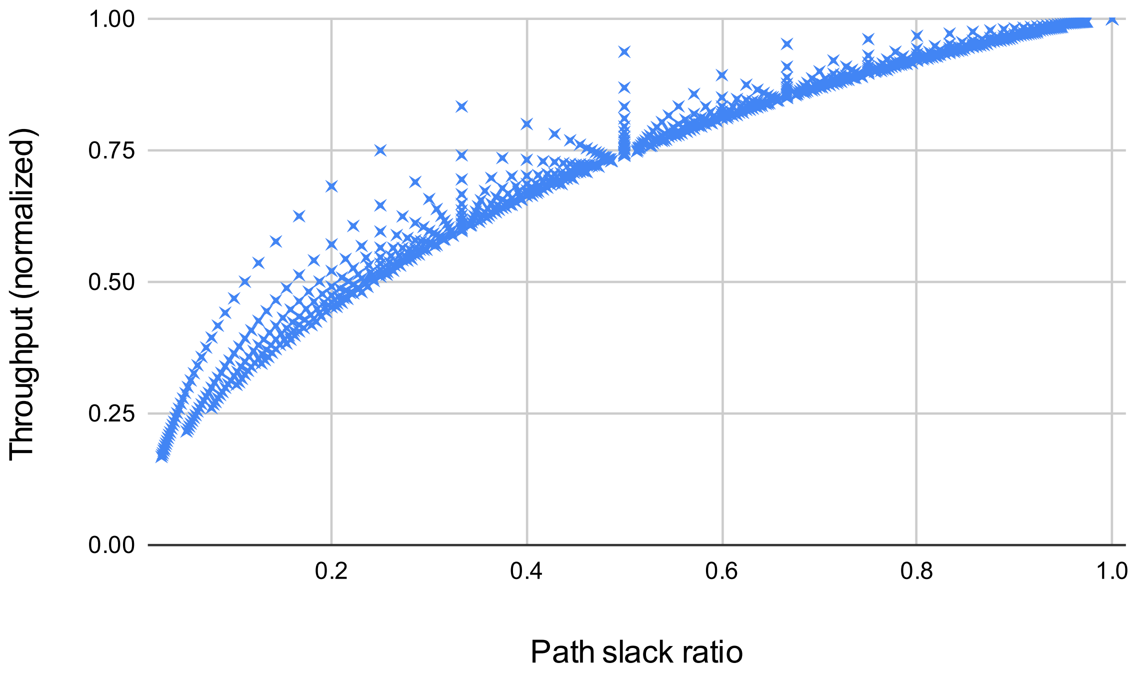


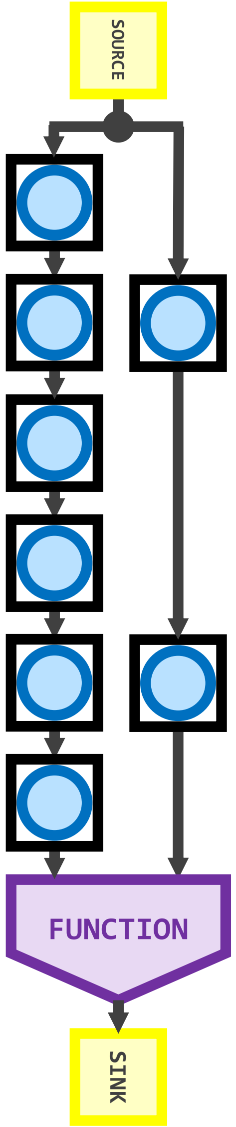
Reconvergent path imbalance vs throughput





Reconvergent path imbalance vs throughput





Homework exercises, references