# StreamIt: High-Level Stream Programming on Raw

Michael Gordon, Michal Karczmarek, Andrew Lamb,
Jasper Lin, David Maze, William Thies, and Saman Amarasinghe

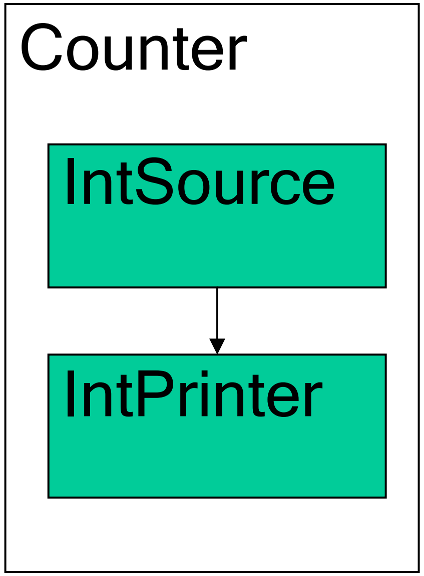
March 6, 2003

# The StreamIt Language

- Why use the StreamIt compiler?
  - Automatic partitioning and load balancing
  - Automatic layout
  - Automatic switch code generation
  - Automatic buffer management
  - Aggressive domain-specific optimizations
- All with a simple, high-level syntax!
  - Language is architecture-independent

# A Simple Counter

```
void->void pipeline Counter() {
    add IntSource();
    add IntPrinter();
}
void->int filter IntSource() {
    int x;
    init { x = 0; }
    work push 1 { push (x++); }
}
int->void filter IntPrinter() {
    work pop 1 { print(pop()); }
}
```



# Demo

- Compile and run the program

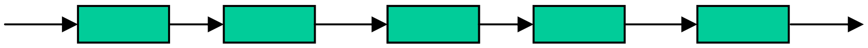
```
counter % knit --raw 4 Counter.str
counter % make -f Makefile.streamit run
```

- Inspect graphs of program

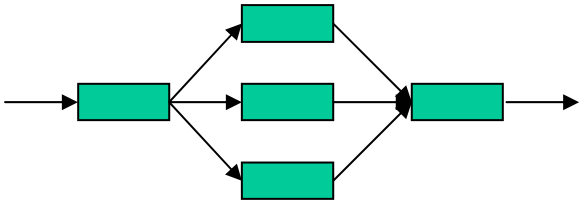
```
counter % dotty schedule.dot
counter % dotty layout.dot
```

# Representing Streams

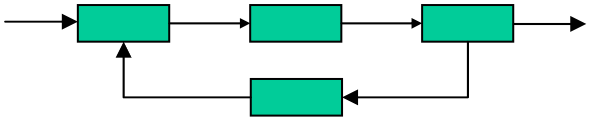
- Hierarchical structures:
  - Pipeline



  - SplitJoin



  - Feedback Loop



- Basic programmable unit:  Filter

# Representing Filters

- Autonomous unit of computation

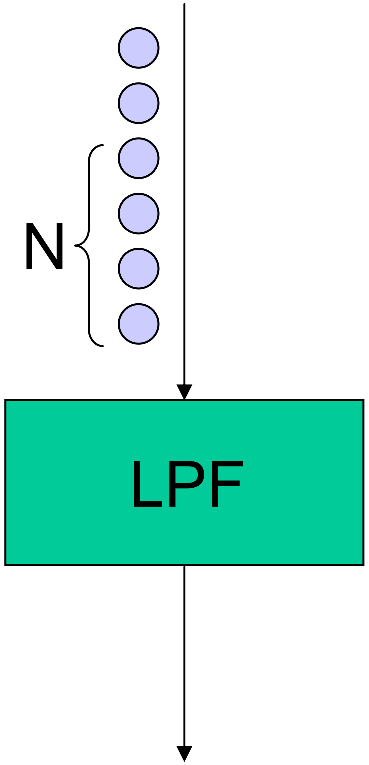
  - No access to global resources
  - Communicates through FIFO channels
    - pop()   - peek(index)   - push(value)
  - Peek / pop / push rates must be constant
- Looks like a Java class, with
  - An initialization function
  - A steady-state “work” function

# Filter Example:  LowPassFilter

```
float->float filter LowPassFilter (int N) {
   float[N] weights;

   init {
      weights = calcWeights(N);
   }

   work push 1 pop 1 peek N {
      float result = 0;
      for (int i=0; i<N; i++) {
         result += weights[i] * peek(i);
      }
      push(result);
      pop();
   }
}
```

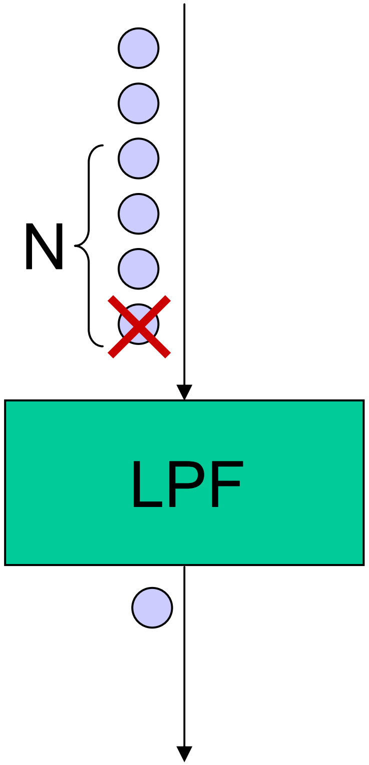
# Filter Example:  LowPassFilter

```
float->float filter LowPassFilter (int N) {
  float[N] weights;

  init {
    weights = calcWeights(N);
  }

  work push 1 pop 1 peek N {
    float result = 0;
    for (int i=0; i<N; i++) {
      result += weights[i] * peek(i);
    }
    push(result);
    pop();
  }
}
```



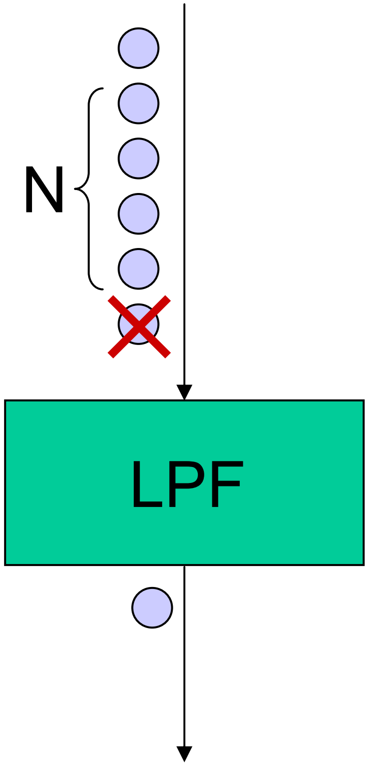
# Filter Example:  LowPassFilter

```
float->float filter LowPassFilter (int N) {
   float[N] weights;

   init {
      weights = calcWeights(N);
   }

   work push 1 pop 1 peek N {
      float result = 0;
      for (int i=0; i<N; i++) {
         result += weights[i] * peek(i);
      }
      push(result);
      pop();
   }
}
```



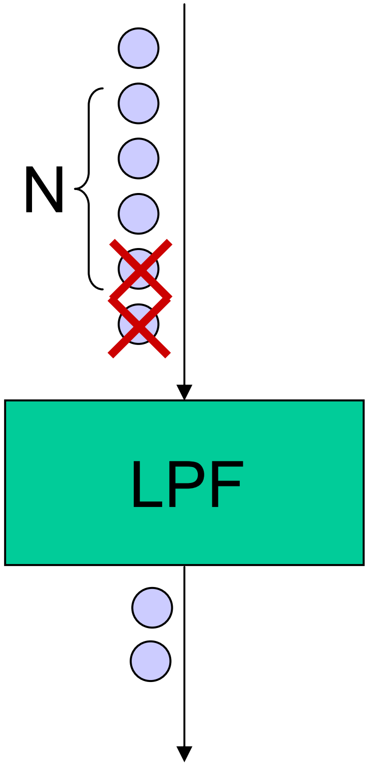
# Filter Example:  LowPassFilter

```
float->float filter LowPassFilter (int N) {
  float[N] weights;

  init {
    weights = calcWeights(N);
  }

  work push 1 pop 1 peek N {
    float result = 0;
    for (int i=0; i<N; i++) {
      result += weights[i] * peek(i);
    }
    push(result);
    pop();
  }
}
```



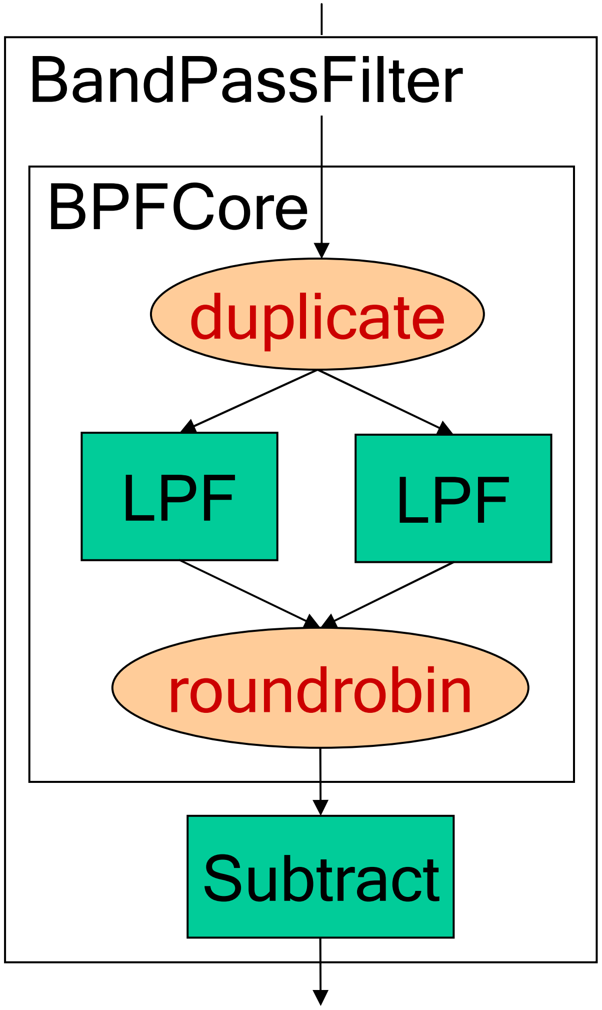
# Filter Example: LowPassFilter

```
float->float filter LowPassFilter (int N) {
  float[N] weights;

  init {
    weights = calcWeights(N);
  }

  work push 1 pop 1 peek N {
    float result = 0;
    for (int i=0; i<N; i++) {
      result += weights[i] * peek(i);
    }
    push(result);
    pop();
  }
}
```



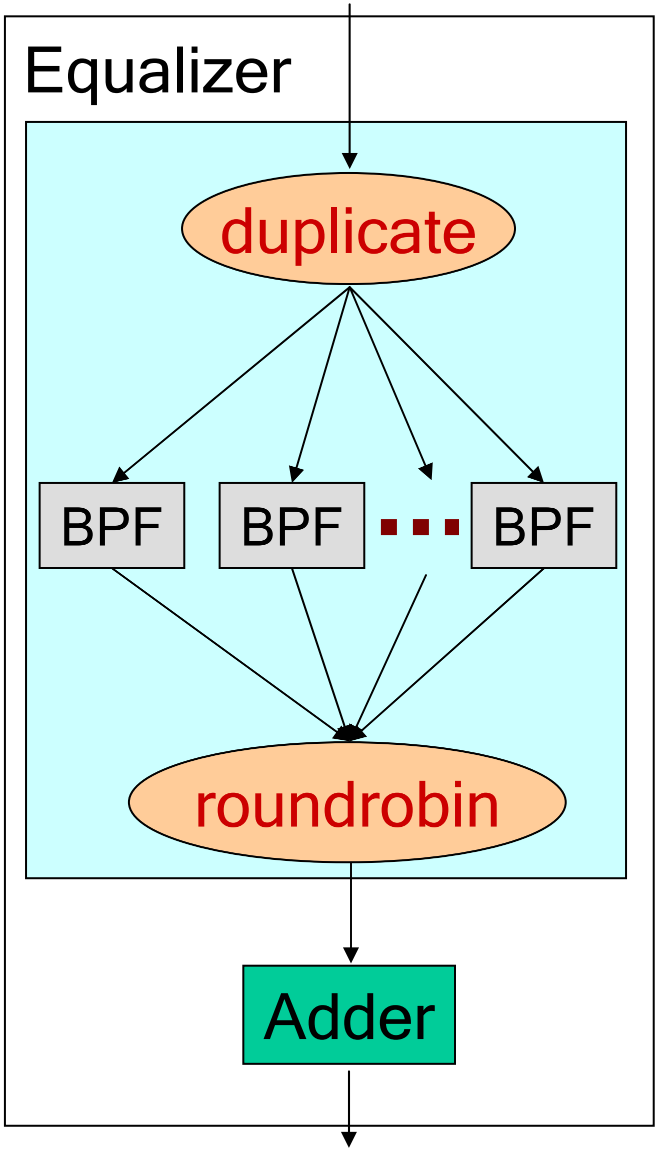
# SplitJoin Example:  BandPass Filter

```
float->float pipeline BandPassFilter(float low, float high) {
  add BPFCore(low, high);
  add Subtract();
}
float->float splitjoin BPFCore(float low, float high) {
  split duplicate;
  add LowPassFilter(high);
  add LowPassFilter(low);
  join roundrobin;
}
float->float filter Subtract {
  work pop 2 push 1 {
    float val1 = pop();
    float val2 = pop();
    push(val1 – val2);
  }
}
```



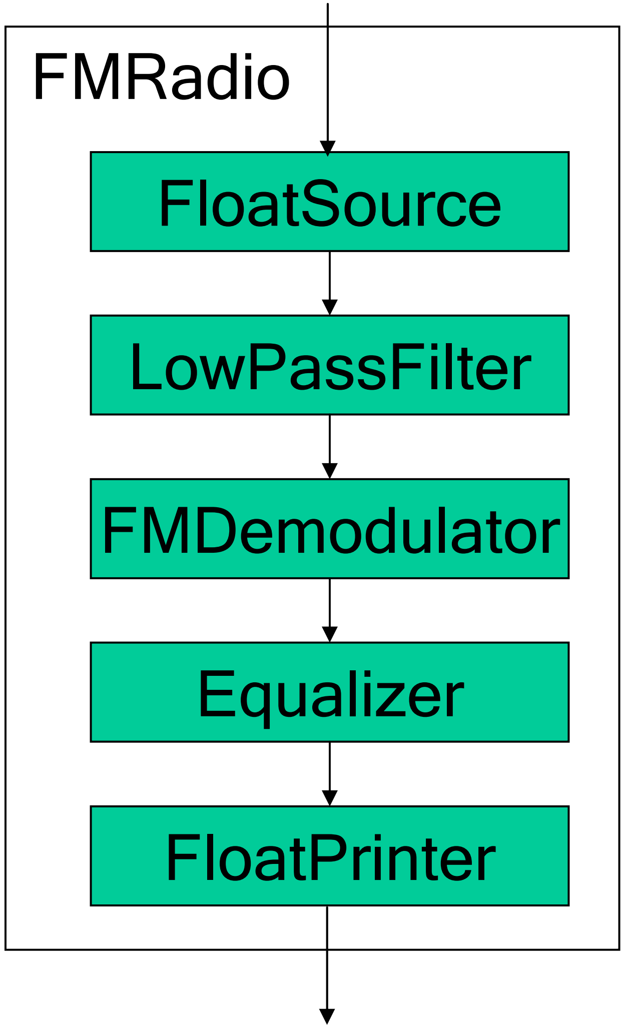
# Parameterization:  Equalizer

```
float->float pipeline Equalizer (int N) {
  add splitjoin {
    split duplicate;
    float freq = 10000;
    for (int i = 0; i < N; i ++, freq*=2) {
      add BandPassFilter(freq, 2*freq);
    }
    join roundrobin;
  }
  add Adder(N);
}
```



# FM Radio

```
float->float pipeline FMRadio {
    add FloatSource();
    add LowPassFilter();
    add FMDemodulator();
    add Equalizer(8);
    add FloatPrinter();
}
```



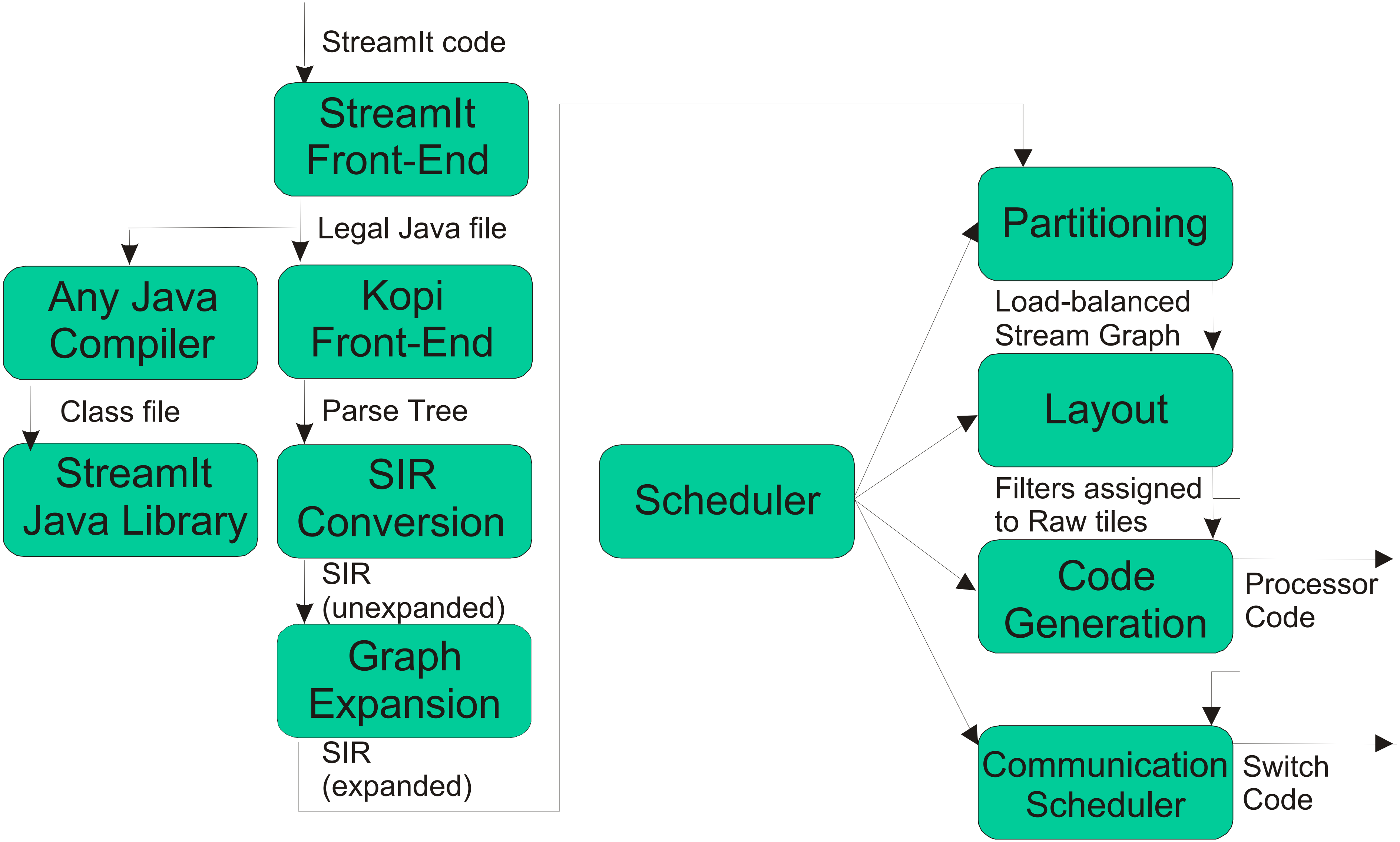
# Demo:  Compile and Run

```
fm % knit --raw 4 --partition --numbers 10 FMRadio.str
fm % make -f Makefile.streamit run
```

Options used:

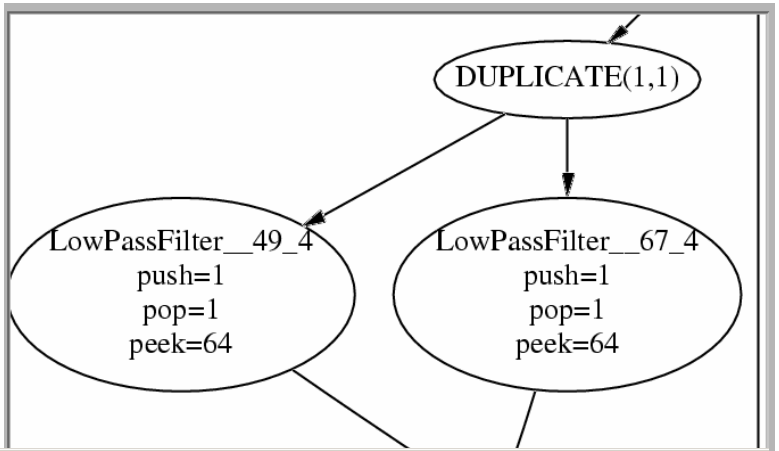
| | |
|---|---|
| --raw 4 | target 4x4 raw machine |
| --partition | use automatic greedy partitioning |
| --numbers 10 | gather numbers for 10 iterations, and store in results.out |

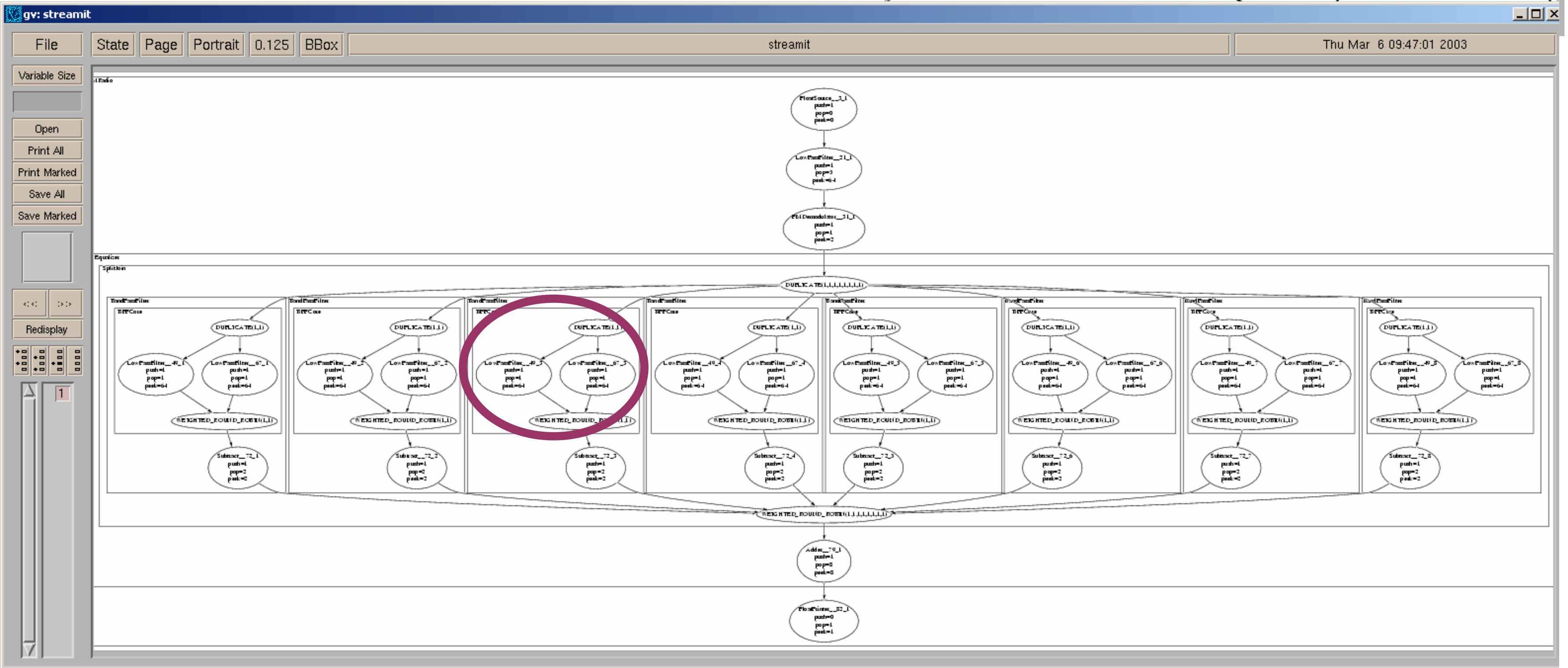
# Compiler Flow Summary



# Stream Graph Before Partitioning

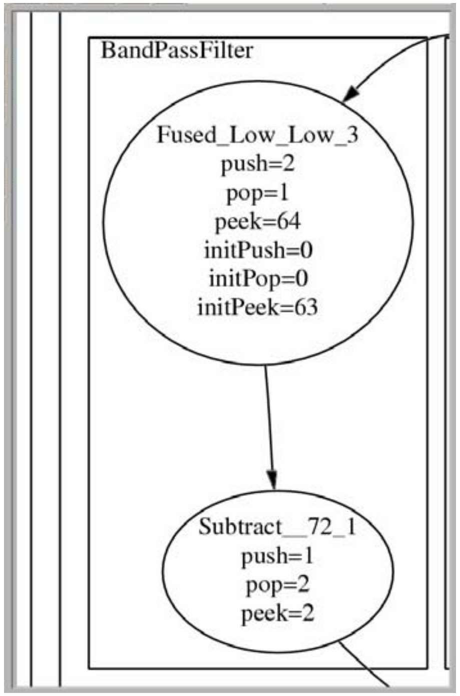
```
fm % dotty before.dot
```

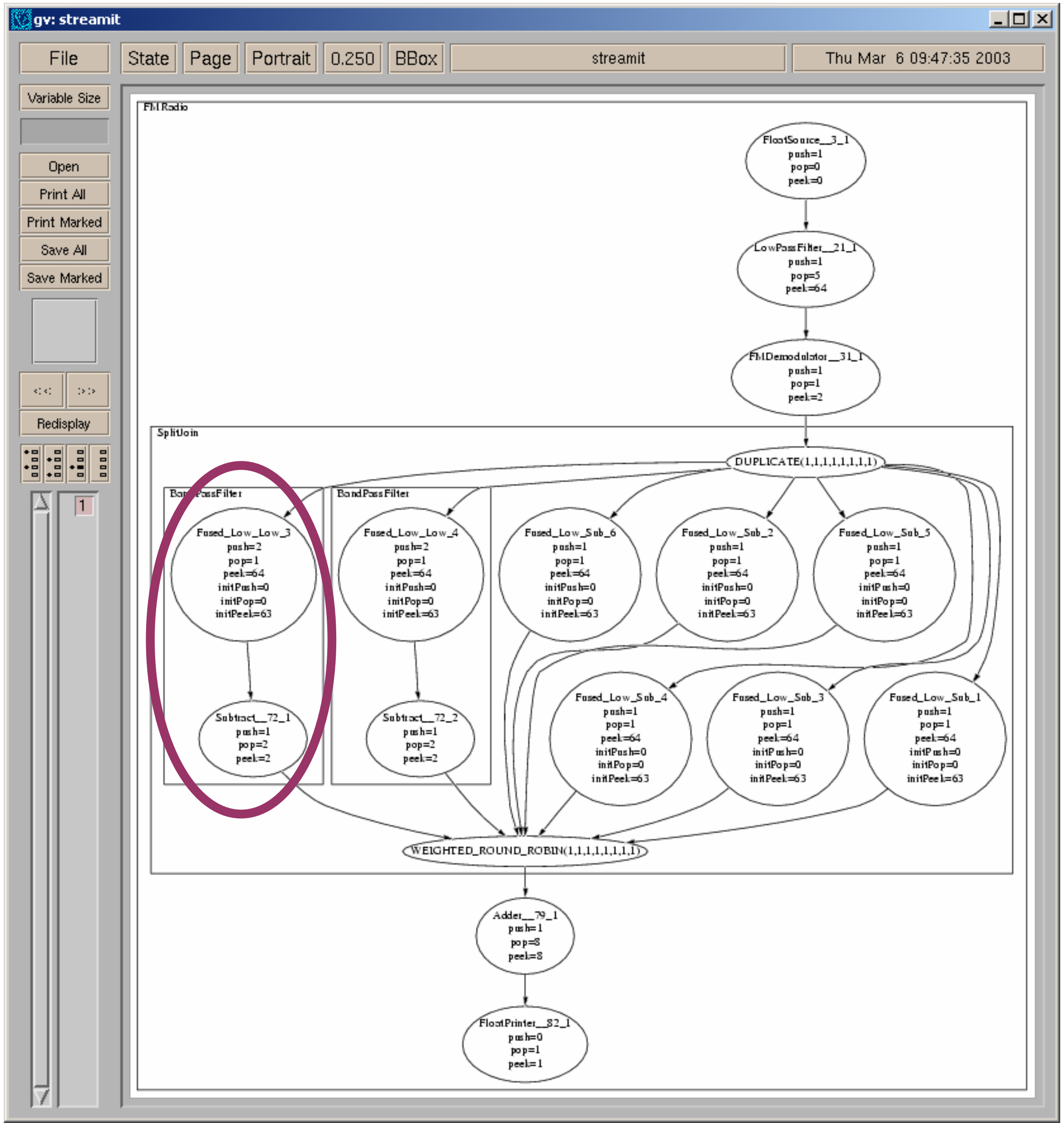




# Stream Graph After Partitioning

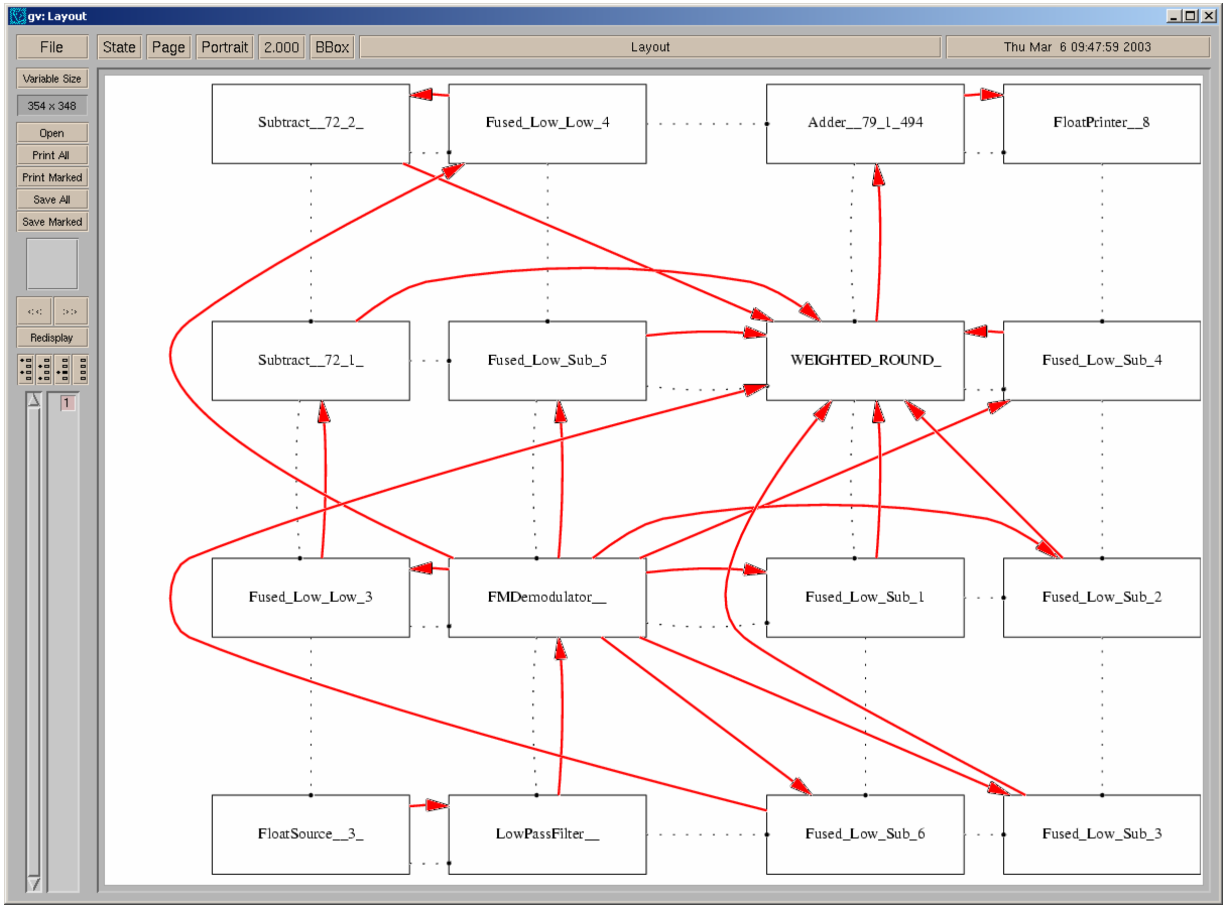
```
fm % dotty after.dot
```





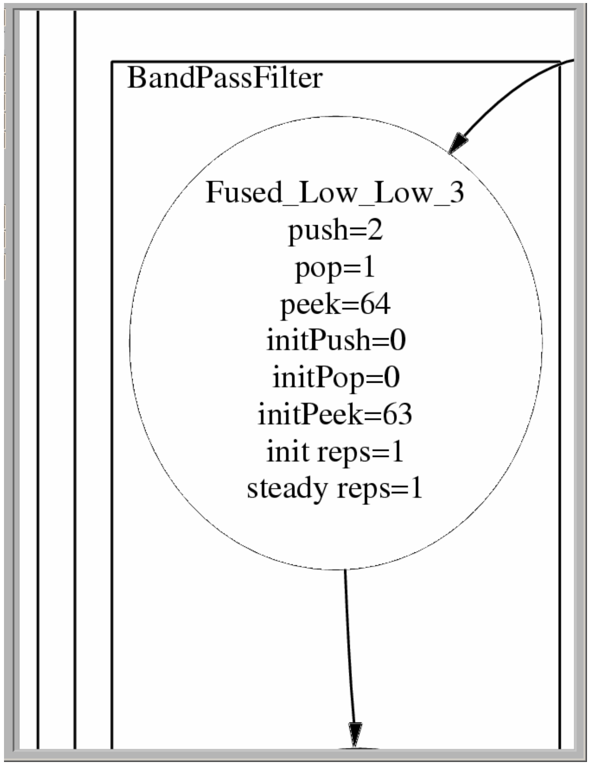
# Layout on Raw

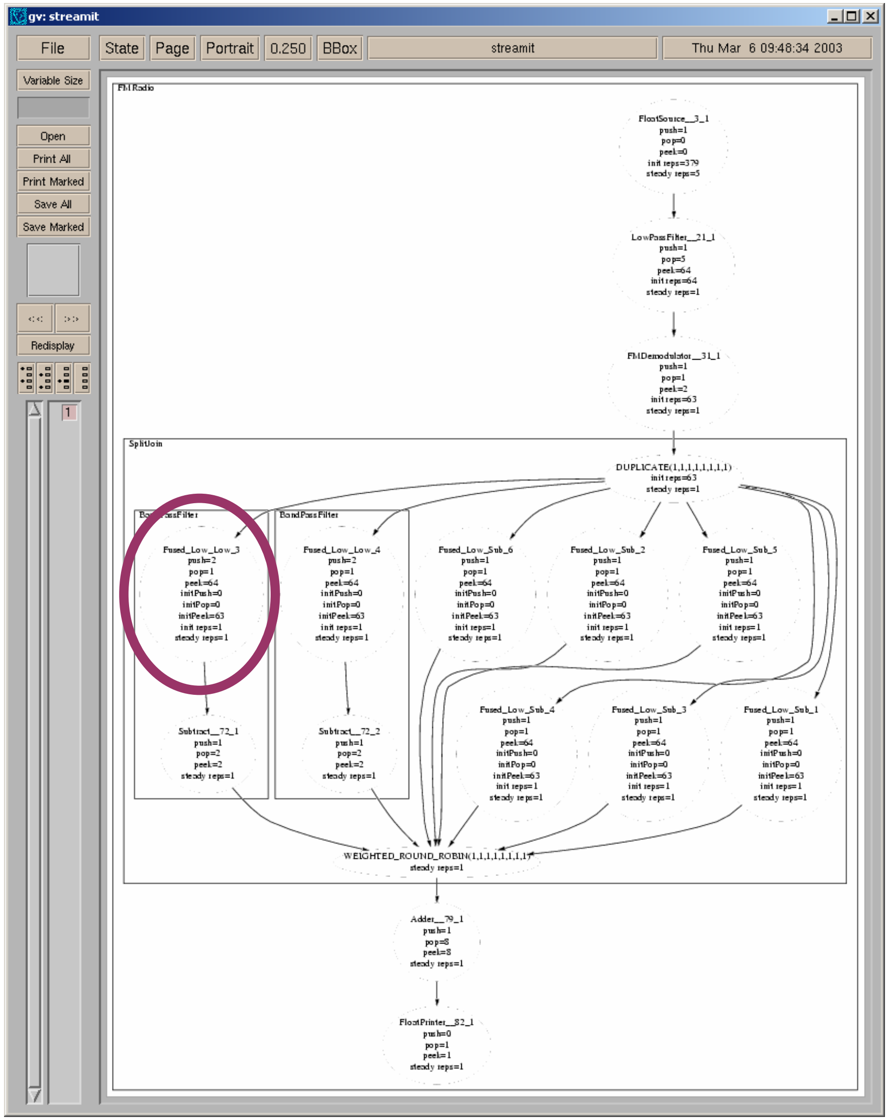
```
fm % dotty layout.dot
```



# Initial and Steady-State Schedule

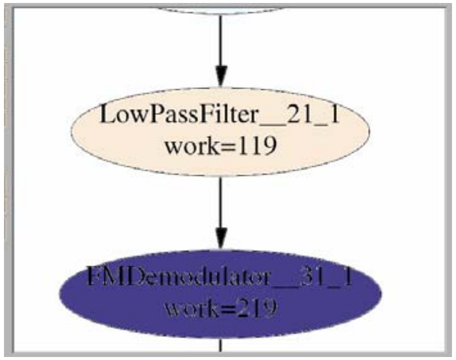
```
fm % dotty schedule.dot
```

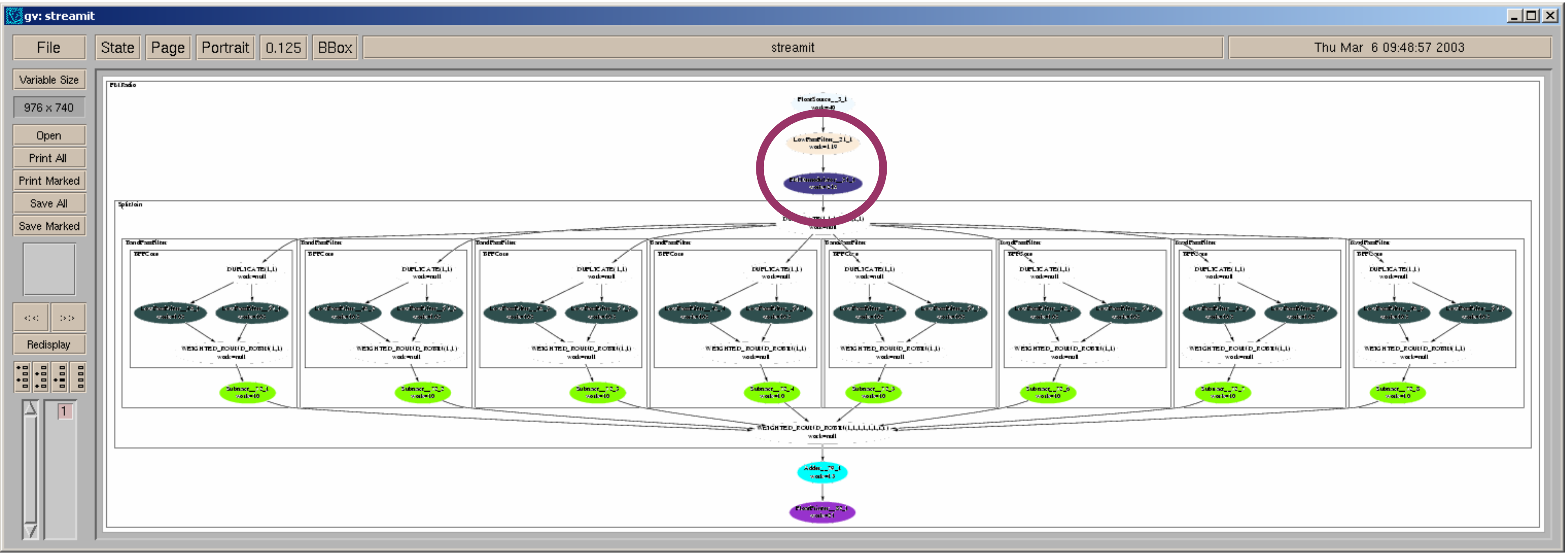




# Work Estimates (Graph)

```
fm % dotty work-before.dot
```





# Work Estimates (Table)

```
fm % cat work-before.txt
Filter               Reps  Measured Work  Estimated Work  (Measured-Estimated)/Measured  Total Measured Work
FMDemodulator__31       1            219             219                              0                  219
LowPassFilter__21       1            119             119                              0                  119
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
LowPassFilter__49       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
LowPassFilter__67       1            103             103                              0                  103
FloatSource__3          5              8               8                              0                   40
```

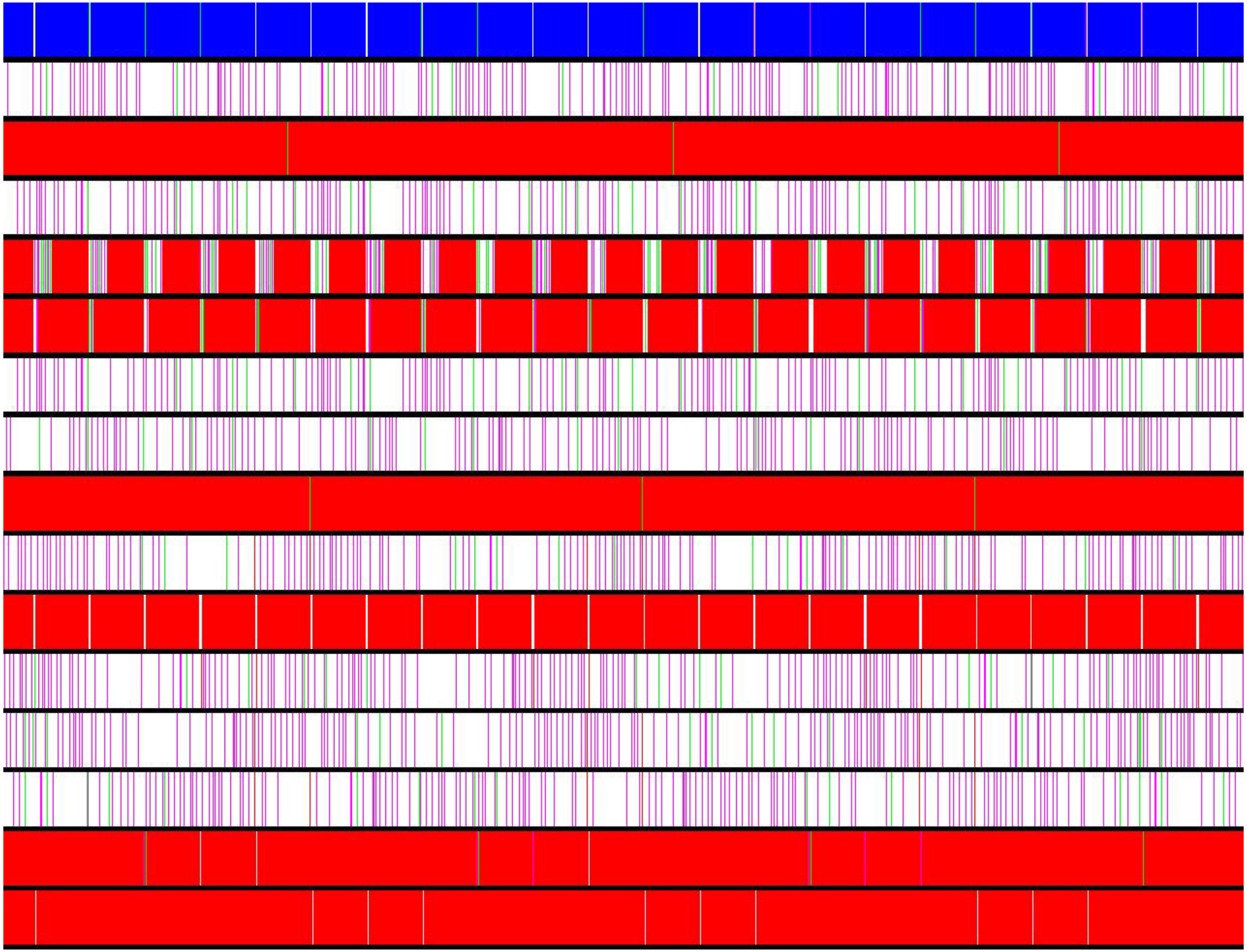
# Collected Results

```
fm % cat results.out

 Performance Results

 Tiles in configuration: 16
 Tiles assigned (to filters or joiners): 16
 Run for 10 steady state cycles.
 With 0 items skipped for init.
 With 1 items printed per steady state.

 cycles MFLOPS work_count
 --------------------------
 2153 350 19227
 2220 347 19731
 2229 310 18963
 2229 291 18512
```

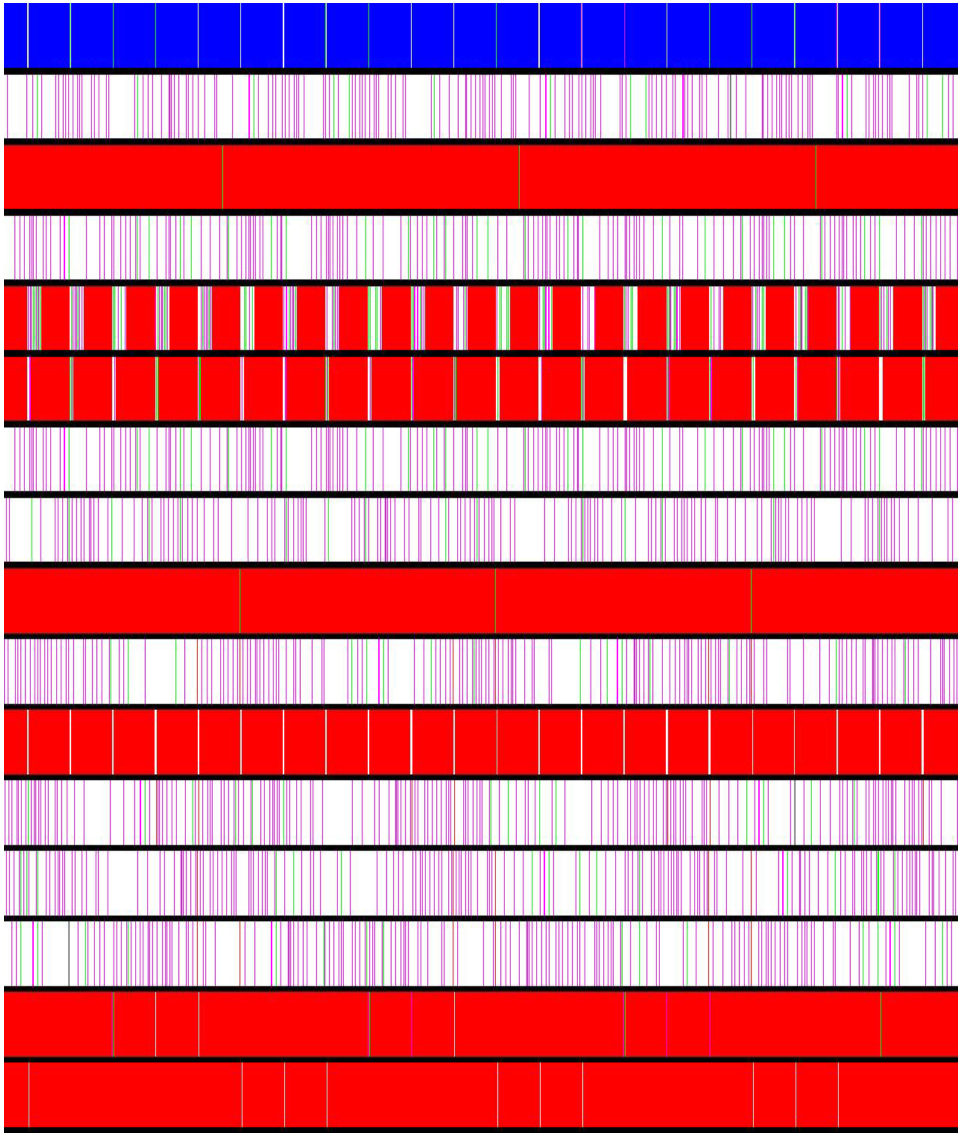
# Collected Results

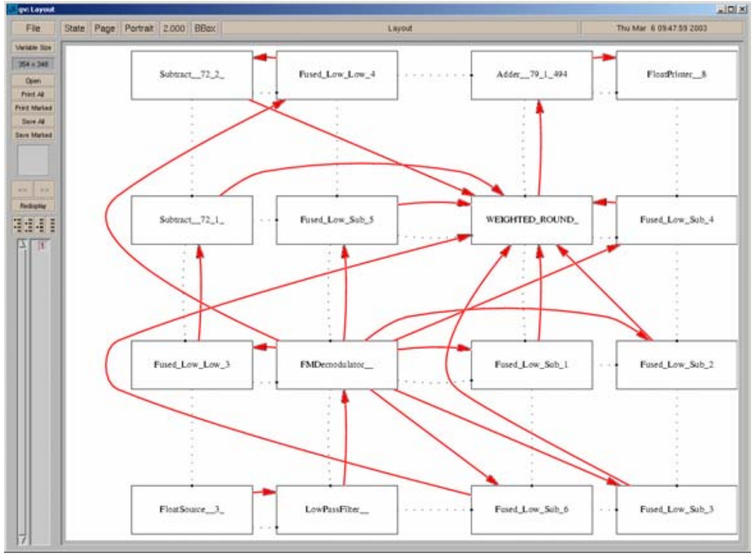
```
fm % cat results.out

 Performance Results

 Tiles in configuration: 16
 Tiles assigned (to filters or joiners): 16
 Run for 10 steady state cycles.
 With 0 items skipped for init.
 With 1 items printed per steady state.

 cycles MFLOPS work_count
 -------------------------
 2153 350 19227
 2220 347 19731
 2229 310 18963
 2229 291 18512
 2229 292 18537
 2229 293 18559
 2229 291 18513
 2229 292 18557
 2229 289 18510
 2229 291 18530

 Summmary:
 Steady State Executions: 10
 Total Cycles: 22205
 Avg Cycles per Steady-State: 2220
 Thruput per 10^5: 45
 Avg MFLOPS: 304
 workCount* = 187639 / 355280
```

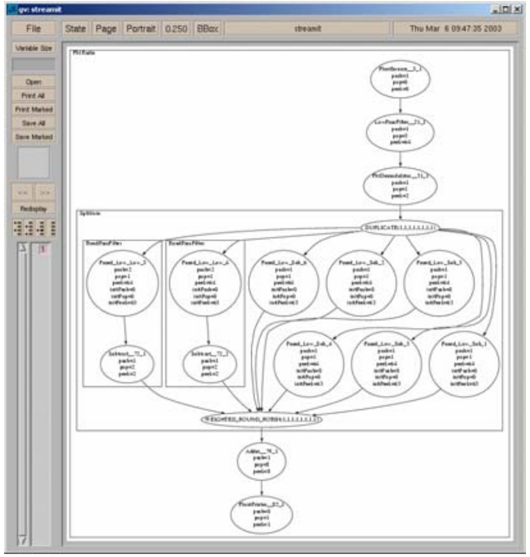
# Understanding Performance



# Understanding Performance







# Demo:  Linear Optimization

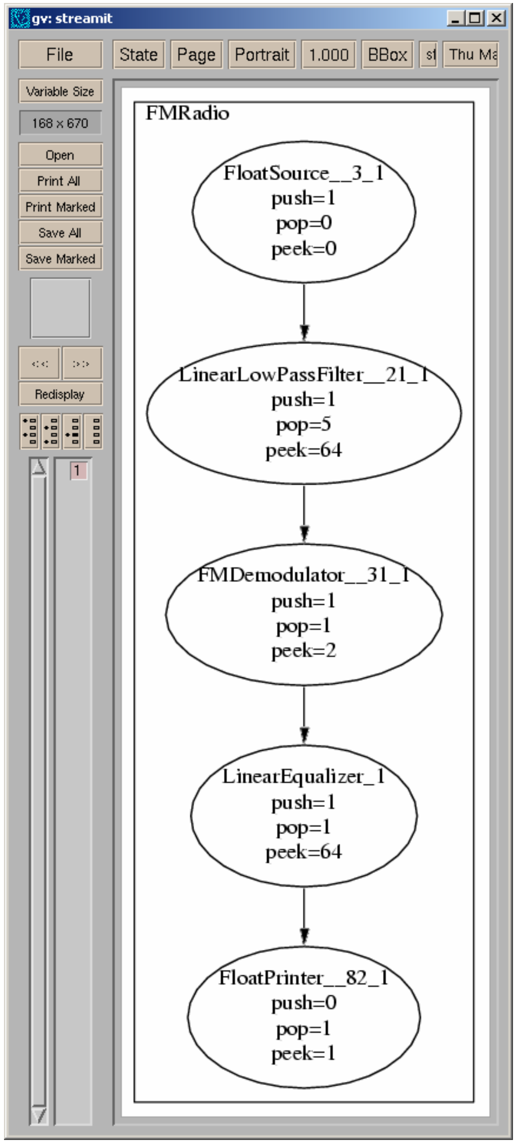
```
fm % knit --linearreplacement
          --raw 4 --numbers 10 FMRadio.str
fm % make -f Makefile.streamit run
```

New option:

--linearreplacement identifies filters which compute linear functions of their input, and replaces adjacent linear nodes with a single matrix-multiply

# Stream Graph Before Partitioning

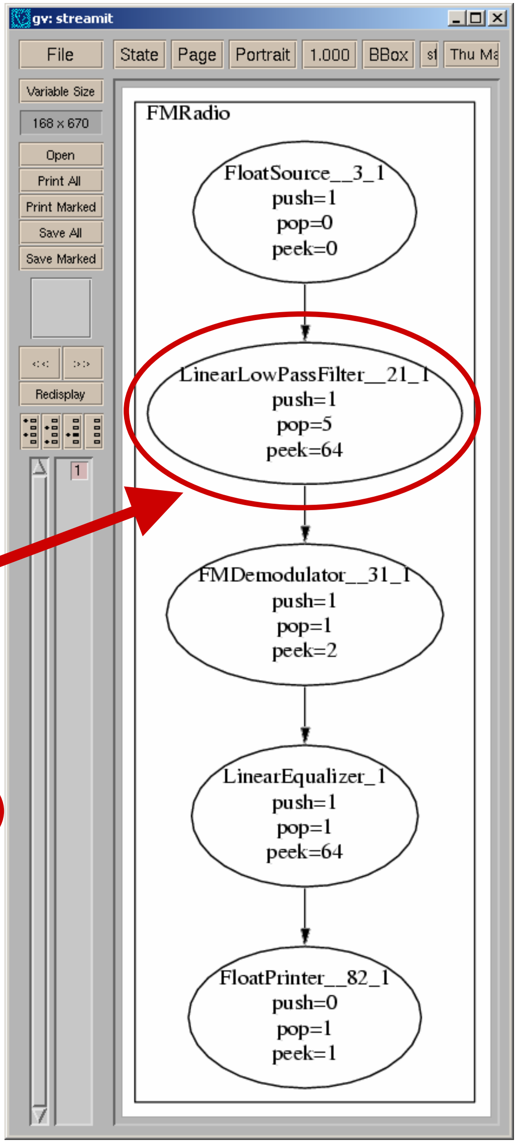
```
fm % dotty before.dot
```

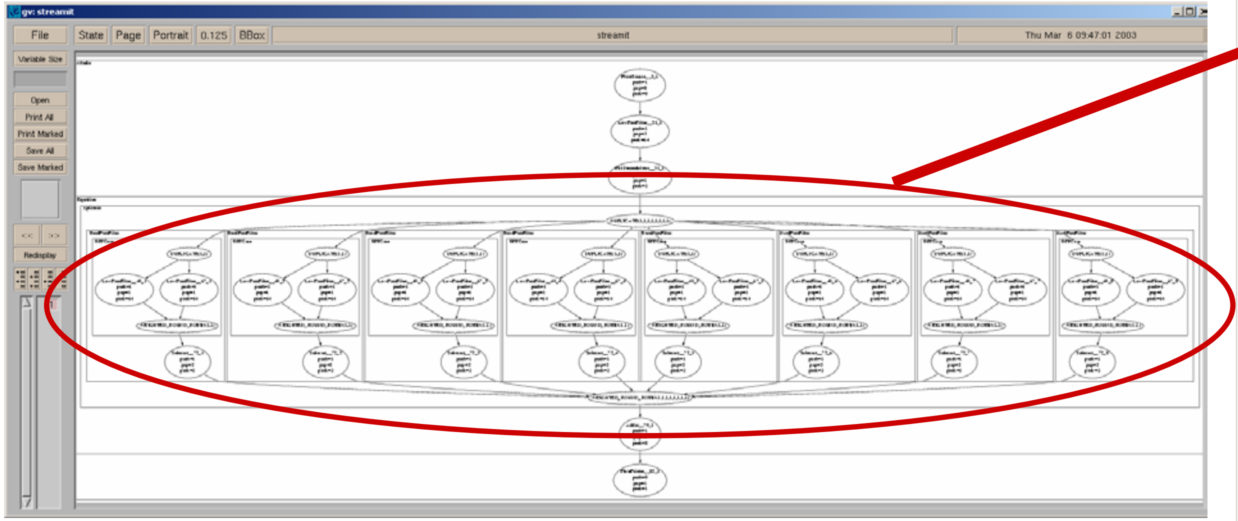


# Stream Graph Before Partitioning

```
fm % dotty before.dot
```

Entire Equalizer collapsed!





*without linear replacement*

# Results with Linear Optimization

```
fm % cat results.out
```

```
Summmary:
Steady State Executions: 10
Total Cycles: 7260
Avg Cycles per Steady-State: 726
Thruput per 10^5: 137
Avg MFLOPS: 128
workCount* = 15724 / 116160
```

# Results with Linear Optimization

```
fm % cat results.out
```

```
Summmary:
Steady State Executions: 10
Total Cycles: 7260
Avg Cycles per Steady-State: 726
Thruput per 10^5: 137
Avg MFLOPS: 128
workCount* = 15724 / 116160
```

Speedup by factor of 3

# Results with Linear Optimization

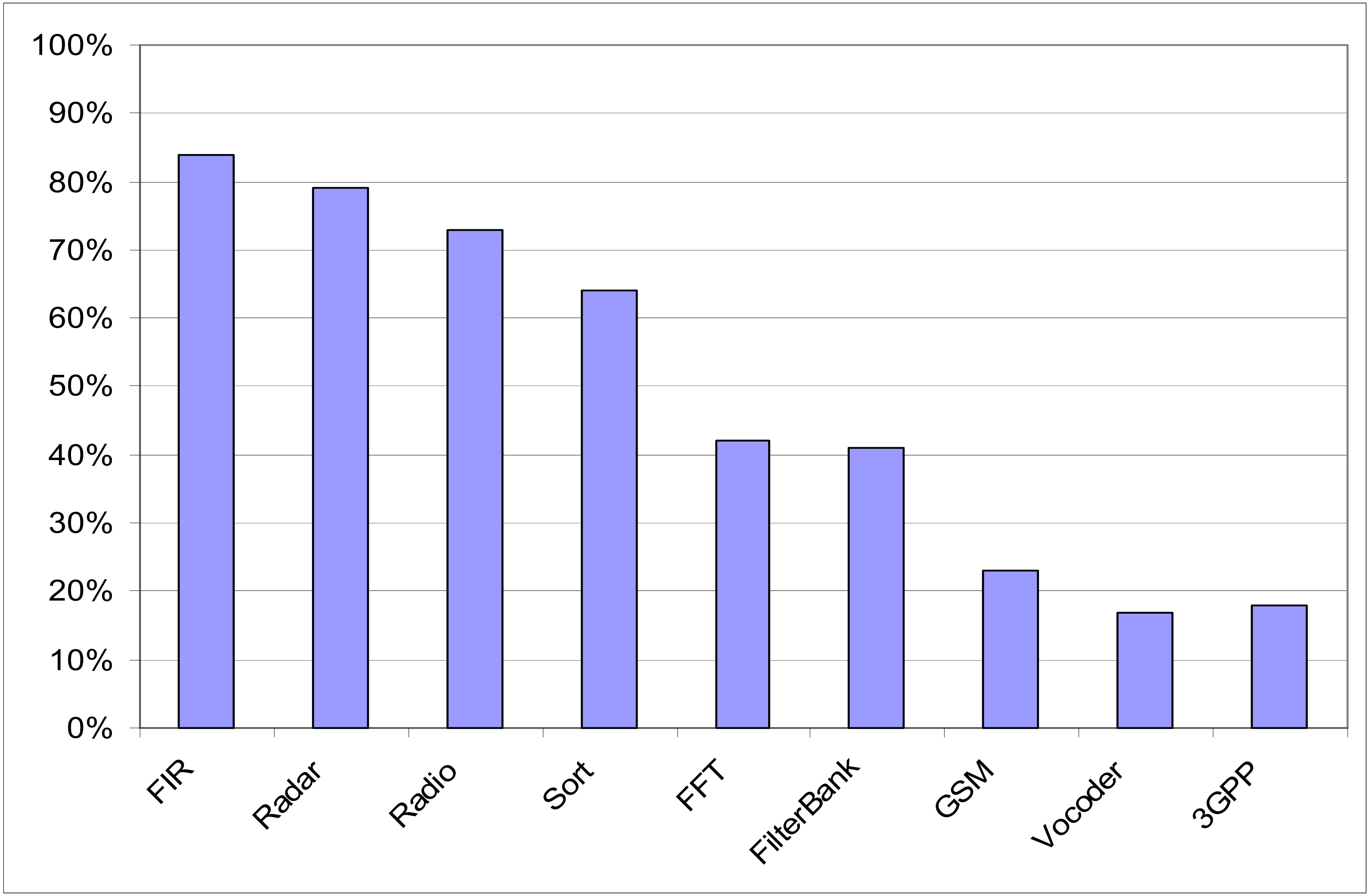
```
fm % cat results.out
```

```
Summmary:
Steady State Executions: 10
Total Cycles: 7260
Avg Cycles per Steady-State: 726
Thruput per 10^5: 137
Avg MFLOPS: 128
workCount* = 15724 / 116160
```

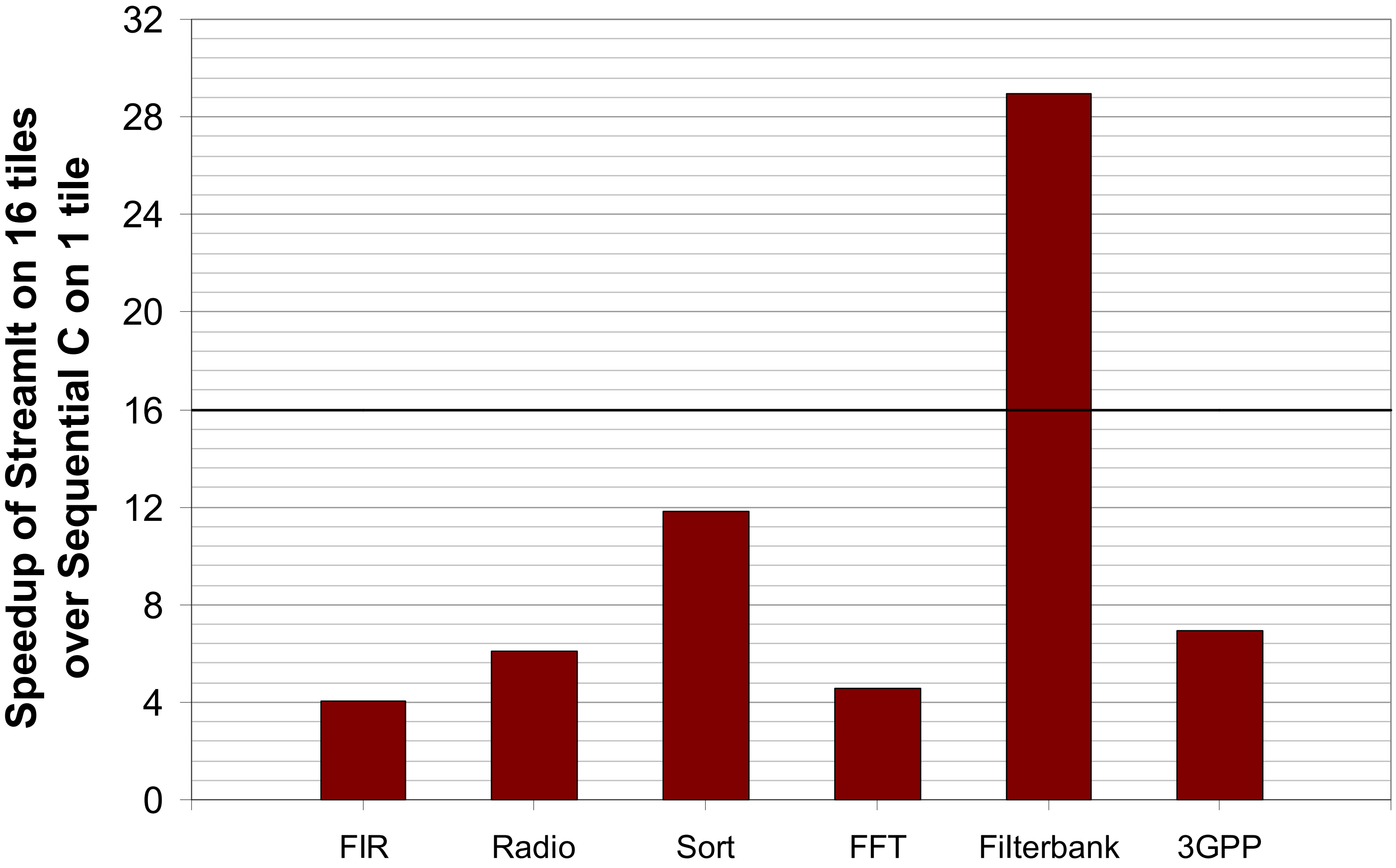
Speedup by factor of 3

Allows programmer to write simple, modular filters which compiler combines automatically

# Other Results: Processor Utilization



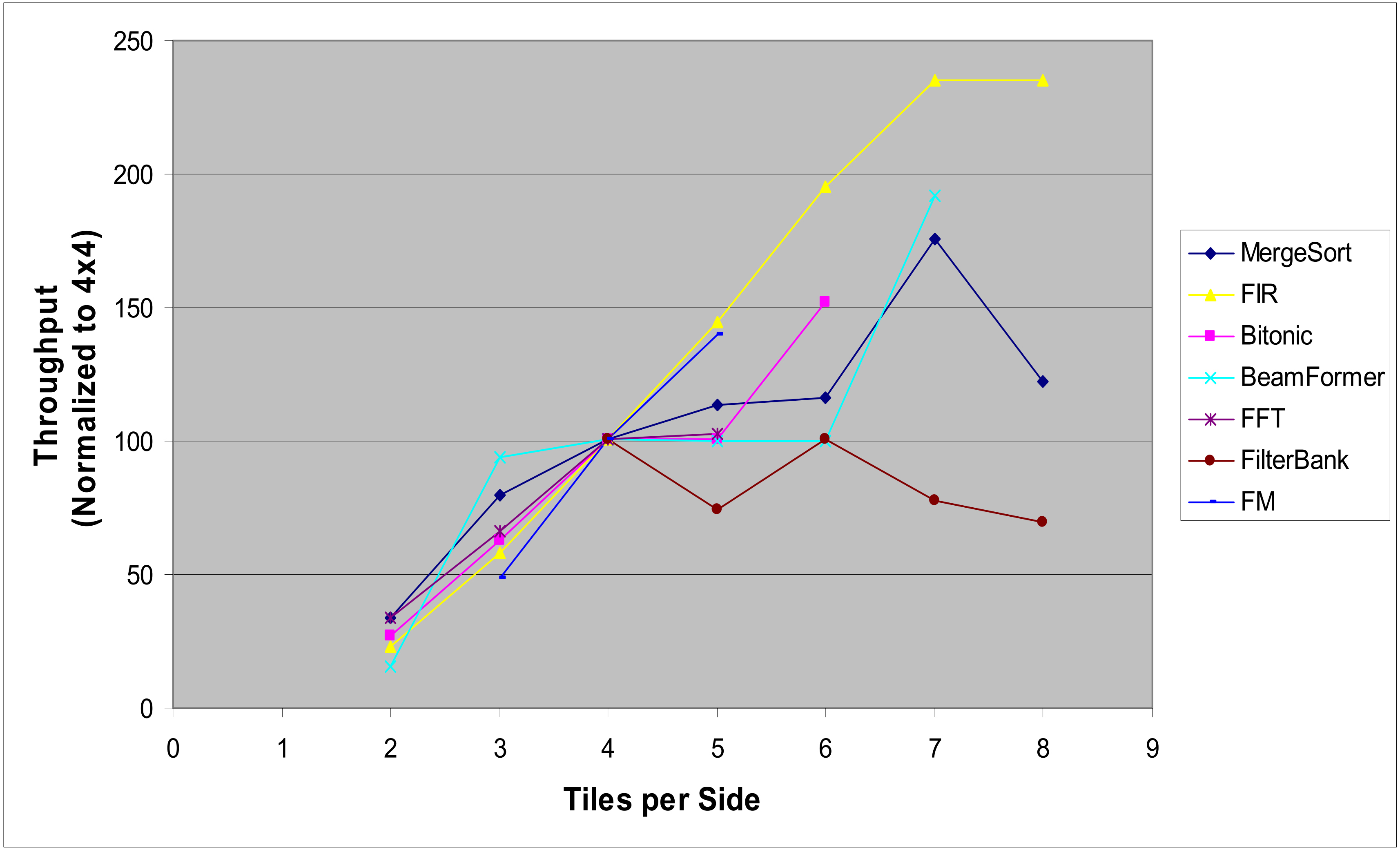
# Speedup Over Single Tile



-For Radio we obtained the C implementation from a 3rd party

-For FIR, Sort, FFT, Filterbank, and 3GPP we wrote the C implementation following a reference algorithm.

# Scaling of Throughput



# Compiler Status

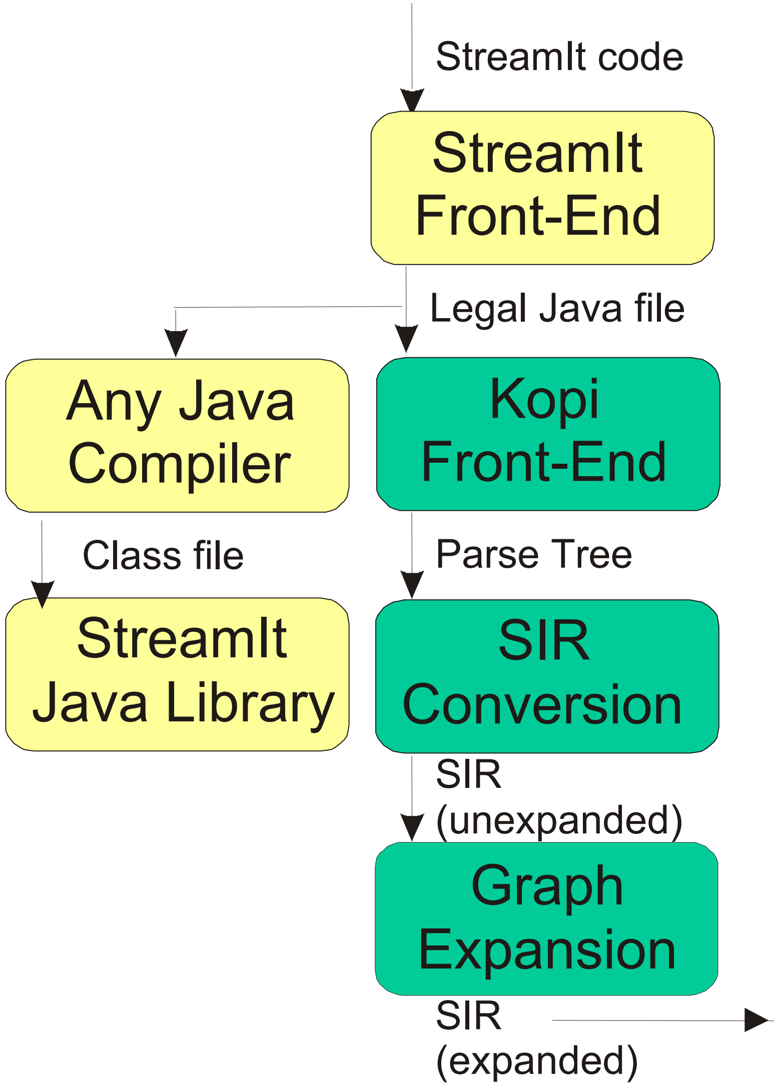
- Raw backend has been working for more than a year
  - Robust partitioning, layout, and scheduling
  - Still working on improvements:
    - Dynamic programming partitioner
    - Optimized scheduling, routing, code generation
- Frontend is relatively new
  - Semantic checker still in progress
  - Some malformed inputs cause Exceptions
- We are eager to gain user feedback!

# Library Support

Option: `--library`

Run with Java library, not the compiler. Greatly facilitates application development, debugging, and verification.

Given File.str, the frontend will produce File.java, which you can edit and instrument like a normal Java file.



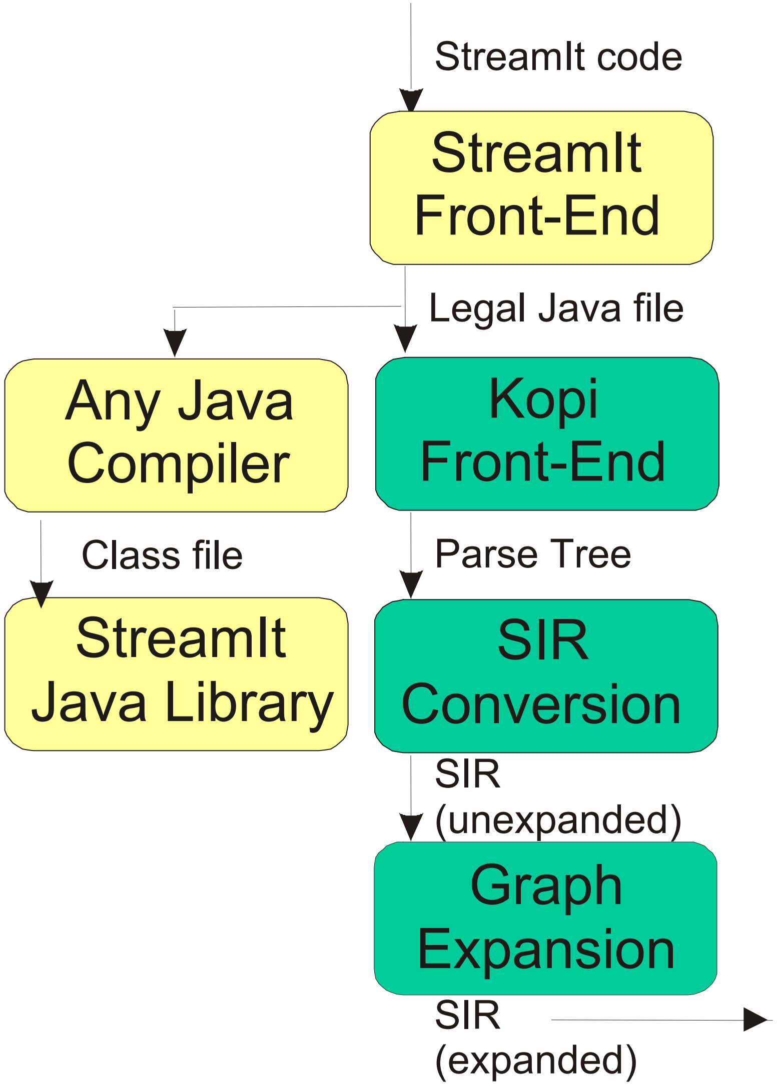
# Library Support

Option: `--library`

Run with Java library, not the compiler. Greatly facilitates application development, debugging, and verification.

Given File.str, the frontend will produce File.java, which you can edit and instrument like a normal Java file.

Many more options will be documented in the release.



# Summary

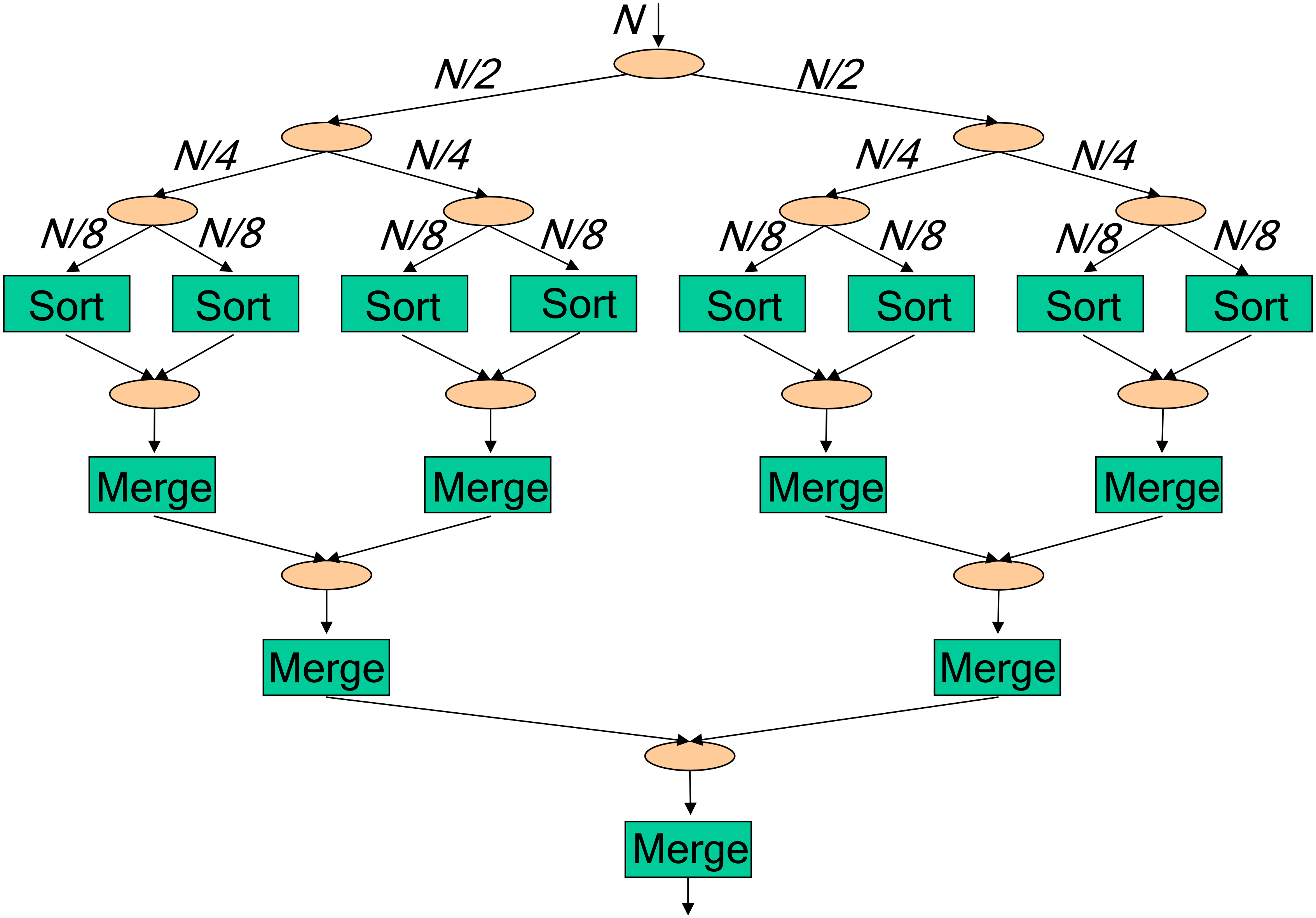
- Why use StreamIt?
  - High-level, architecture-independent syntax
  - Automatic partitioning, load balancing, layout, switch code generation, and buffer management
  - Aggressive domain-specific optimizations
  - Many graphical outputs for programmer
- Release by next Friday, 3/14/03

StreamIt Homepage

http://cag.lcs.mit.edu/streamit

# Backup Slides

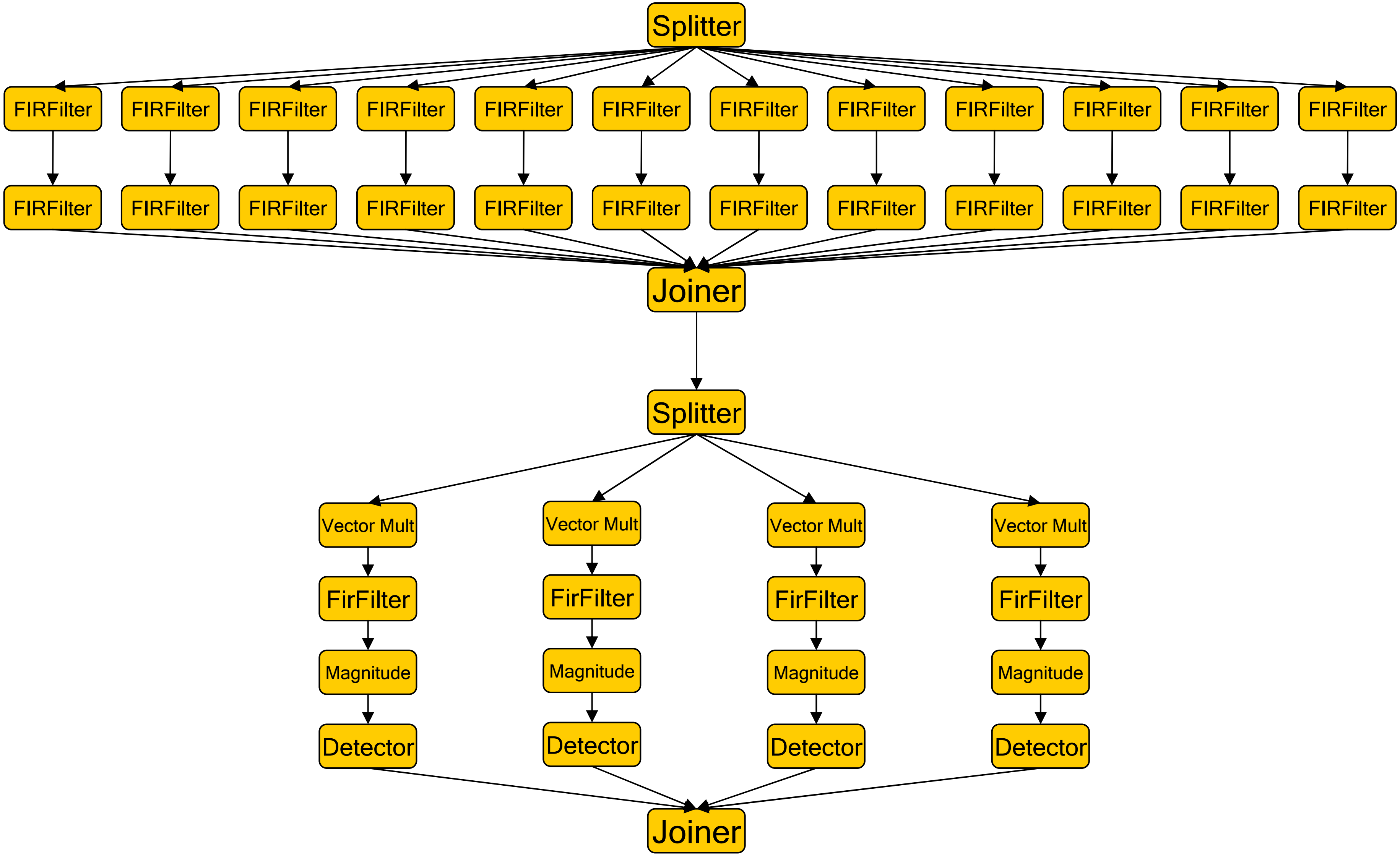
# N-Element Merge Sort (3-level)



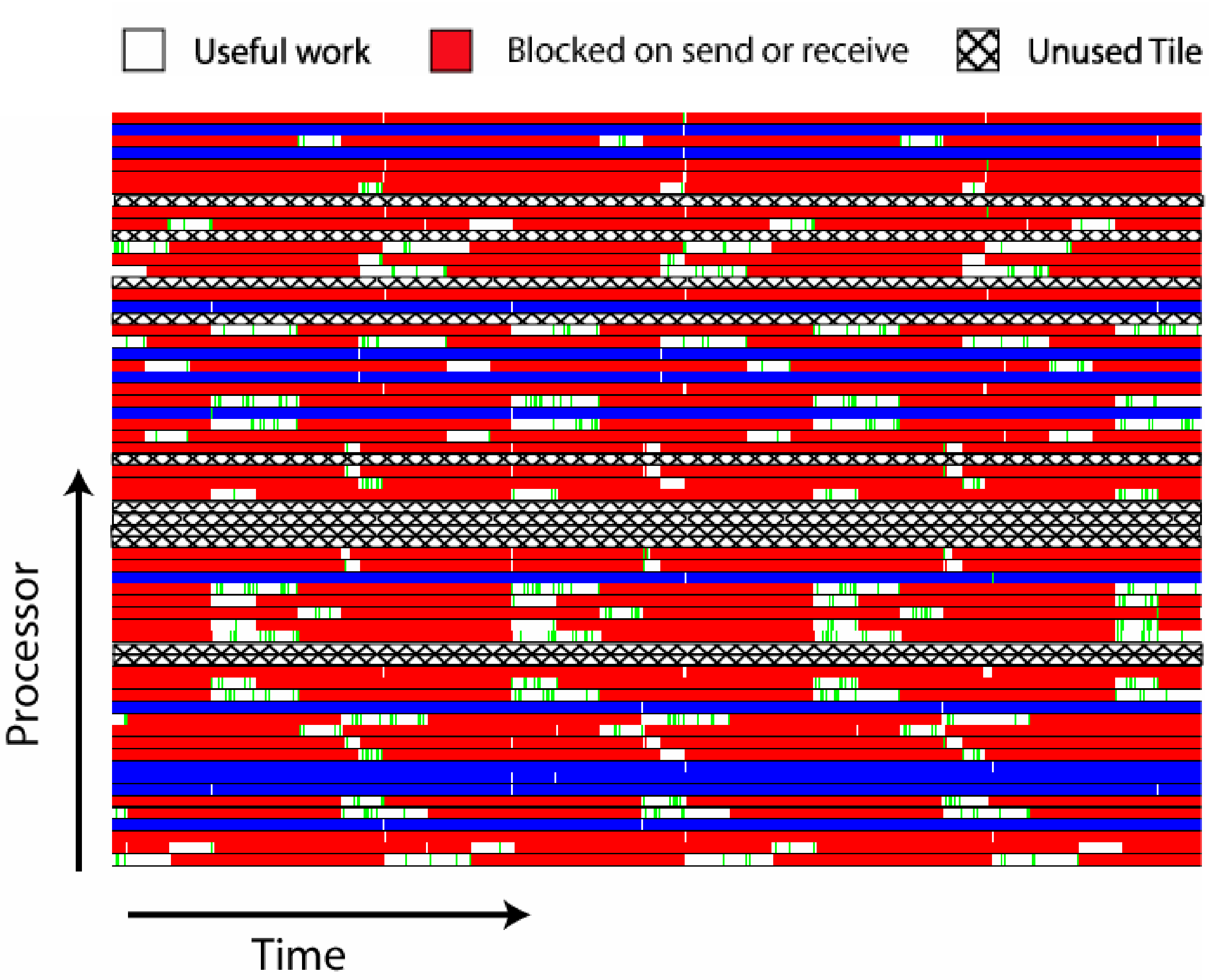
# N-Element Merge Sort (K-level)

```
pipeline MergeSort (int N, int K) {
    if (K==1) {
        add Sort(N);
    } else {
        add splitjoin {
            split roundrobin;
            add MergeSort(N/2, K-1);
            add MergeSort(N/2, K-1);
            joiner roundrobin;
        }
    }
    add Merge(N);
  }
}
```

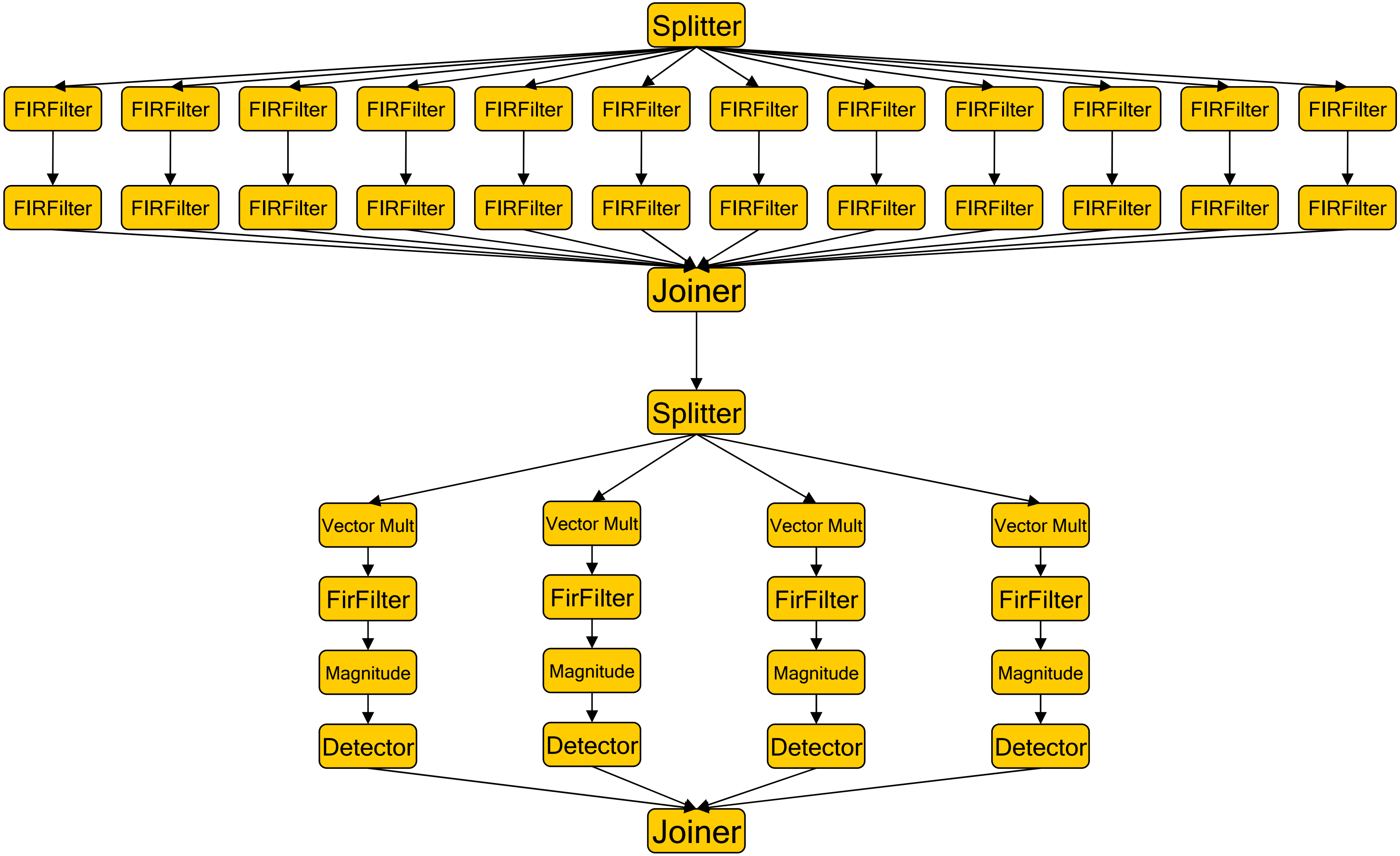
# Example: Radar App. (Original)



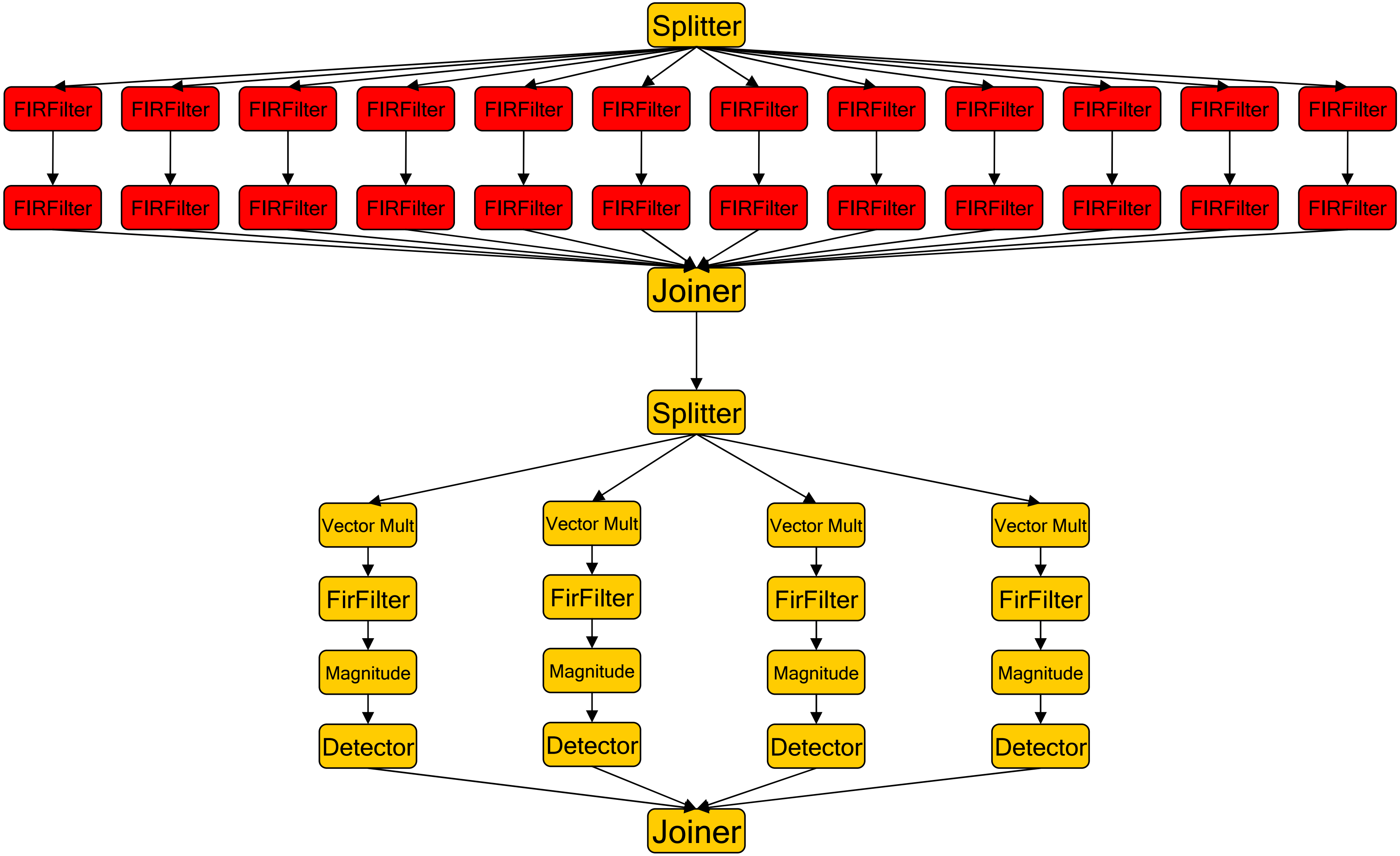
# Example: Radar App. (Original)



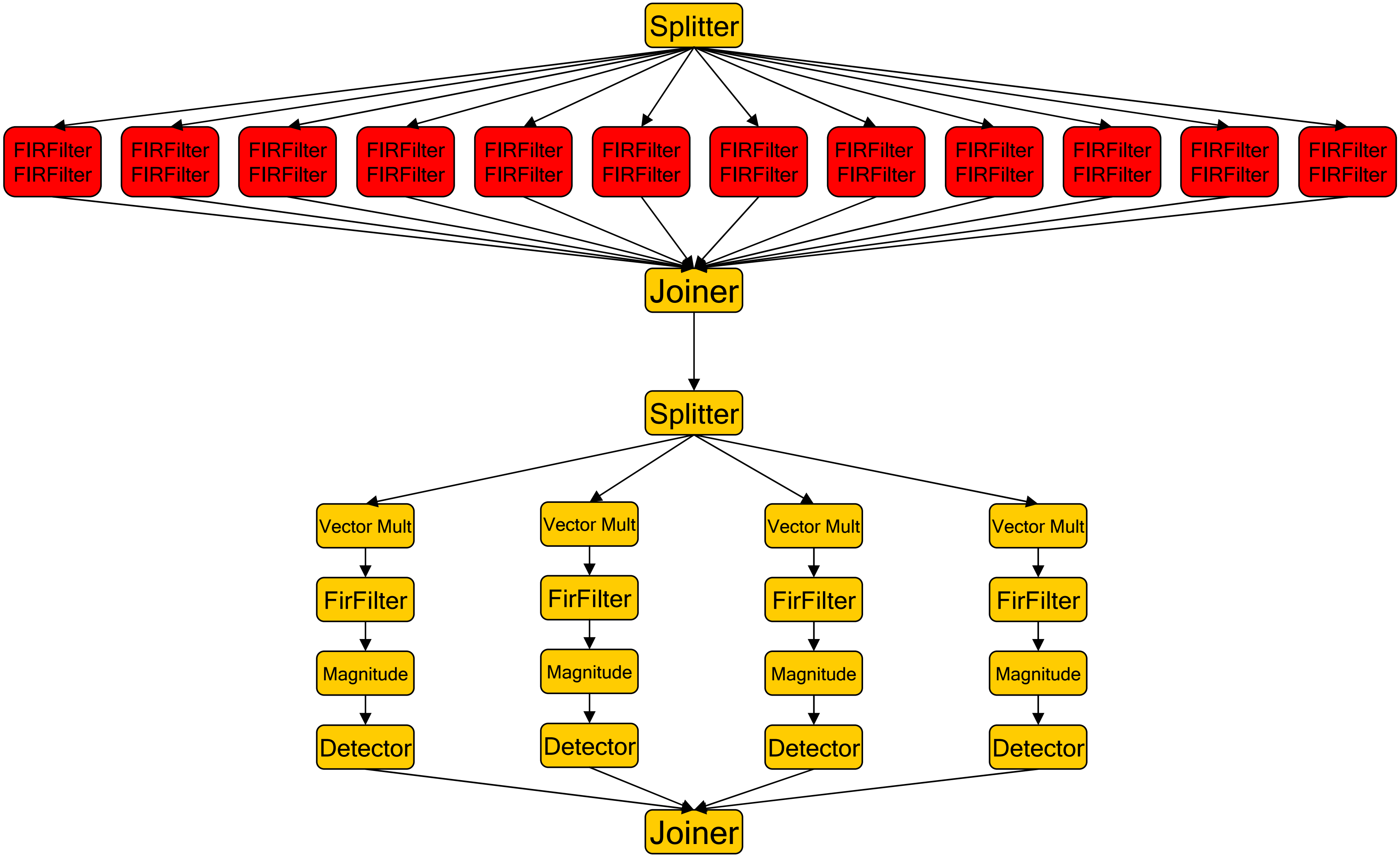
# Example: Radar App. (Original)



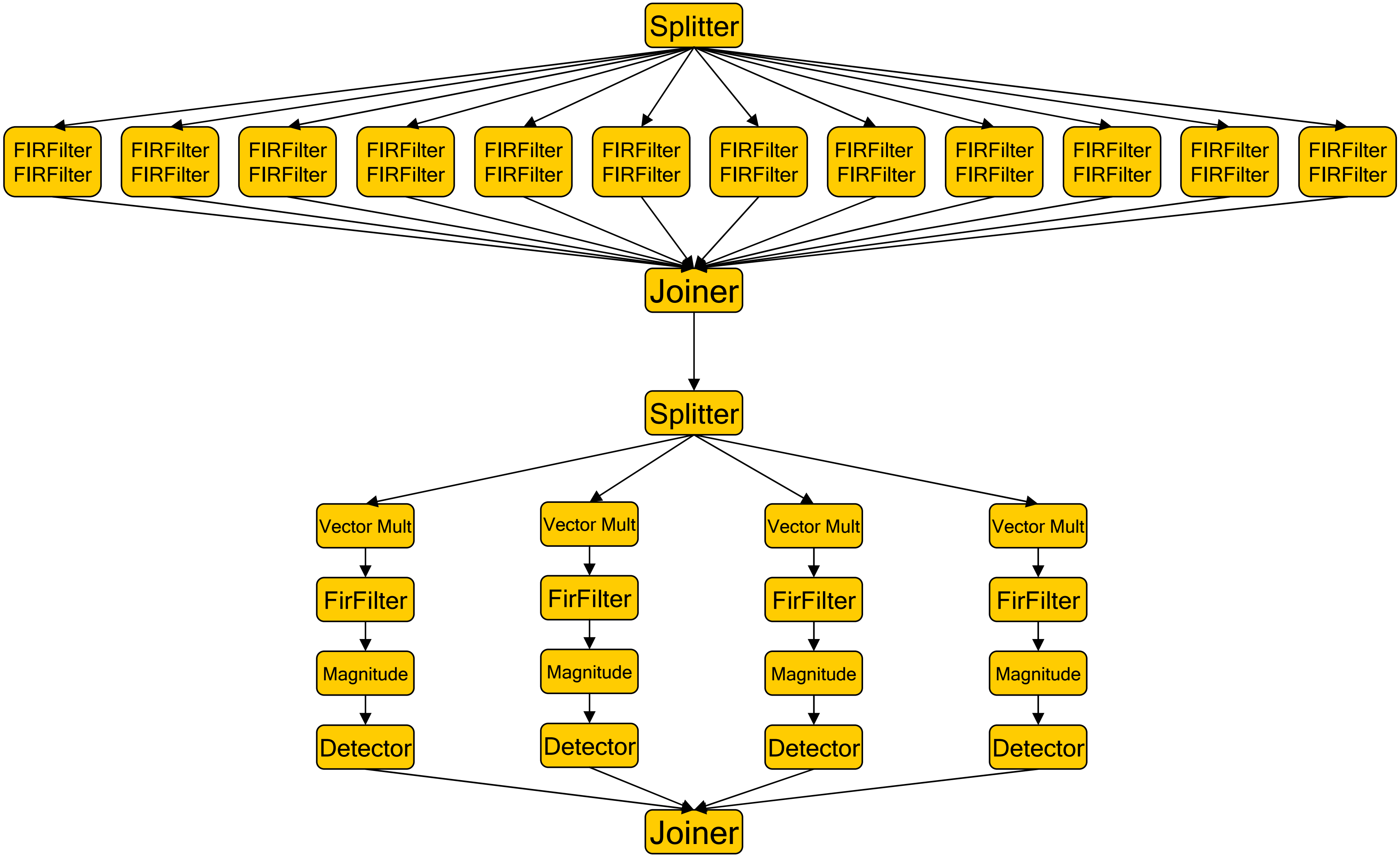
# Example: Radar App. (Original)



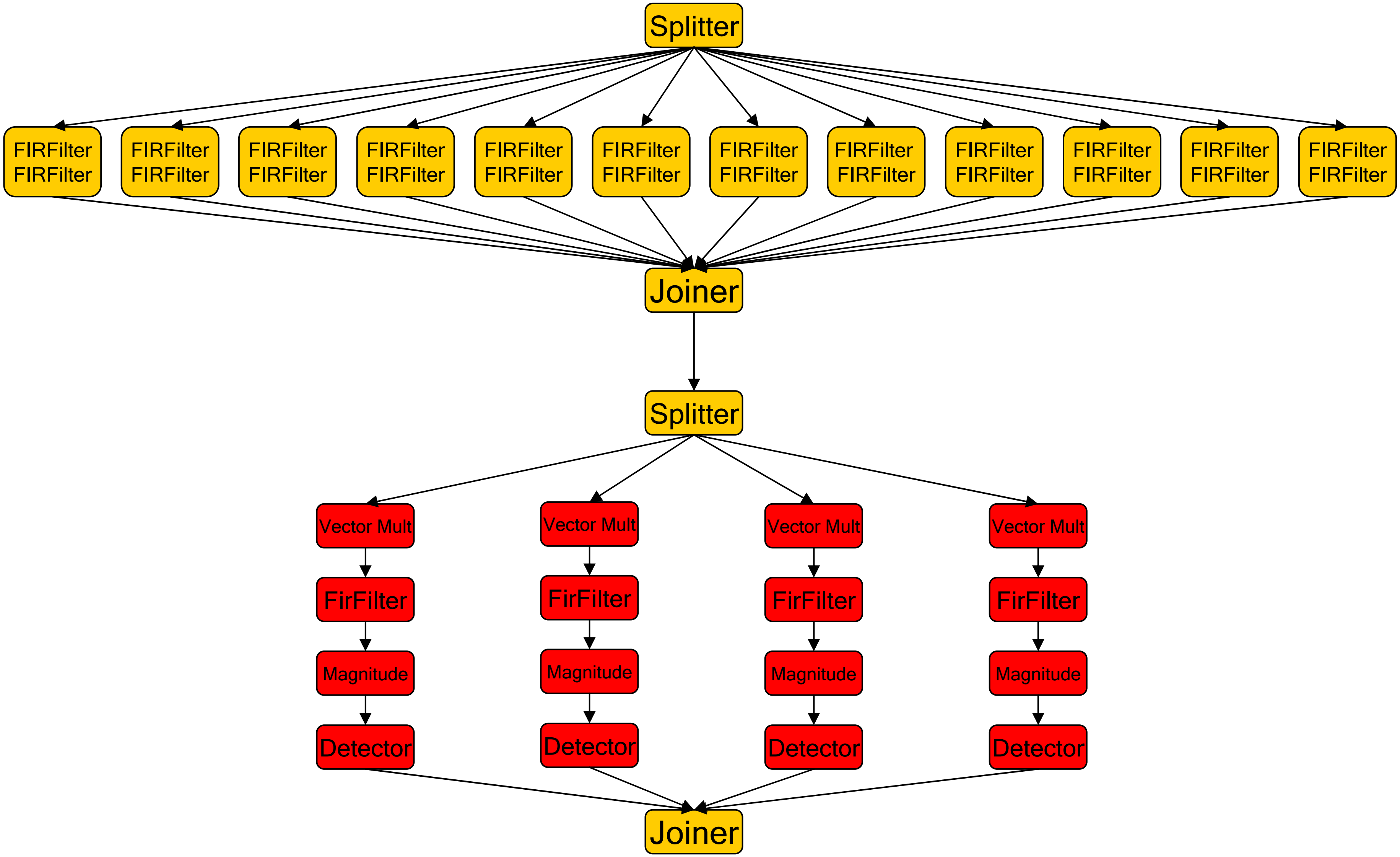
# Example: Radar App.



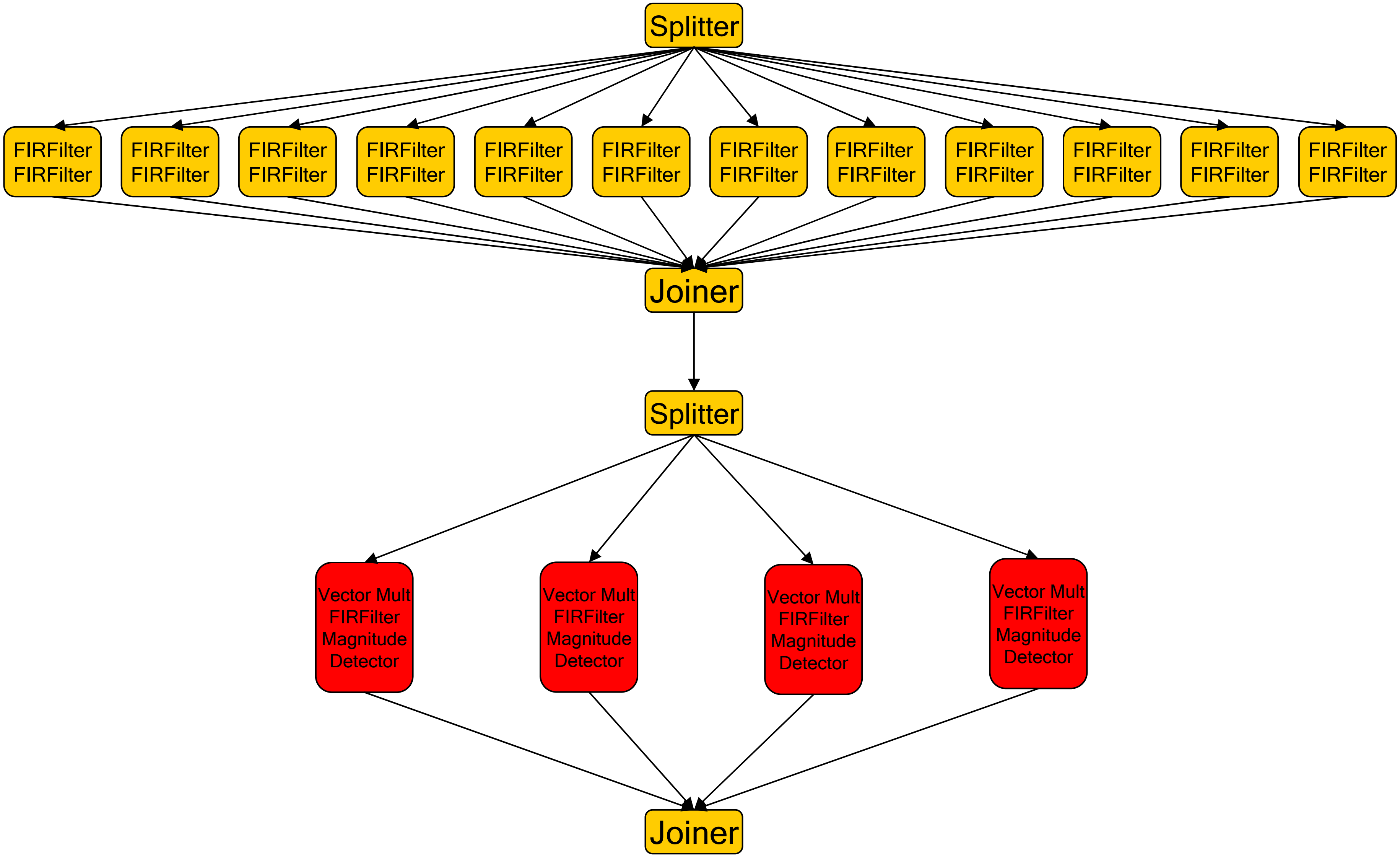
# Example: Radar App.



# Example: Radar App.



# Example: Radar App.



Splitter

FIRFilter<br>FIRFilter (×12)

| FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter | FIRFilter |

Joiner

Splitter

Vector Mult
FIRFilter
Magnitude
Detector

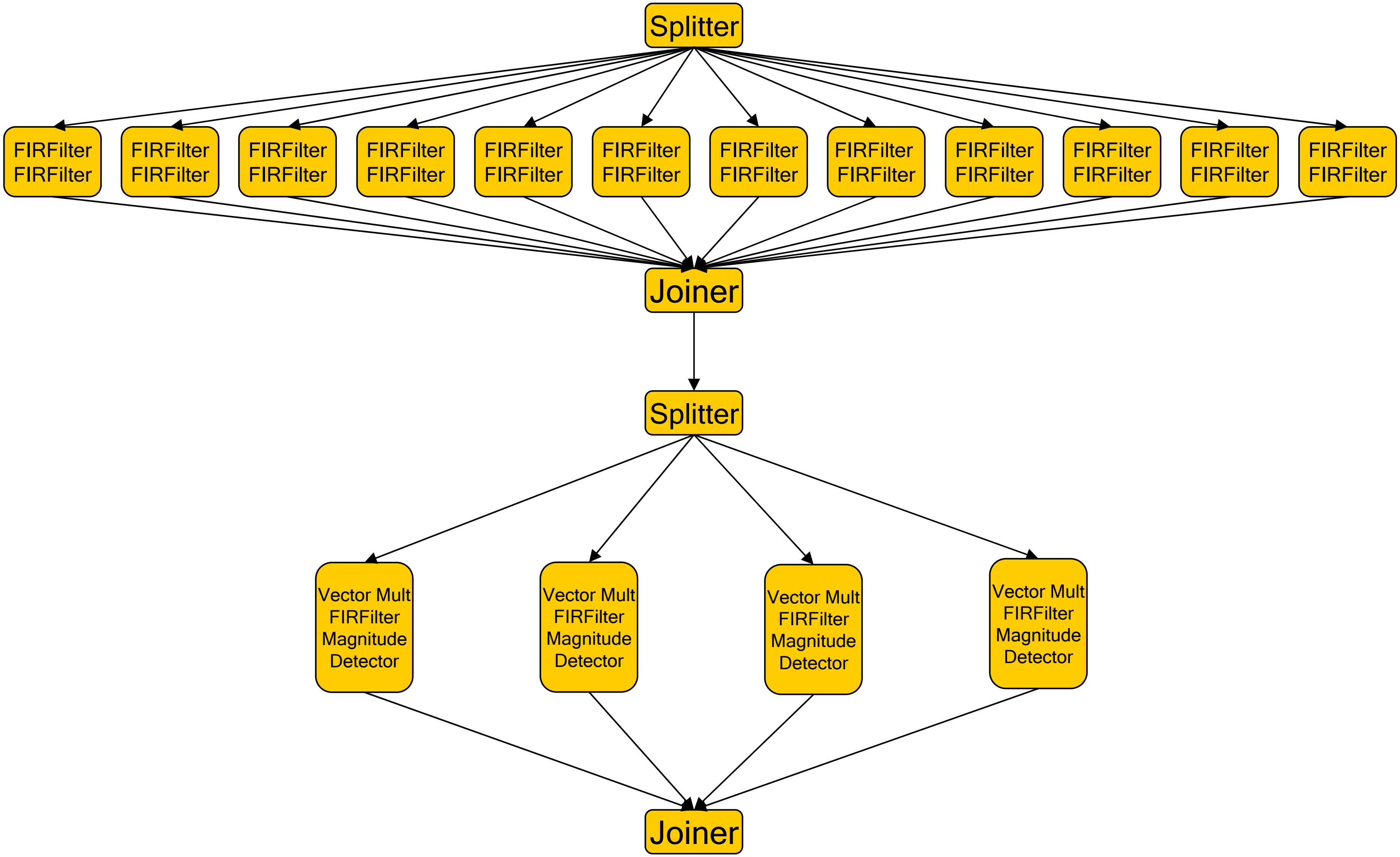
Vector Mult
FIRFilter
Magnitude
Detector

Vector Mult
FIRFilter
Magnitude
Detector

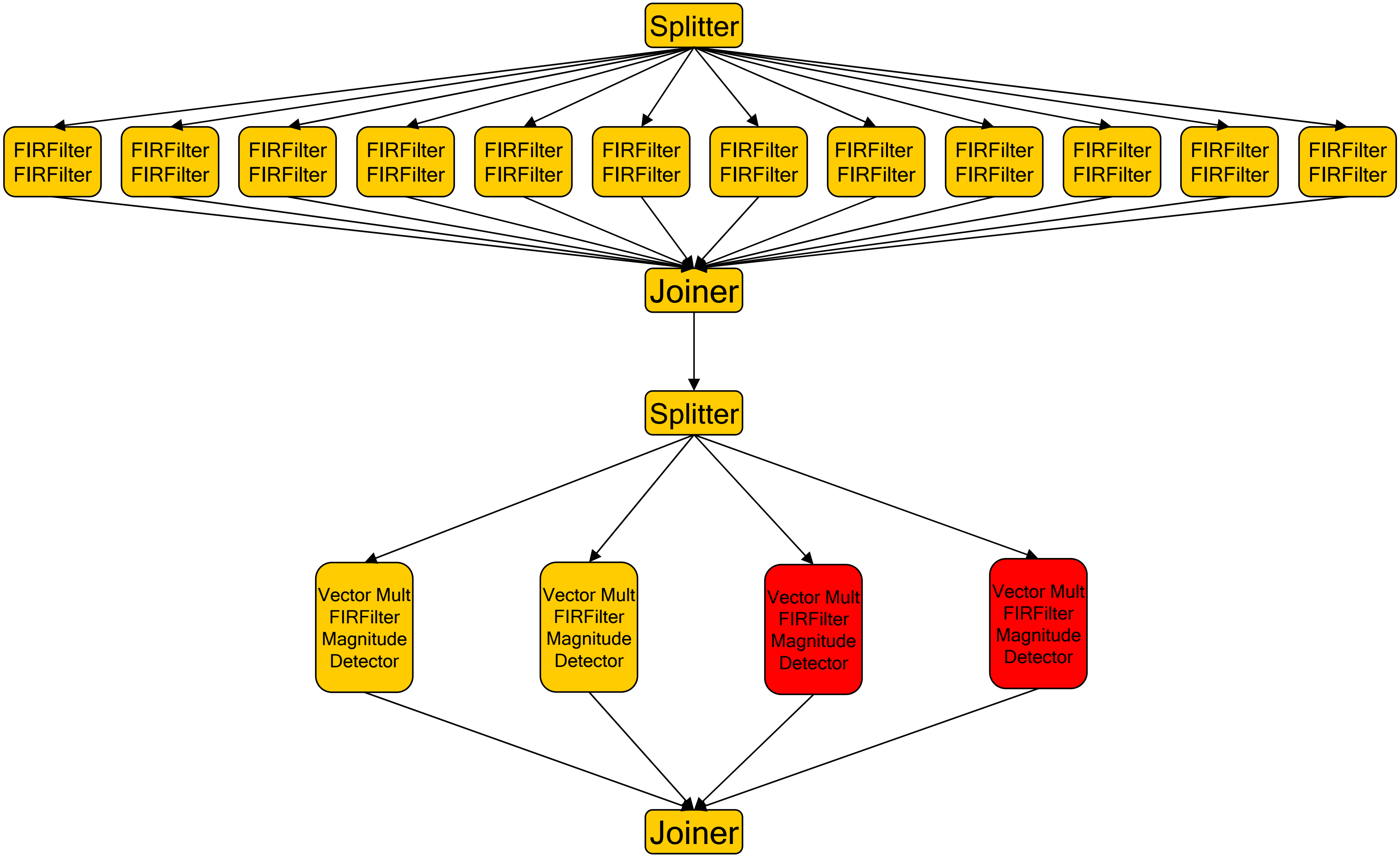
Vector Mult
FIRFilter
Magnitude
Detector

Joiner

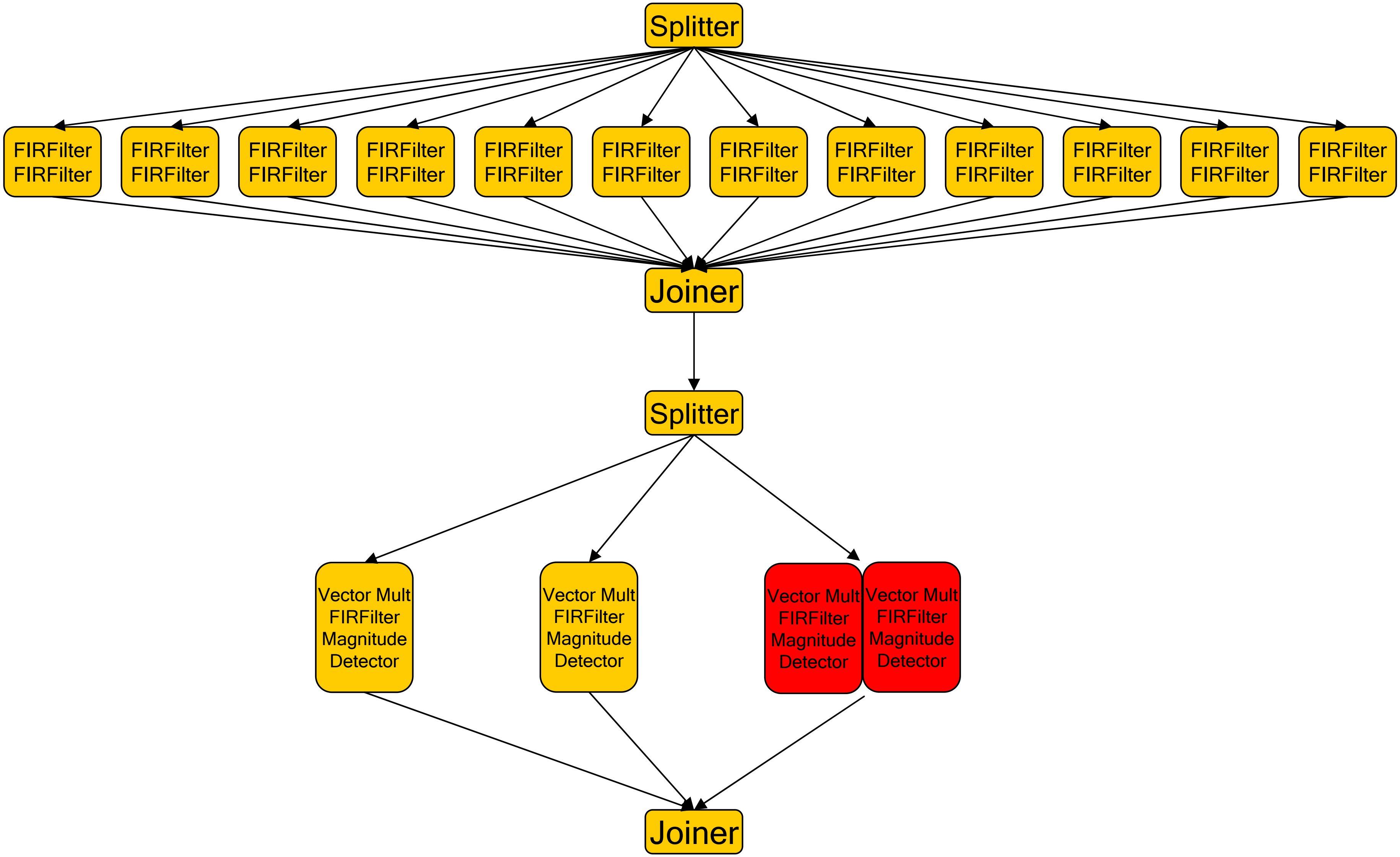
# Example: Radar App.



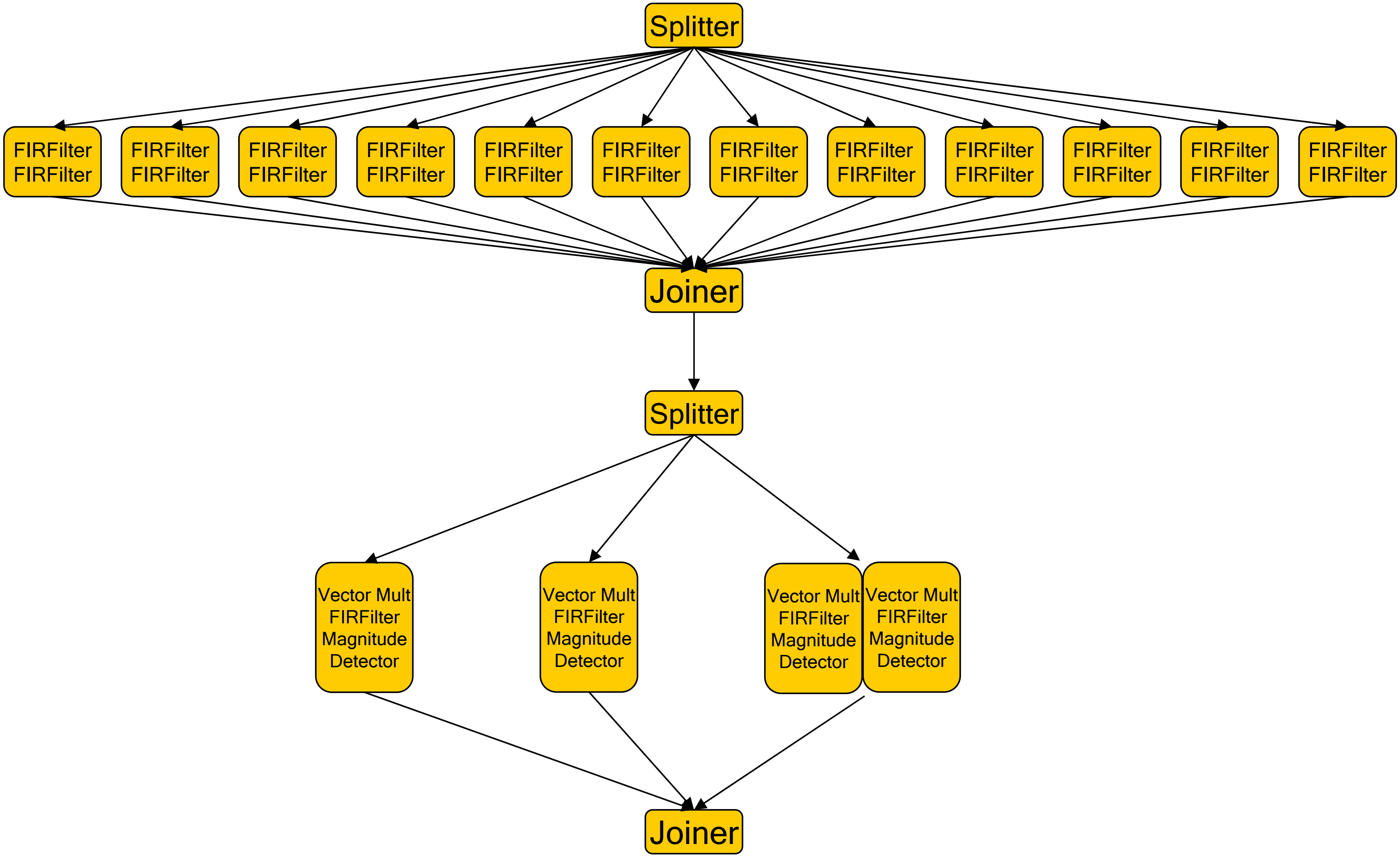
# Example: Radar App.



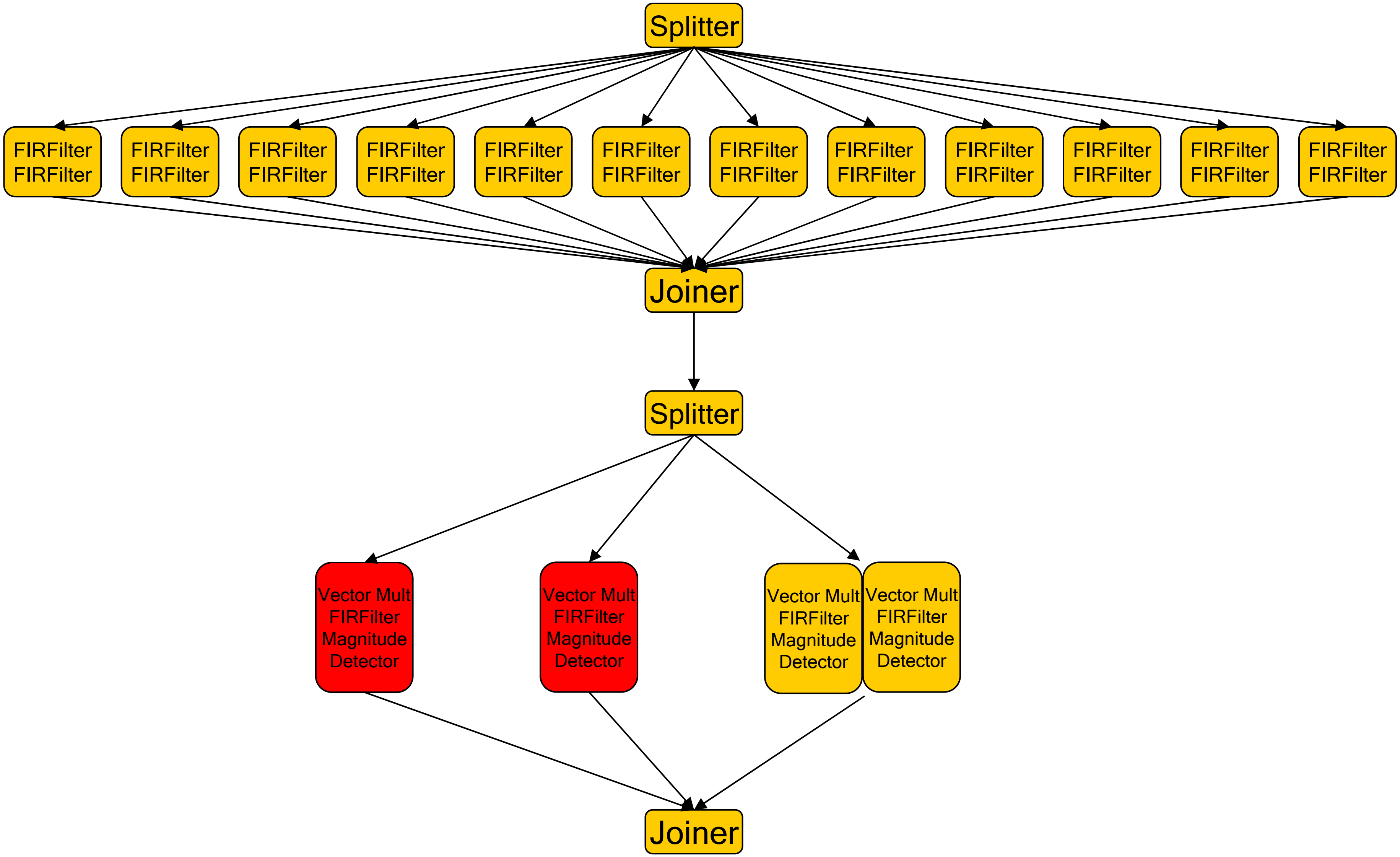
# Example: Radar App.



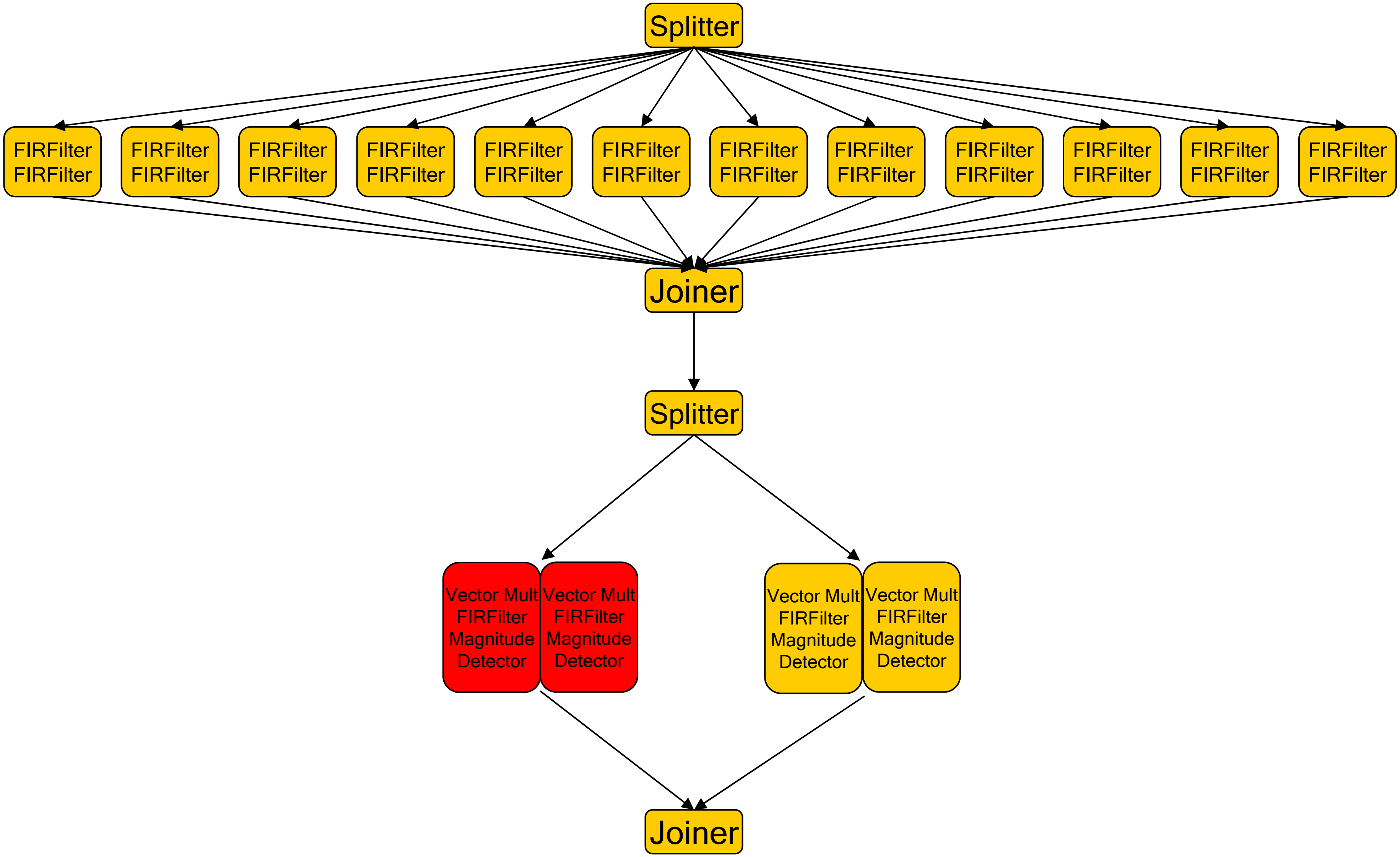
# Example: Radar App.



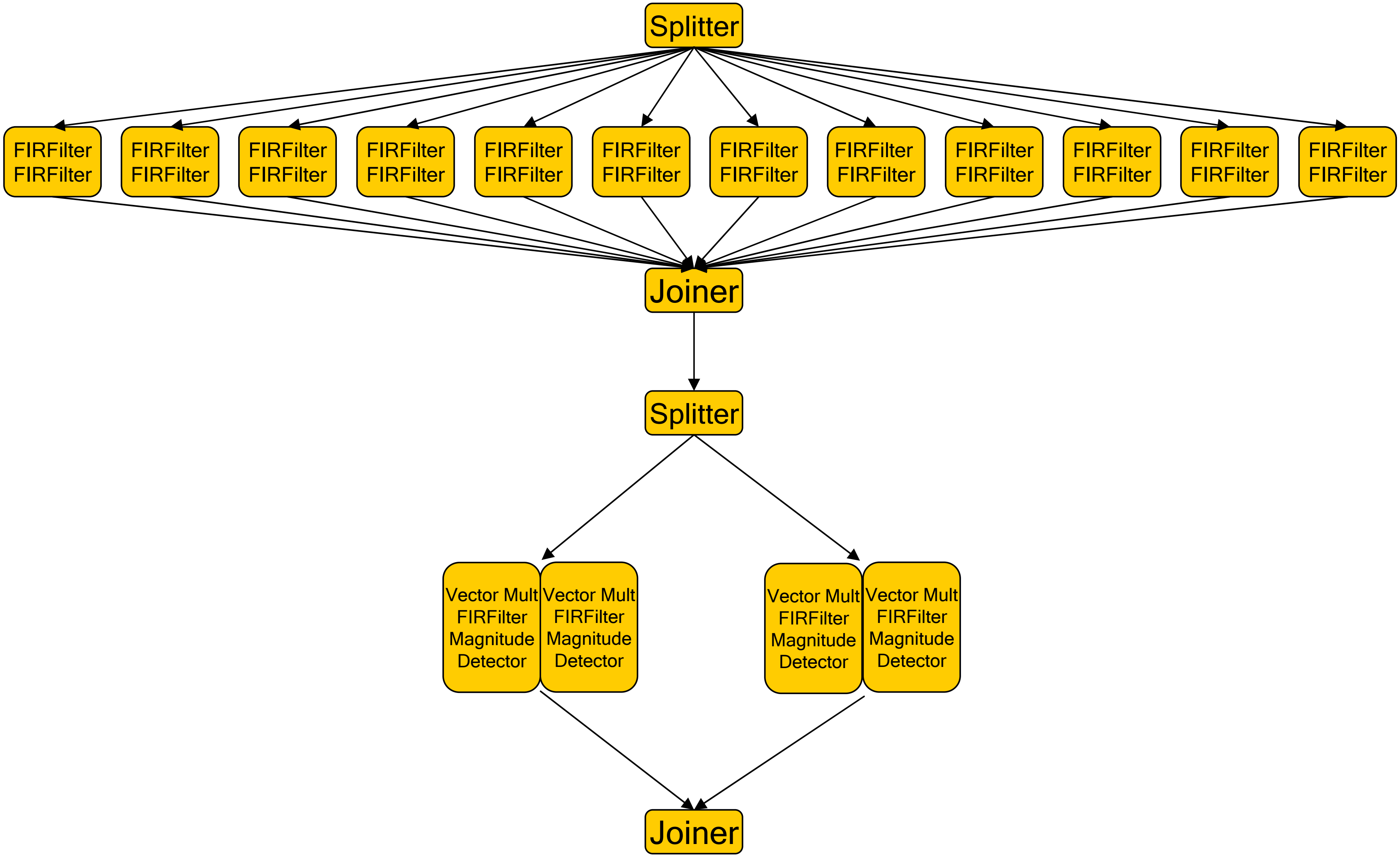
# Example: Radar App.



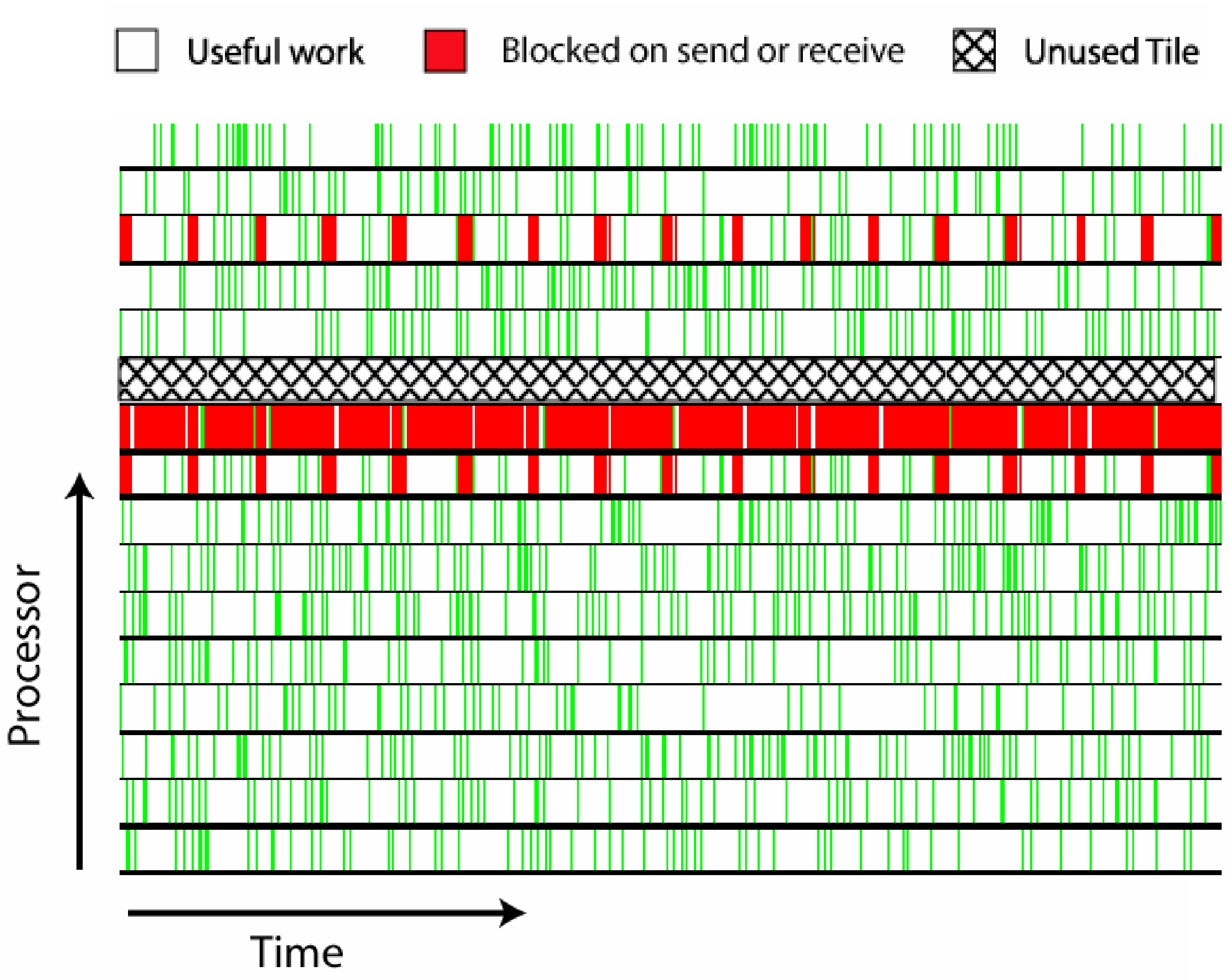
# Example: Radar App.



# Example: Radar App. (Balanced)

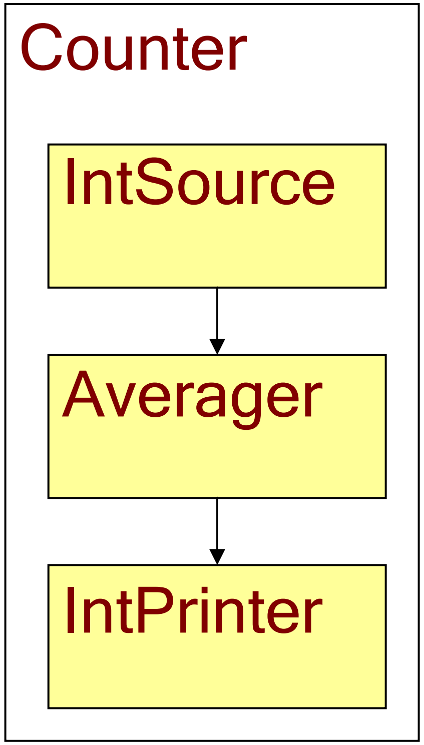


# Example: Radar App. (Balanced)



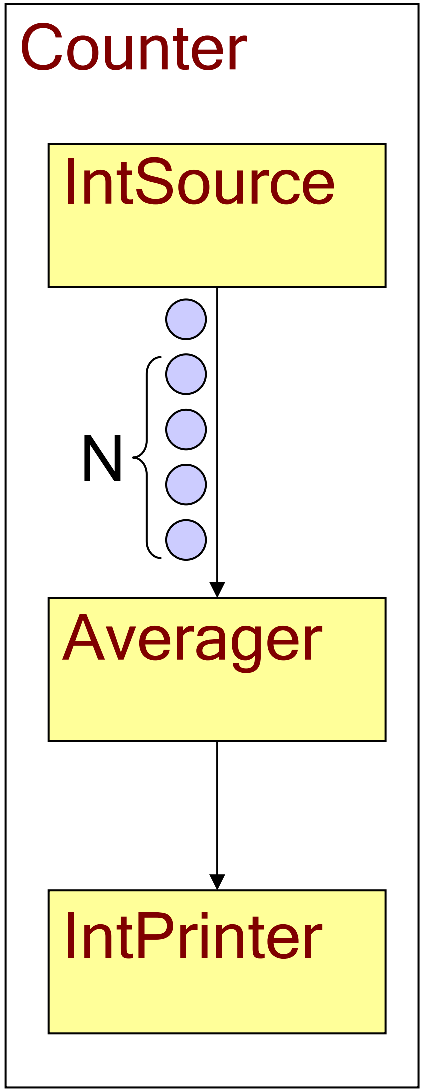
# A Moving Average

```
void->void pipeline MovingAverage() {
    add IntSource();
    add Averager(10);
    add IntPrinter();
}
int->int filter Averager(int N) {
    work pop 1 push 1 peek N-1 {
      int sum = 0;
      for (int i=0; i<N; i++) {
        sum += peek(i);
      }
    push(sum/N);
    pop();
}
```



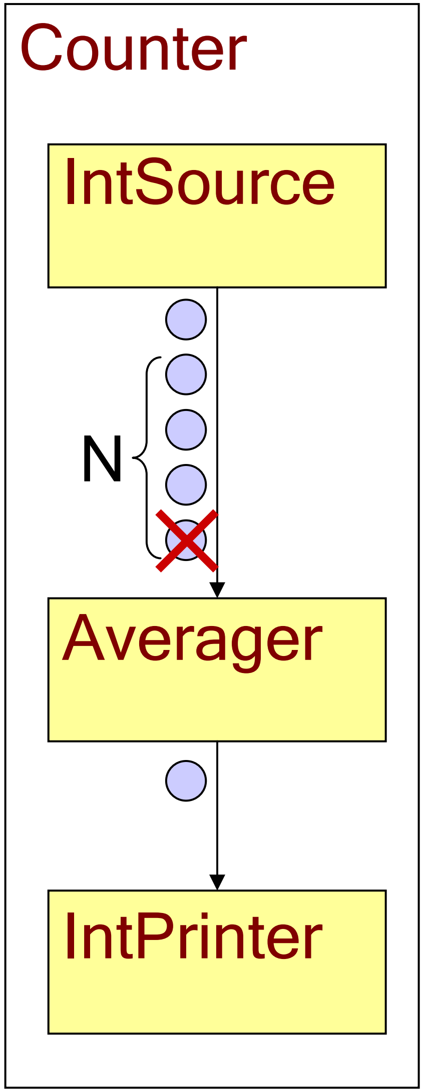
# A Moving Average

```
void->void pipeline MovingAverage() {
    add IntSource();
    add Averager(4);
    add IntPrinter();
}
int->int filter Averager(int N) {
    work pop 1 push 1 peek N-1 {
      int sum = 0;
      for (int i=0; i<N; i++) {
        sum += peek(i);
      }
    push(sum/N);
    pop();
}
```



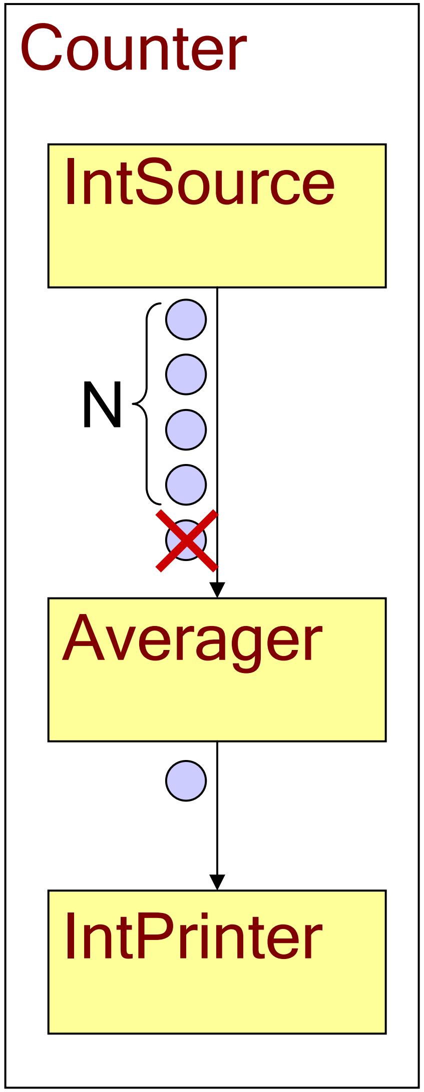
# A Moving Average

```
void->void pipeline MovingAverage() {
    add IntSource();
    add Averager(4);
    add IntPrinter();
}
int->int filter Averager(int N) {
    work pop 1 push 1 peek N-1 {
      int sum = 0;
      for (int i=0; i<N; i++) {
        sum += peek(i);
      }
    push(sum/N);
    pop();
}
```



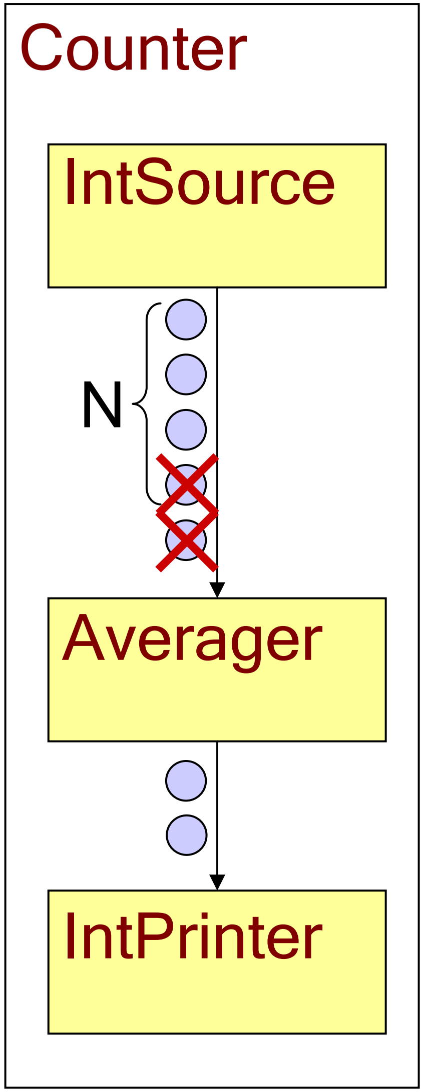
# A Moving Average

```
void->void pipeline MovingAverage() {
    add IntSource();
    add Averager(4);
    add IntPrinter();
}
int->int filter Averager(int N) {
    work pop 1 push 1 peek N-1 {
      int sum = 0;
      for (int i=0; i<N; i++) {
        sum += peek(i);
      }
    push(sum/N);
    pop();
}
```



# A Moving Average

```
void->void pipeline MovingAverage() {
    add IntSource();
    add Averager(4);
    add IntPrinter();
}
int->int filter Averager(int N) {
    work pop 1 push 1 peek N-1 {
      int sum = 0;
      for (int i=0; i<N; i++) {
        sum += peek(i);
      }
    push(sum/N);
    pop();
}
```


Counter
IntSource
N
Averager
IntPrinter