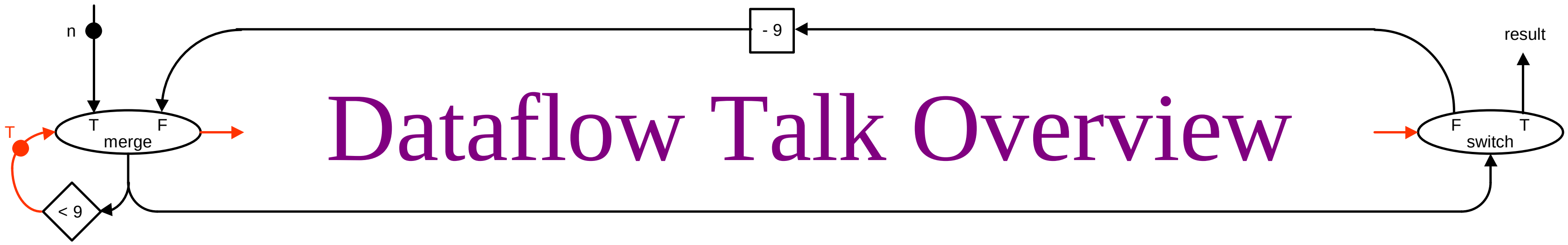
# Datawhat?

J.P. Grossman

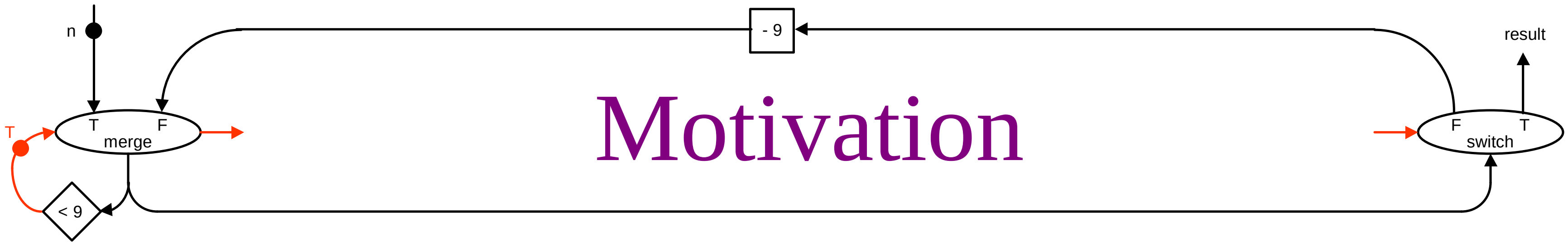
March 14, 19100

# Dataflow Talk Overview



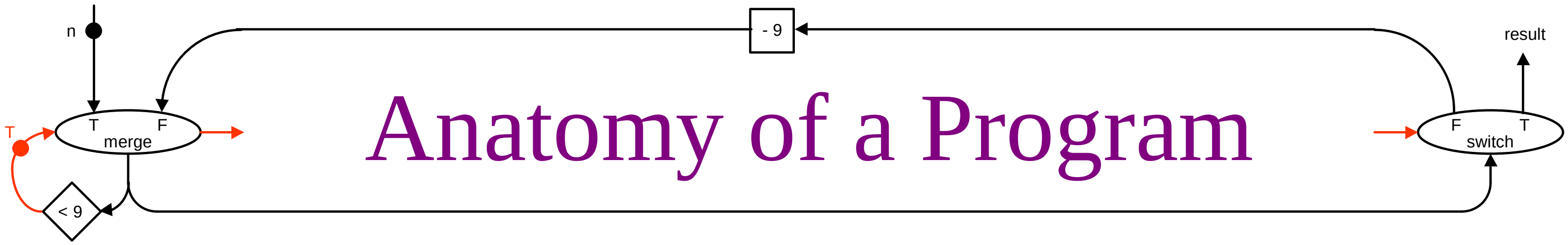

- Major focus: How, what and why
  - What is dataflow?
  - Why is it a good idea?
  - How do I build a dataflow computer?
- Briefly:
  - Existing machines
  - Programming models
  - Problems with dataflow
  - Hybrid approaches

# Motivation



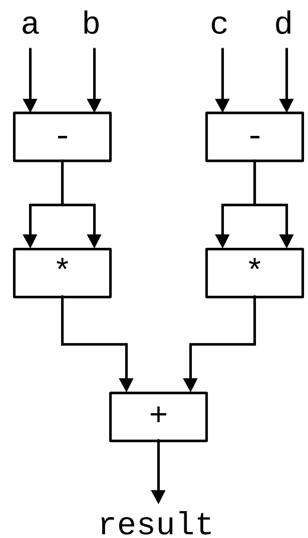

- Given a sequential program, how do we:
  - Find and exploit parallelism?
    - Identify independent operations
    - Execute them in parallel
  - Keep the processor busy with useful work?
    - Tolerate latency
    - Avoid pipeline stalls
    - Cheap synchronization

# Anatomy of a Program

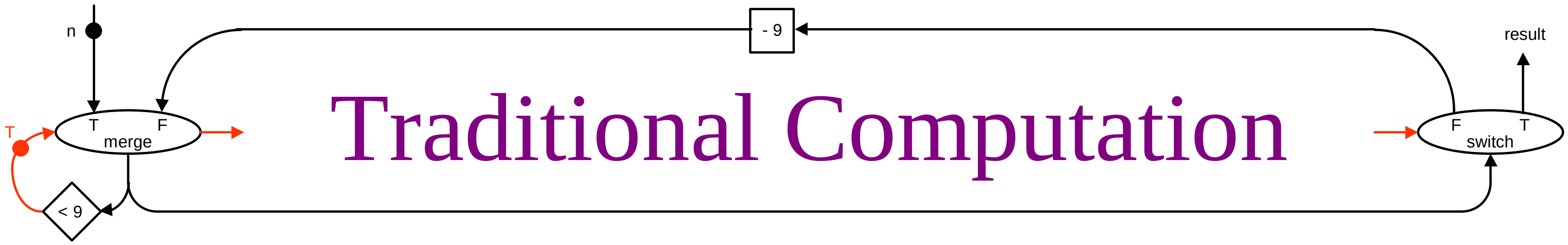



- Computation can be described by a dataflow graph
- All computers evaluate the dataflow graph

```
x = a - b
y = c - d
result = x^2 + y^2
```

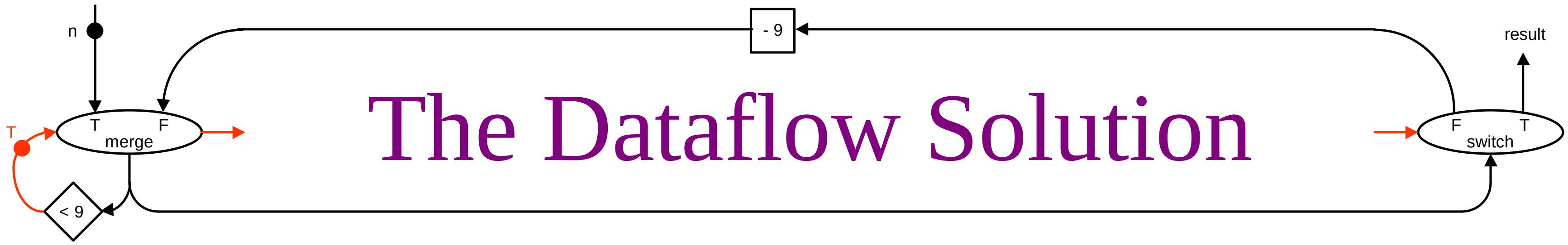


# Traditional Computation



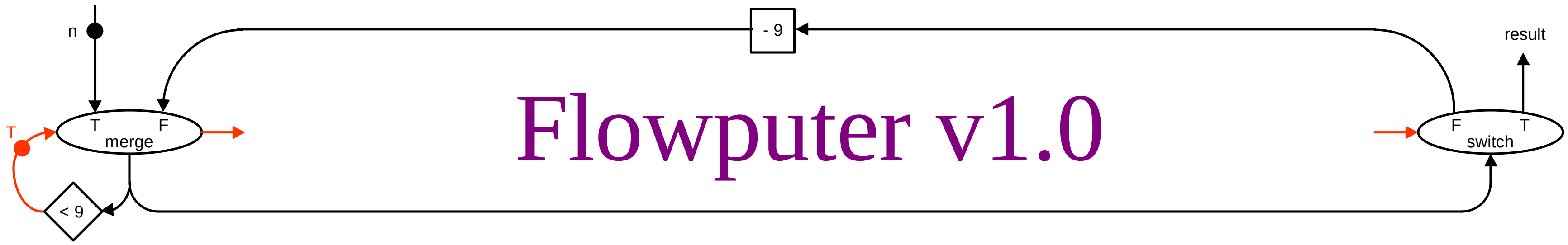

- Operations in dataflow graph are given a sequential order at compile time
- Problems:
  - – Stalls can occur due to dynamic data dependencies
  - – Hides parallelism rather than uses it
  - – Compiling code with a finite number of registers creates artificial dependencies
    - Makes life difficult for architectures that try to be clever
  - – Synchronization is expensive in parallel machines

# The Dataflow Solution



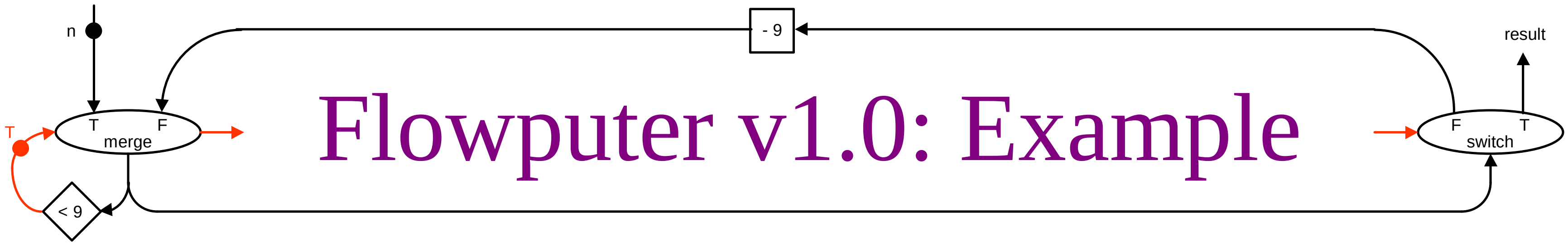

- Program Counters are evil - get rid of them
  – instructions should execute when their operands are available, not when an arbitrary PC says it’s time
- Compile and run the dataflow graph directly
  – compiler produces a set of operations
  – each op specifies one or more result destinations
  – an op can execute as soon its operands are available
- An “ideal” dataflow machine represents the *fastest possible* execution of a program

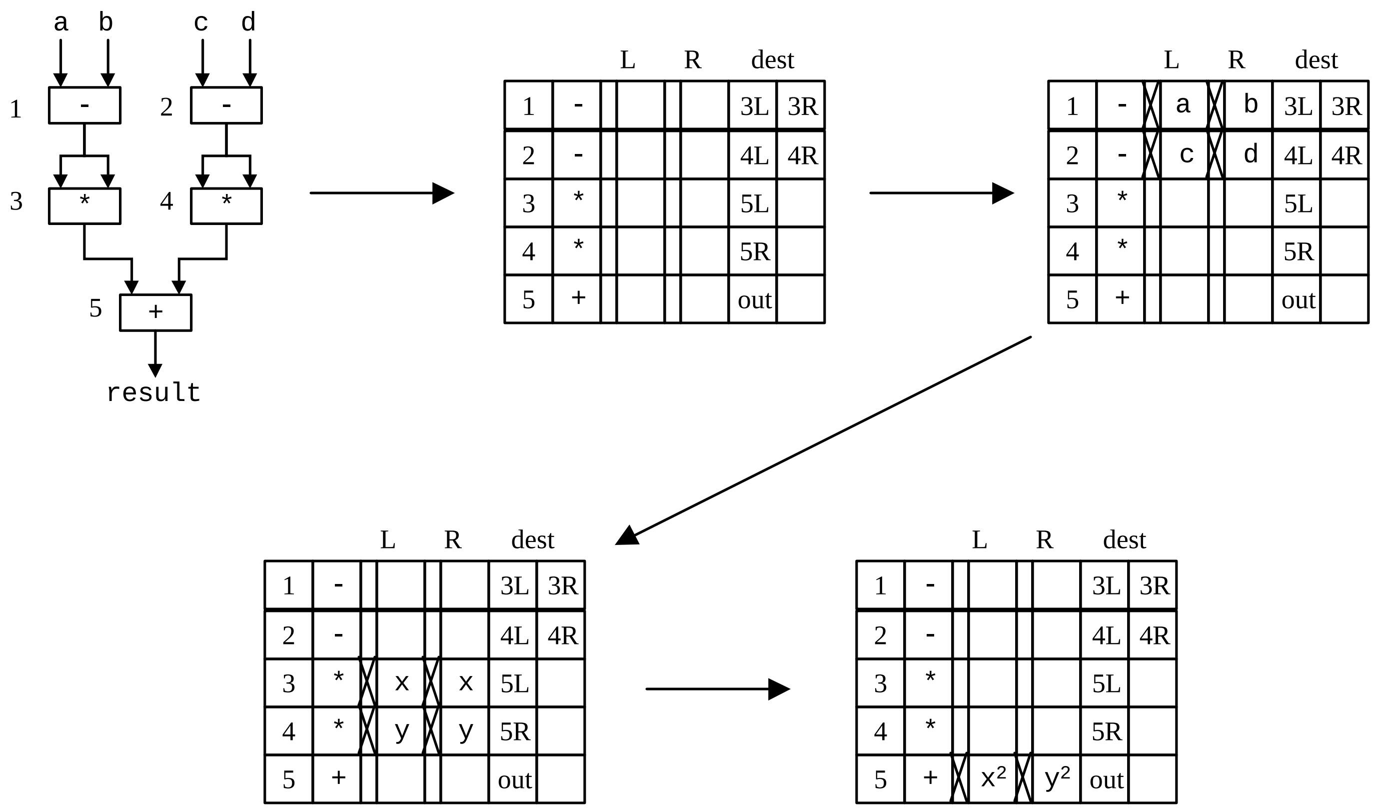
# Flowputer v1.0

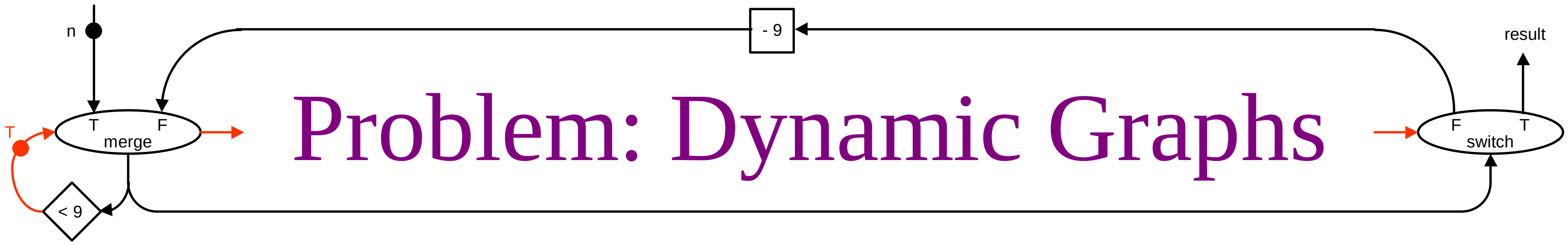



- Basic data structure is an *activity template*
  - opcode, storage for operands (with present bits), destination specifier(s)
- Flowputer maintains a set of *ready* operations
- On each cycle, grab *n* ready operations:
  - execute the operation
  - store the result in the destination(s)
  - check to see if any new operations become ready and, if so, add them to the ready set

# Flowputer v1.0: Example



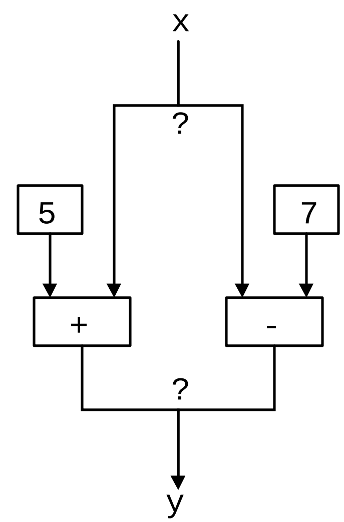





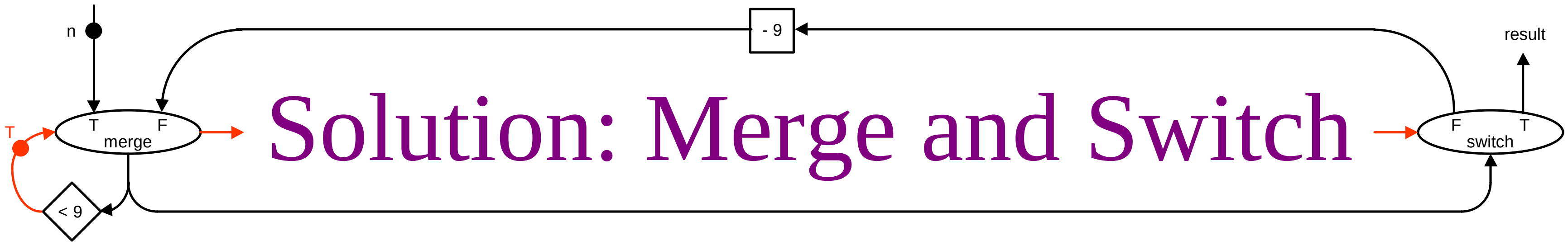

# Problem: Dynamic Graphs

- Most program dataflow graphs are dynamically determined at run time
- Example:

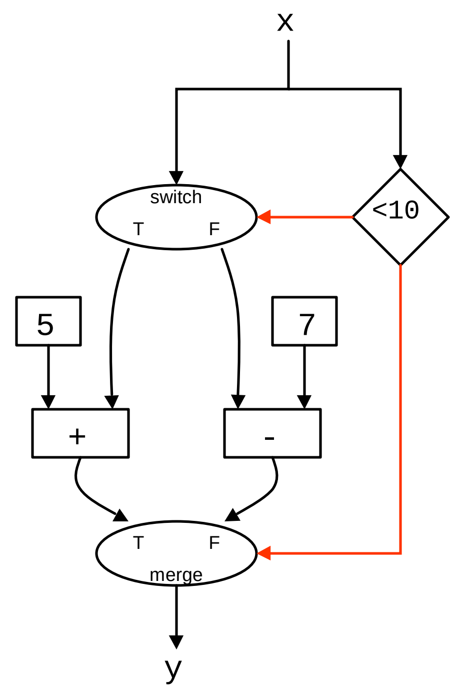
```
if (x < 10)
    y = x + 5;
else
    y = x - 7;
```



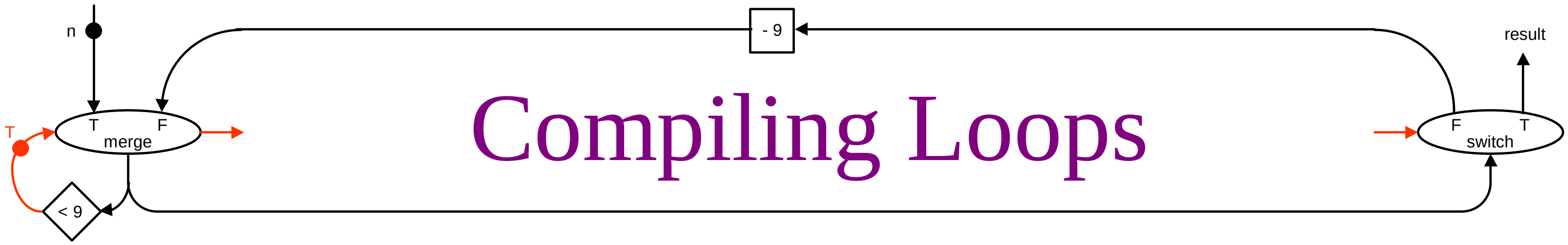
# Solution: Merge and Switch



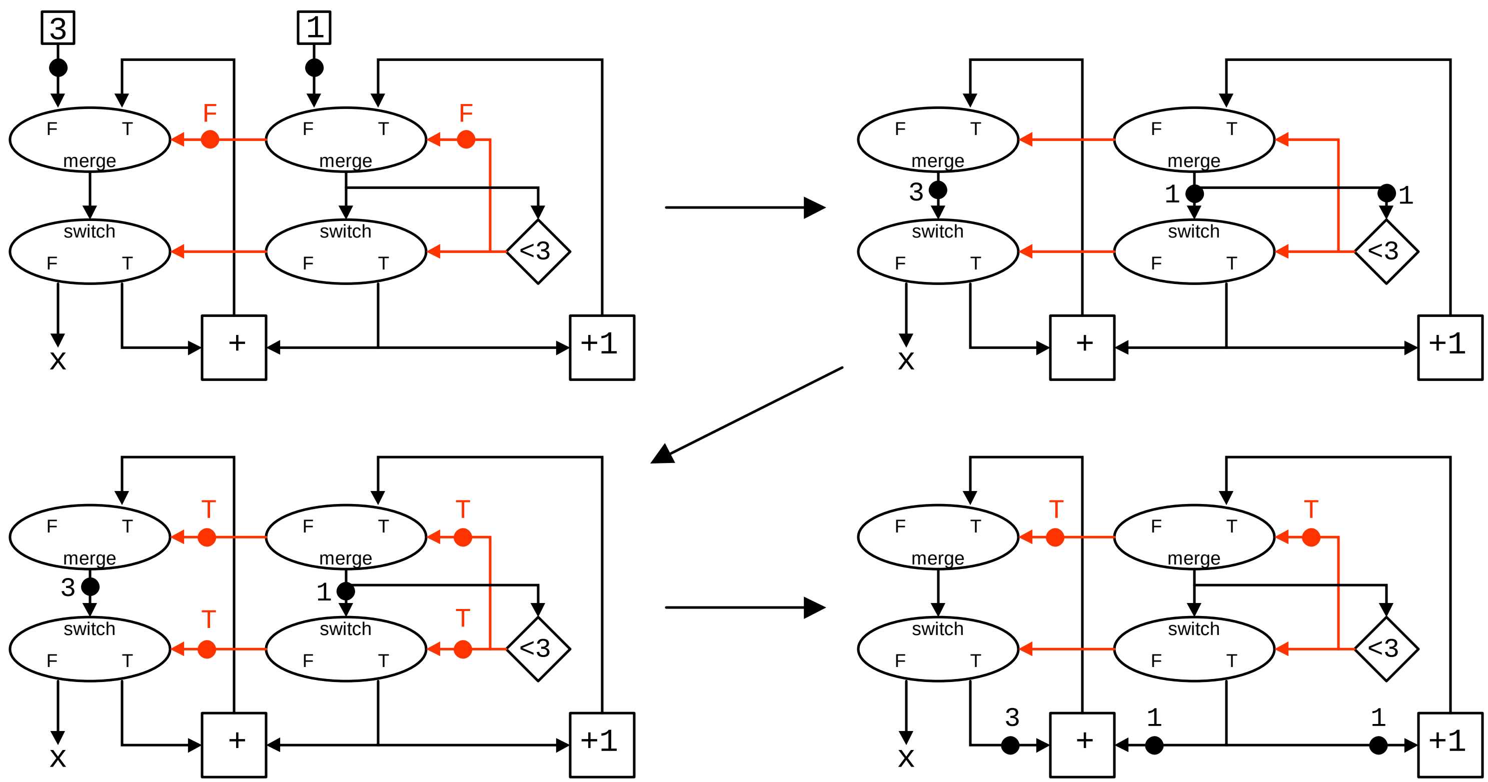

```
if (x < 10)
    y = x + 5;
else
    y = x - 7;
```



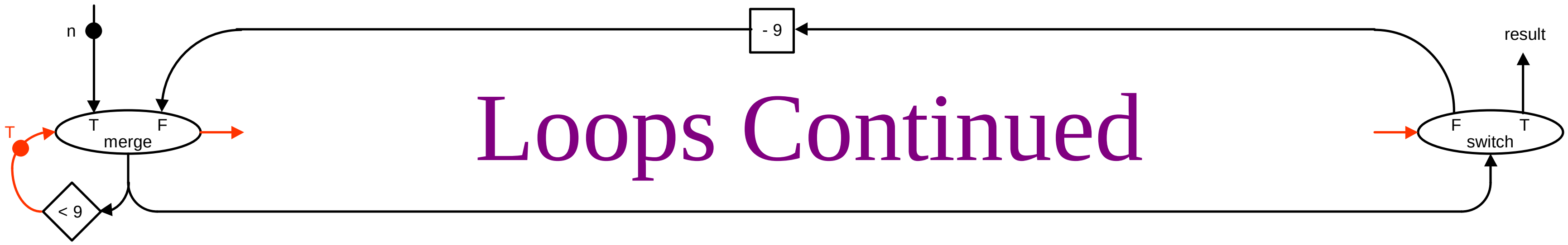
# Compiling Loops

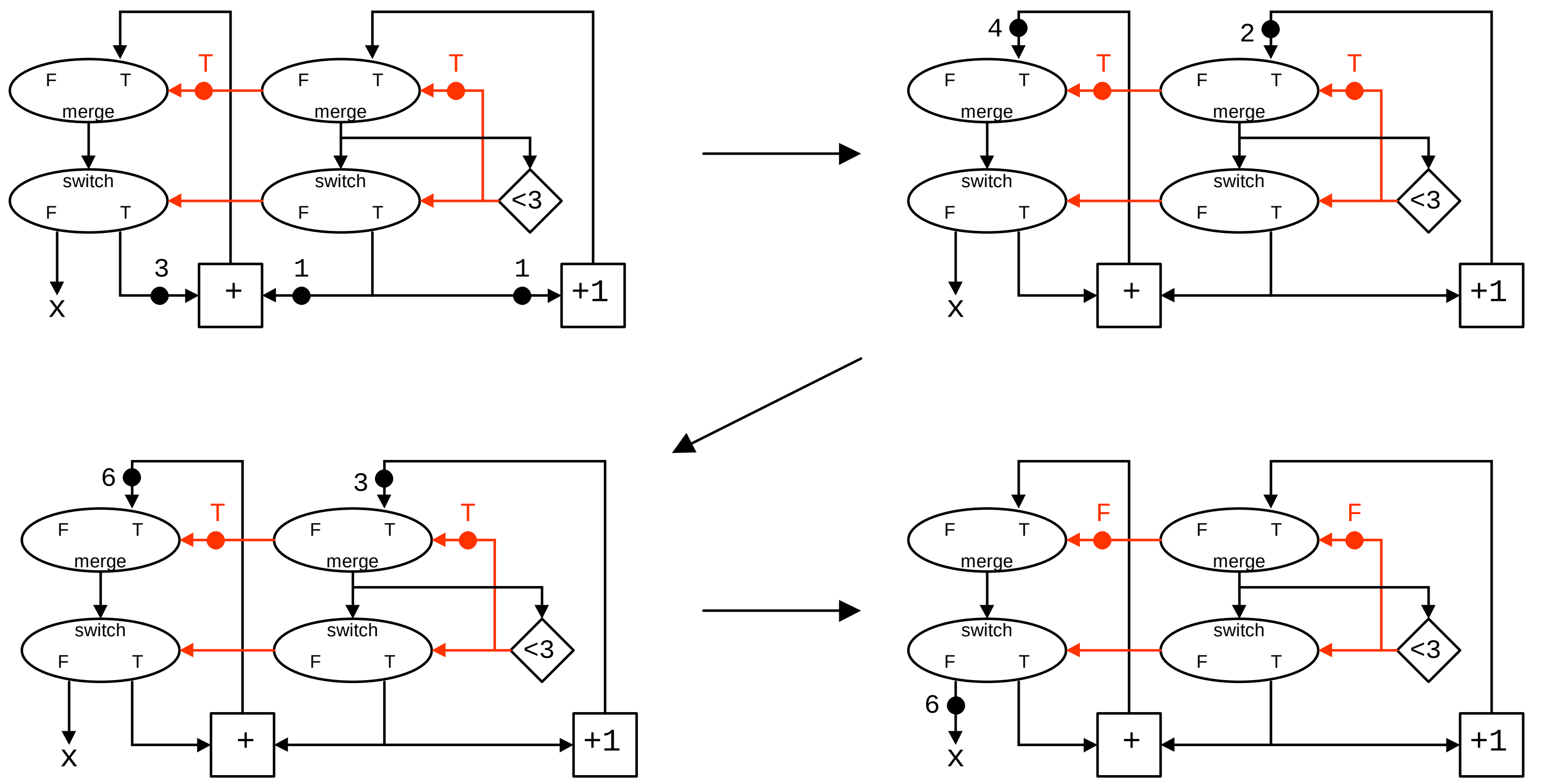



```
X = 3;
for (i = 1 ; i < 3 ; i++)
    x = x + i;
```

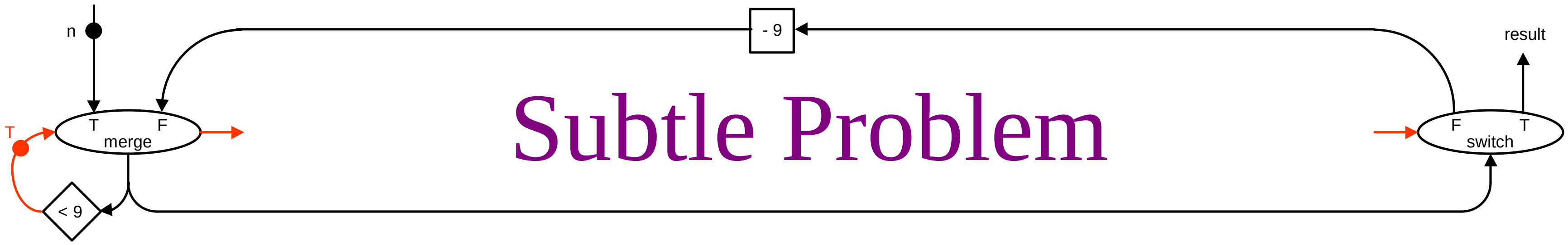


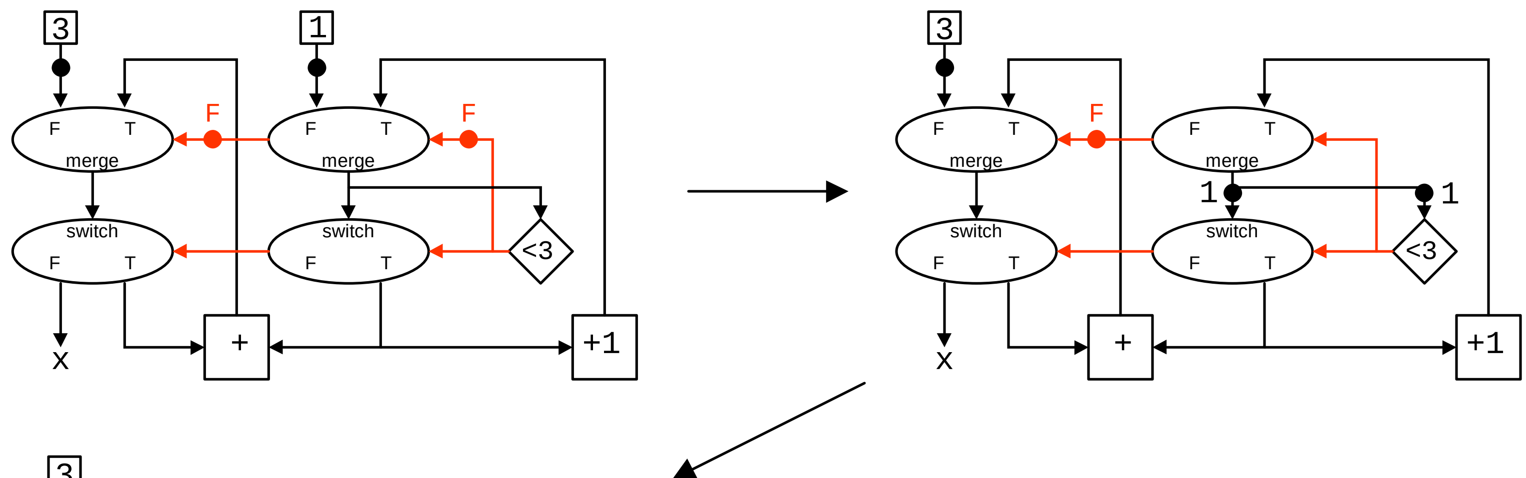
# Loops Continued

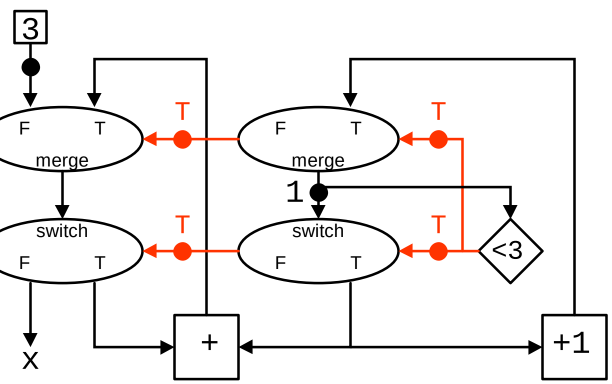


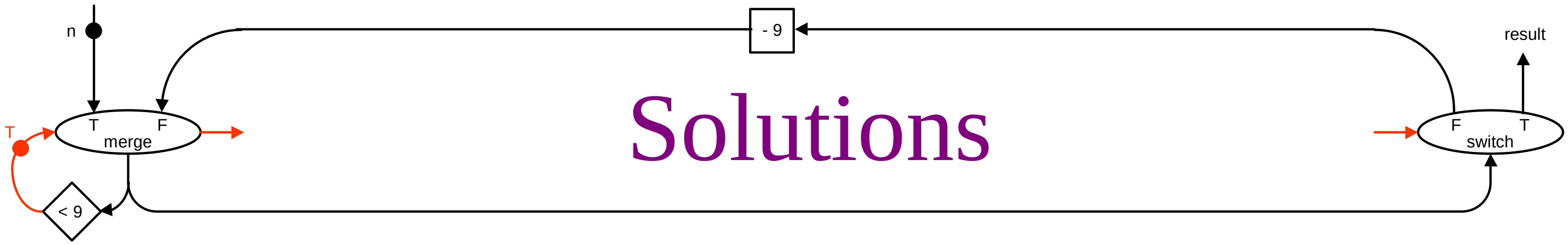
# Subtle Problem



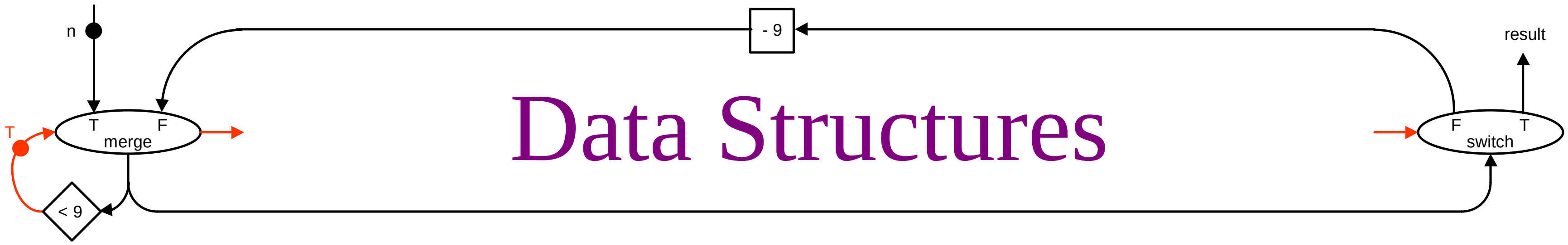

- “F” token is overwritten by “T” token
- Machine deadlocks




# Solutions

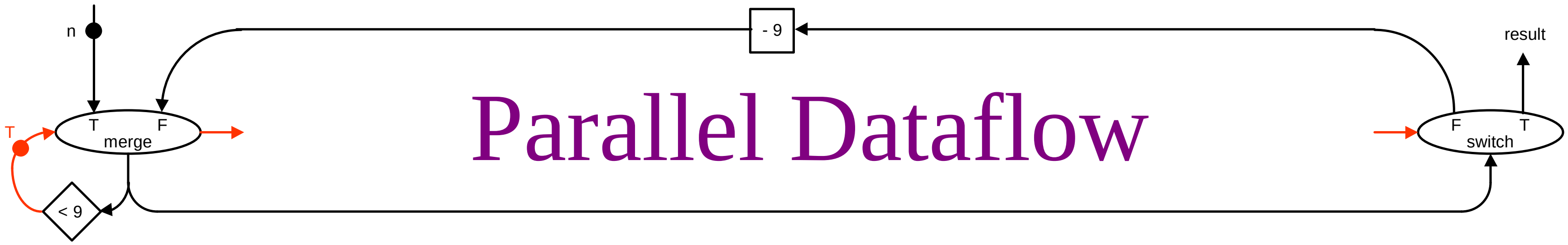
- Only allow one token per arc (Dennis)
  - Complicates hardware
  - Reduces performance and flexibility.. no recursion!
- Tagged Token Dataflow (Manchester, SIGMA-1)
  - Give each function call/loop iteration a unique tag
  - Also complicates hardware (tag matching)
- Explicit Token Store (Monsoon, EM-4)
  - Dynamically allocate frame memory for operands
  - Compiler ensures one token per arc

# Data Structures

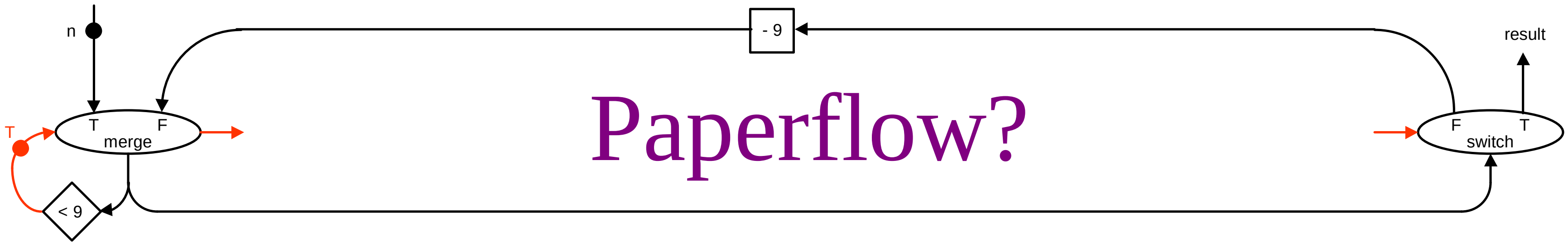



- Can’t pass entire data structures in tokens
- Pass pointers instead.. but pointers to what?
  - Need to emulate “availability of operands”
- Solution: I-structure memory
  - Single write, multiple read, split-phase memory
  - Read to a full slot causes the value to be sent back
  - Read to an empty slot blocks until slot is filled
  - Write to an empty slot causes all waiting reads to be satisfied

# Parallel Dataflow



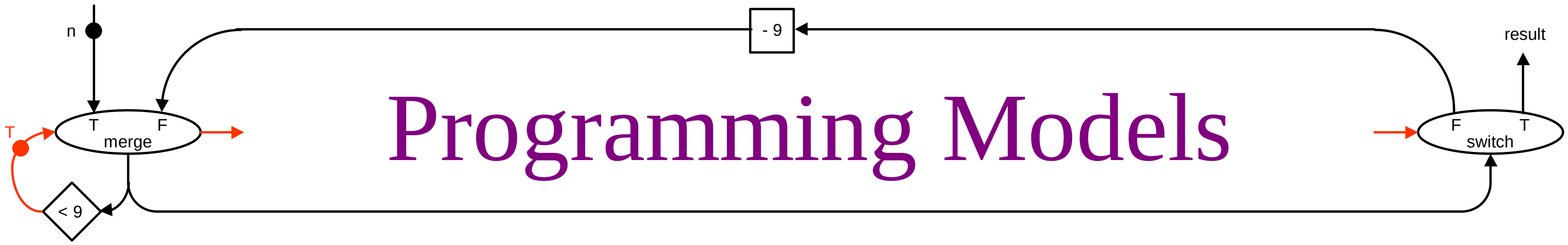

- At any point in time, a dataflow program generally has many ready operations
- Pick your favorite way to execute them in parallel
- Big question: do we get good speedup?
- Experiments with Monsoon indicate that the answer is “yes”
  - 8 processors, interconnection network for routing tokens to correct processor
  - Average speedup of 7.15x




# Paperflow?

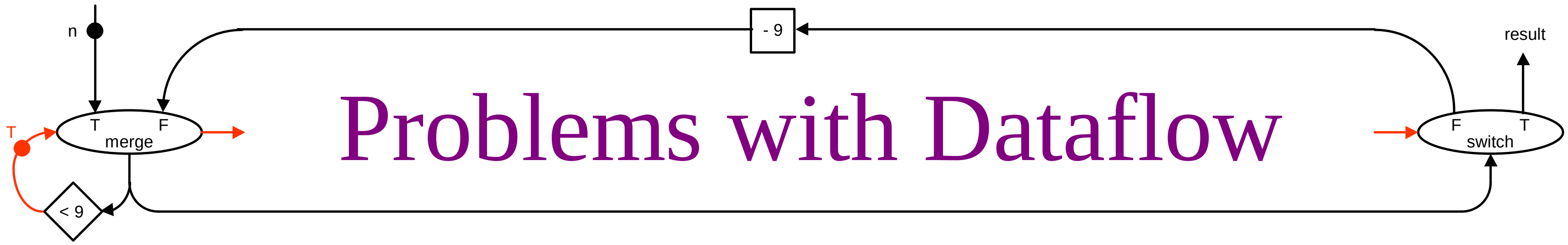
- Lots of paper, not too many actual machines
- Manchester (1981) - Tagged Token
  - 12 functional units, 1 MIPS, 4-8x slower than VAX
- SIGMA-1 (1987) - Tagged Token
  - 128 processors, 170 MFlops
- Monsoon (1988) - Explicit Token Store
  - 8 processor configuration out-performs MIPS R3000
- EM-4 (1990) - Hybrid Explicit Token Store
  - 80 processors, 1 GIPS, 14.63 GB/s peaks

# Programming Models



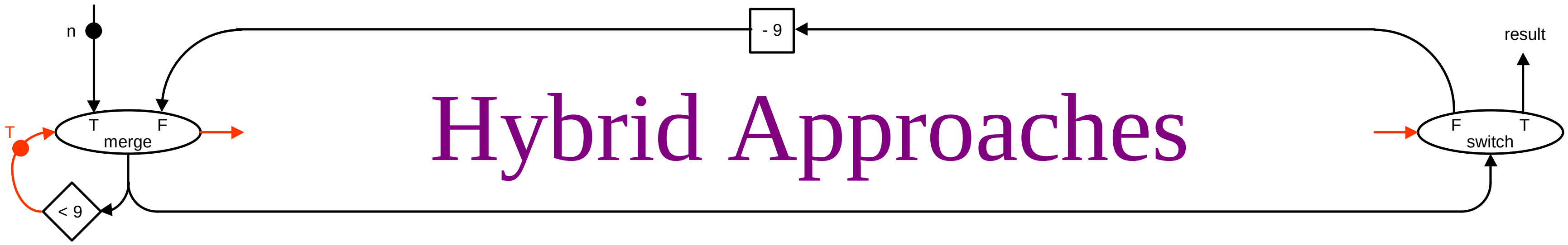

- Id (Monsoon)
  - functional language augmented with I structures
- SISAL (Manchester)
  - Pascal-like single-assignment language
- Both languages enforce a write-once read-many programming style
  - Relatively easy to compile to efficient dataflow code
  - Relatively easy to lose toes

# Problems with Dataflow



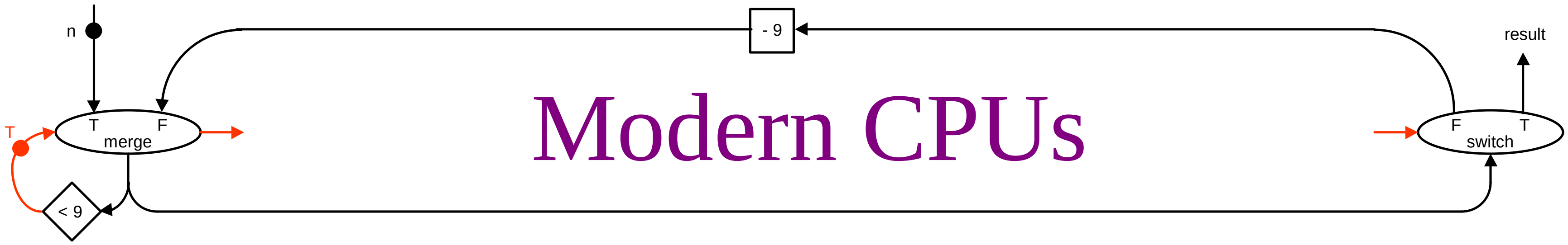

- Some obvious problems:
  - complicated and expensive hardware
  - confusing programming model
- Some not-so-obvious problems:
  - Extremely high bandwidth requirements
    - Each operation injects token(s) into the network
  - Runaway resource requirements
    - Direct consequence of the goal of limitless parallelism
    - Bounding loops works, but doesn’t deal with general recursion

# Hybrid Approaches



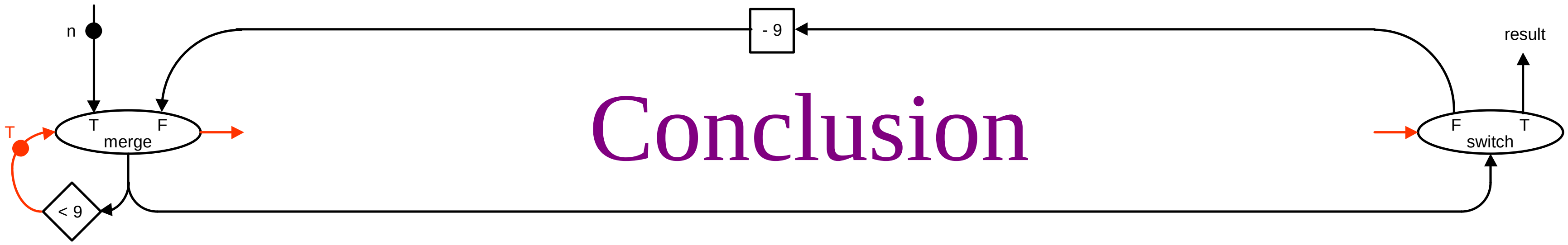

- Scheduling Quanta (Iannucci)
  - Use dataflow mechanisms to schedule instructions on a coarser granularity
  - Similar to “Strongly Connected Blocks” (EM-4)
- P-RISC (Nikhil, Arvind)
  - Looks like RISC with a few extra instructions
    - unless you look at the storage model (not recommended)

# Modern CPUs




- Most modern CPUs are really dataflow computers on a small scale!
- Each arc should have a unique name
  - Register renaming!
- Results should be sent directly to ops
  - Out of order instruction buffer!
- Independent ops should execute in parallel
  - Superscalar!

# Conclusion




- Advantages of dataflow:
  - Naturally finds all available parallelism
  - Will never stall when there is useful work to do
- Disadvantages of dataflow:
  - Complex hardware
  - Overall performance/efficiency is still worse than a good modern processor
  - Confusing programming model
- Five star rating: ★ ★