# Language and Compiler Support for Stream Programs

Bill Thies

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

Thesis Defense

September 11, 2008

**Date:** Wed, 17 Nov 1999
**From:** Saman Amarasinghe <saman@lcs.mit.edu>
**To:** Bill Thies <thies@mit.edu>
**Subject:** UROP Opportunities

Hi Bill,

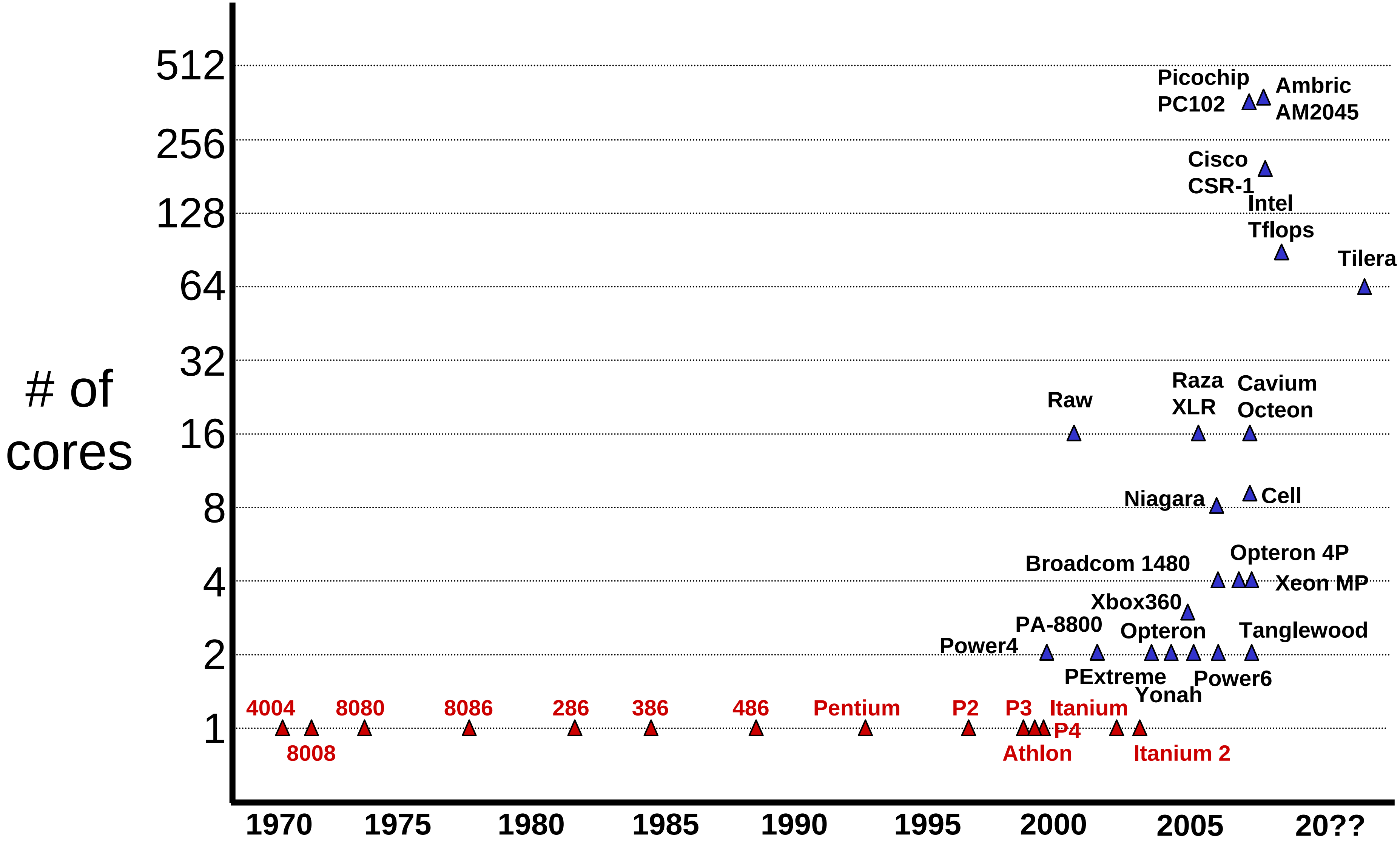
I have a few UROP opportunities in the RAW project ...

Most of the projects can lead to an MENG thesis and beyond ...

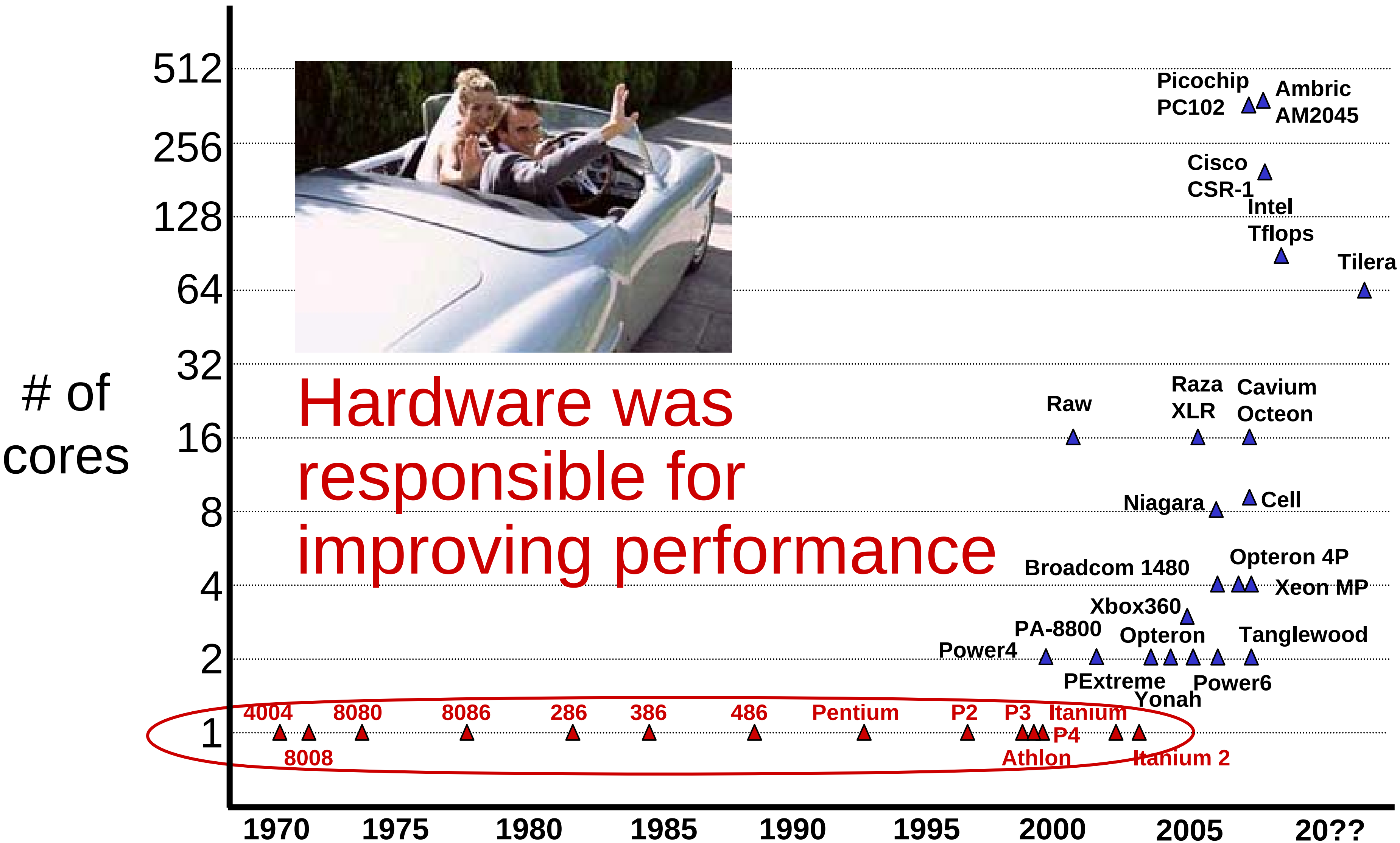
# Acknowledgments

- **Project supervisors**
  - Prof. Saman Amarasinghe
  - Dr. Rodric Rabbah
- **Contributors to this talk**
  - Michael I. Gordon (Ph.D. student) – *led development of Raw backend*
  - Andrew A. Lamb (M.Eng) – *led development of linear optimizations*
  - Sitij Agrawal (M.Eng) – *led development of statespace optimizations*
- **Compiler developers**
  - Kunal Agrawal
  - Allyn Dimock
  - Steve Hall
  - Qiuyuan Jimmy Li
  - Jasper Lin
  - Michal Karczmarek
  - David Maze
  - Janis Sermulins
  - Phil Sung
  - Ceryen Tan
  - David Zhang
- **Application developers**
  - Basier Aziz
  - Matthew Brown
  - Jiawen Chen
  - Matthew Drake
  - Shirley Fung
  - Hank Hoffmann
  - Chris Leger
  - Ali Meli
  - Mani Narayanan
  - Satish Ramaswamy
  - Jeremy Wong
- **User interface developers**
  - Kimberly Kuo
  - Juan Reyes

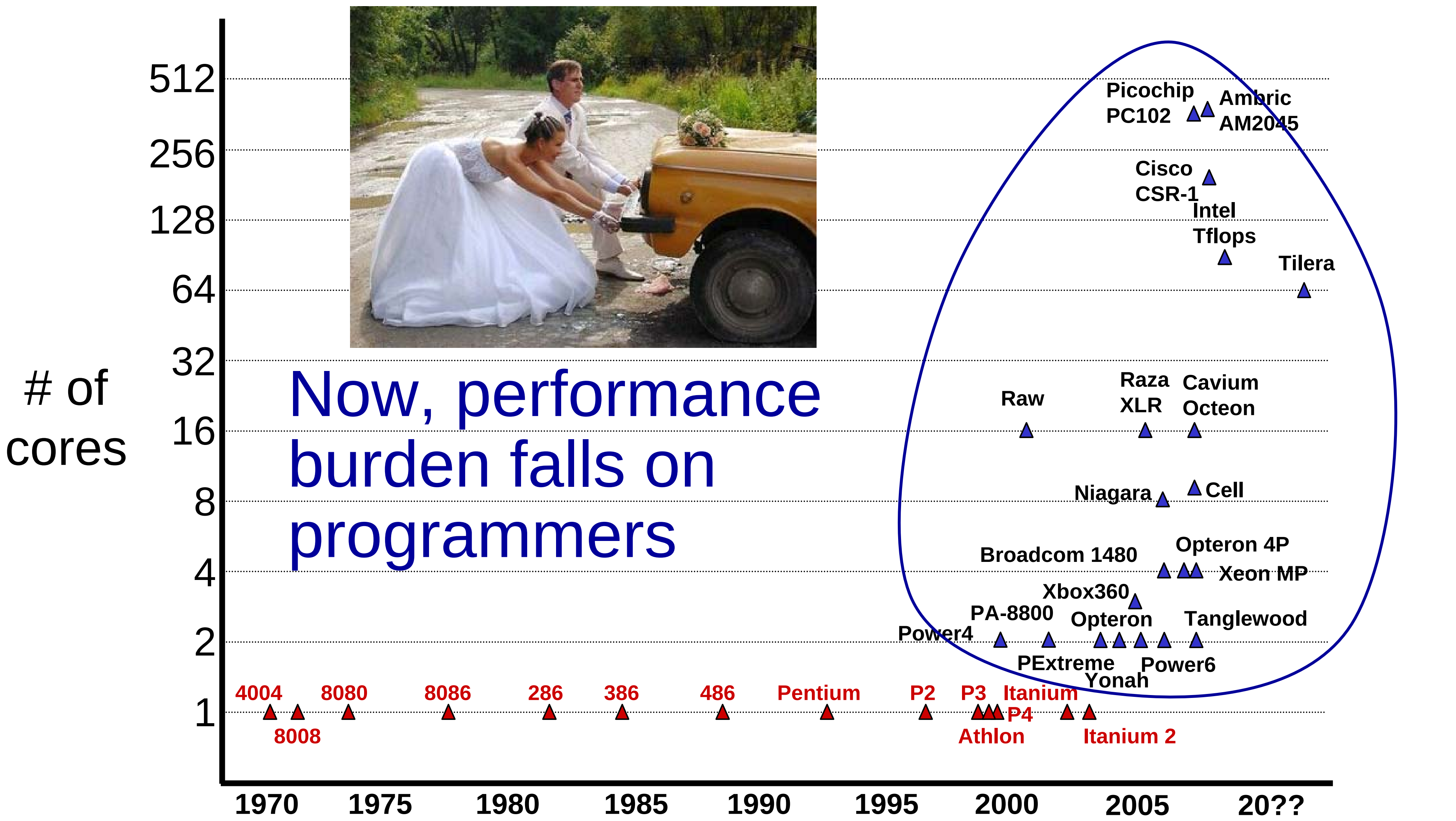
# Multicores are Here



# Multicores are Here



# Multicores are Here

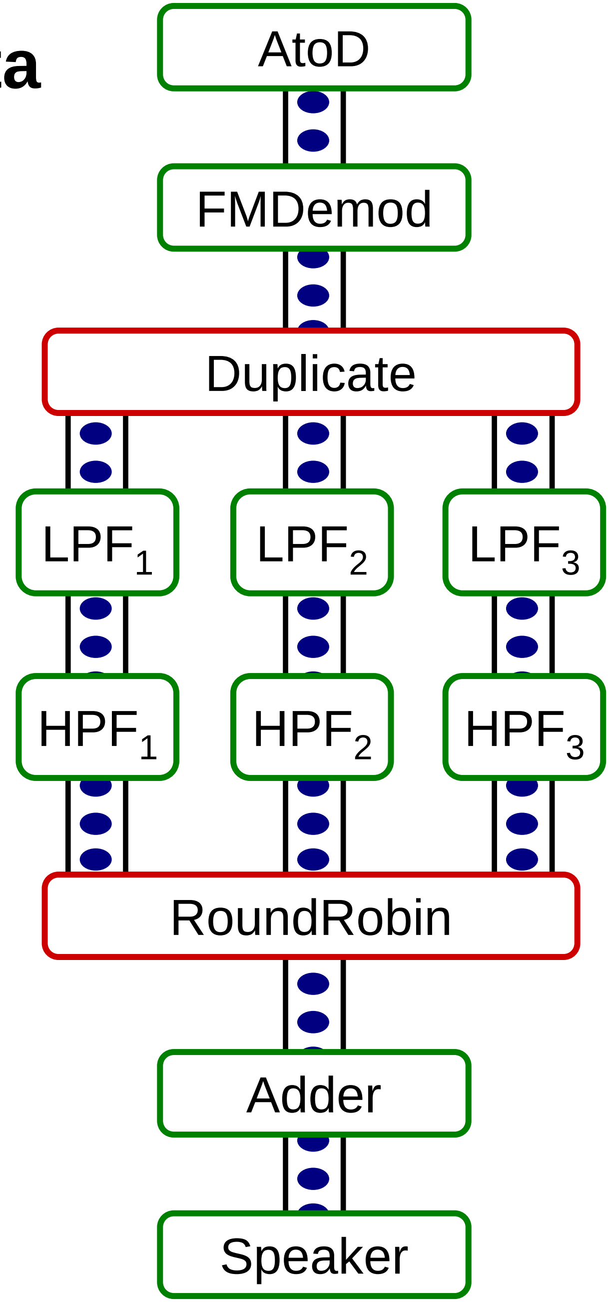


# Is Parallel Programming a New Problem?

- **No!  Decades of research targeting multiprocessors**
  – Languages, compilers, architectures, tools…
- **What is different today?**
  1. **Multicores vs. multiprocessors.** Multicores have:
     - New interconnects with non-uniform communication costs
     - Faster on-chip communication than off-chip I/O, memory ops
     - Limited per-core memory availability
  2. **Non-expert programmers**
     - Supercomputers with >2048 processors today: 100 [top500.org]
     - Machines with >2048 cores in 2020: >100 million [ITU, Moore]
  3. **Application trends**
     - Embedded: 2.7 billion cell phones vs 850 million PCs [ITU 2006]
     - Data-centric: YouTube streams 200 TB of video daily

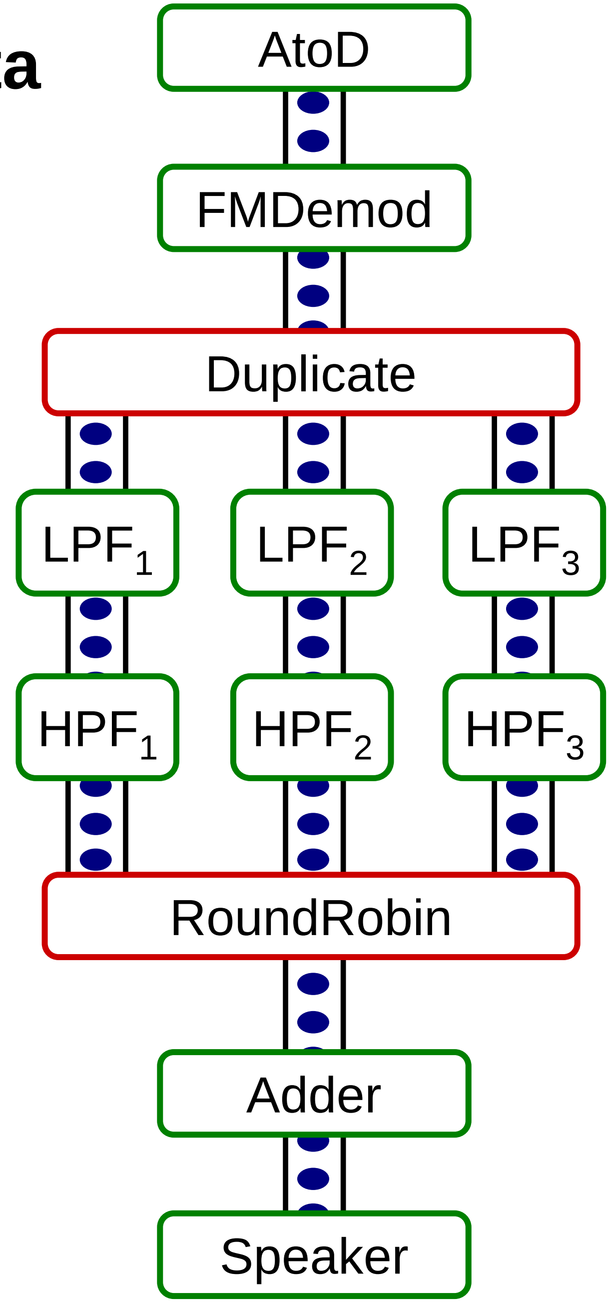
# Streaming Application Domain

- **For programs based on streams of data**
  - Audio, video, DSP, networking, and cryptographic processing kernels
  - Examples: HDTV editing, radar tracking, microphone arrays, cell phone base stations, graphics

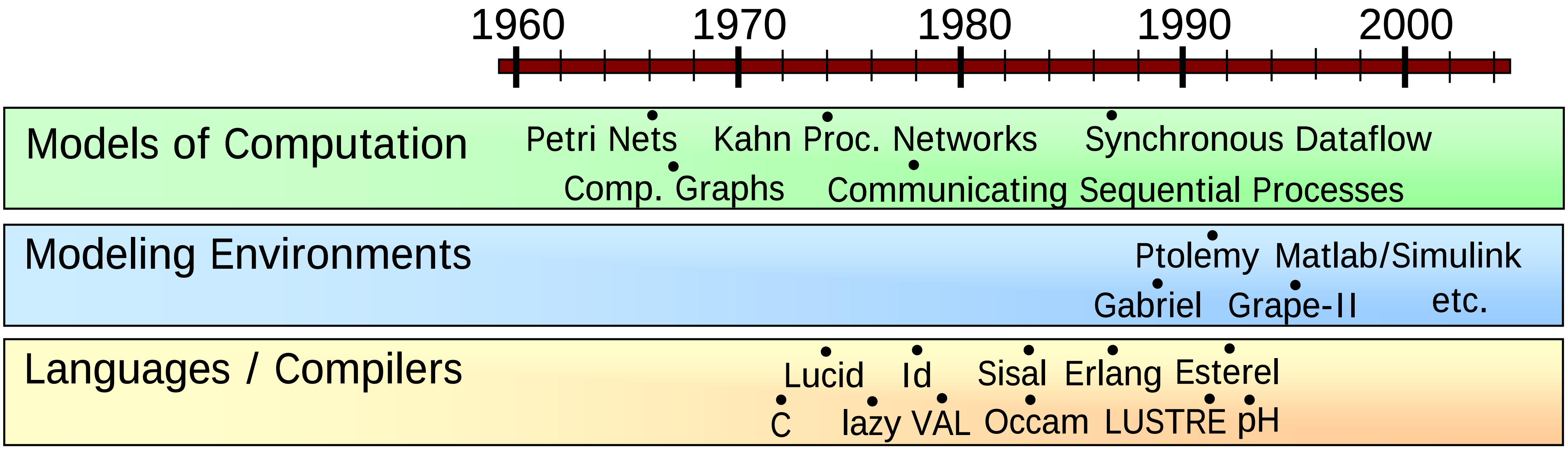


# Streaming Application Domain

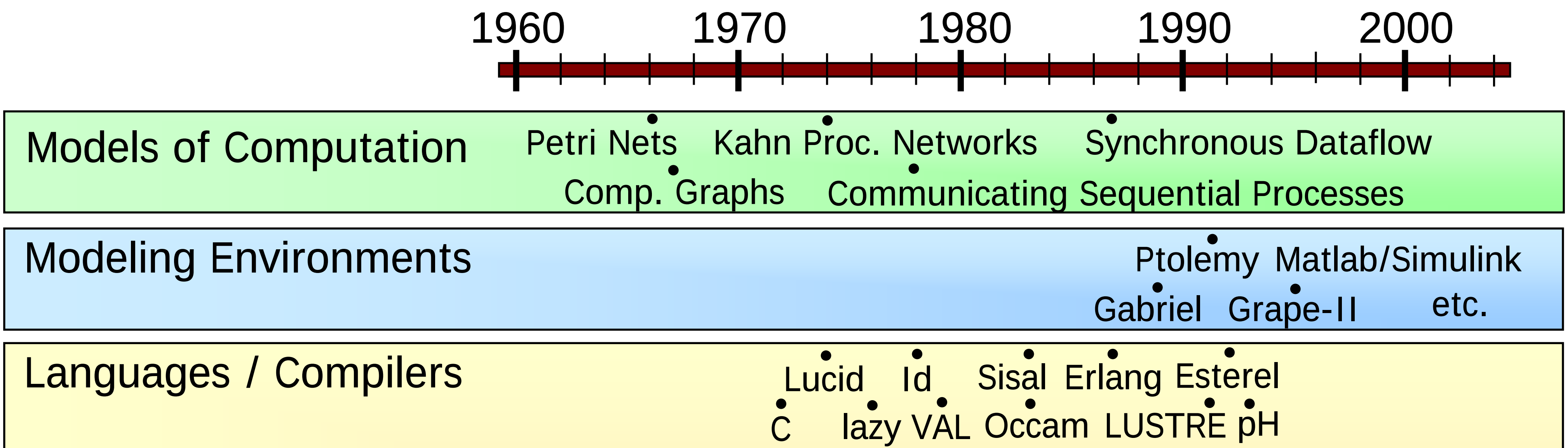
- **For programs based on streams of data**
  - Audio, video, DSP, networking, and cryptographic processing kernels
  - Examples: HDTV editing, radar tracking, microphone arrays, cell phone base stations, graphics
- **Properties of stream programs**
  - Regular and repeating computation
  - Independent filters with explicit communication
  - Data items have short lifetimes



# Brief History of Streaming



# Brief History of Streaming




**Strengths**
- Elegance
- Generality

**Weaknesses**
- Unsuitable for static analysis
- Cannot leverage deep results from DSP / modeling community

# Brief History of Streaming

1960 1970 1980 1990 2000

**Models of Computation**
Petri Nets  Kahn Proc. Networks  Synchronous Dataflow
Comp. Graphs  Communicating Sequential Processes

**Modeling Environments**
Ptolemy  Matlab/Simulink
Gabriel  Grape-II  etc.

**Languages / Compilers**
Lucid  Id  Sisal  Erlang  Esterel
C  lazy  VAL  Occam  LUSTRE  pH

StreamIt  Brook
Cg  StreamC

***"Stream Programming"***

| **Strengths** | **Weaknesses** |
|---|---|
| • Elegance | • Unsuitable for static analysis |
| • Generality | • Cannot leverage deep results from DSP / modeling community |

# StreamIt: A Language and Compiler for Stream Programs

- **Key idea: design language that enables static analysis**
- **Goals:**
  1. Expose and exploit the parallelism in stream programs
  2. Improve programmer productivity in the streaming domain
- **Project contributions:**
  - Language design for streaming [CC'02, CAN'02, PPoPP'05, IJPP'05]
  - Automatic parallelization [ASPLOS'02, G.Hardware'05, ASPLOS'06]
  - Domain-specific optimizations [PLDI'03, CASES'05, TechRep'07]
  - Cache-aware scheduling [LCTES'03, LCTES'05]
  - Extracting streams from legacy code [MICRO'07]
  - User + application studies [PLDI'05, P-PHEC'05, IPDPS'06]
  - **7 years, 25 people, 300 KLOC**
  - **700 external downloads, 5 external publications**

# StreamIt: A Language and Compiler for Stream Programs

- **Key idea: design language that enables static analysis**
- **Goals:**
  1. Expose and exploit the parallelism in stream programs
  2. Improve programmer productivity in the streaming domain
- **I contributed to:**
  - Language design for streaming [CC'02, CAN'02, PPoPP'05, IJPP'05]
  - Automatic parallelization [ASPLOS'02, G.Hardware'05, ASPLOS'06]
  - Domain-specific optimizations [PLDI'03, CASES'05, TechRep'07]
  - Cache-aware scheduling [LCTES'03, LCTES'05]
  - Extracting streams from legacy code [MICRO'07]
  - User + application studies [PLDI'05, P-PHEC'05, IPDPS'06]
  - **7 years, 25 people, 300 KLOC**
  - **700 external downloads, 5 external publications**

# StreamIt: A Language and Compiler for Stream Programs

- **Key idea: design language that enables static analysis**
- **Goals:**
  1. Expose and exploit the parallelism in stream programs
  2. Improve programmer productivity in the streaming domain
- **This talk:**
  - Language design for streaming [CC'02, CAN'02, PPoPP'05, IJPP'05]
  - Automatic parallelization [ASPLOS'02, G.Hardware'05, ASPLOS'06]
  - Domain-specific optimizations [PLDI'03, CASES'05, TechRep'07]
  - Cache-aware scheduling [LCTES'03, LCTES'05]
  - Extracting streams from legacy code [MICRO'07]
  - User + application studies [PLDI'05, P-PHEC'05, IPDPS'06]
  - **7 years, 25 people, 300 KLOC**
  - **700 external downloads, 5 external publications**

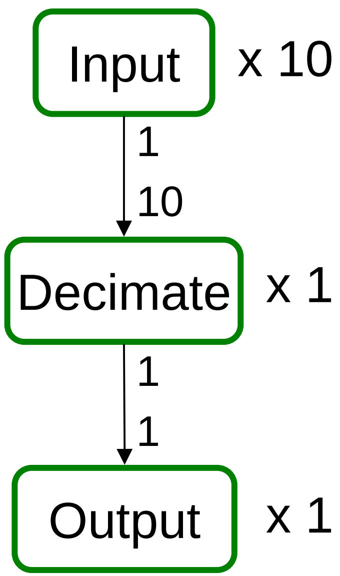
# Part 1:  Language Design

*William Thies, Michal Karczmarek, Saman Amarasinghe (CC'02)*

*William Thies, Michal Karczmarek, Janis Sermulins, Rodric Rabbah, Saman Amarasinghe (PPoPP'05)*

# StreamIt Language Basics

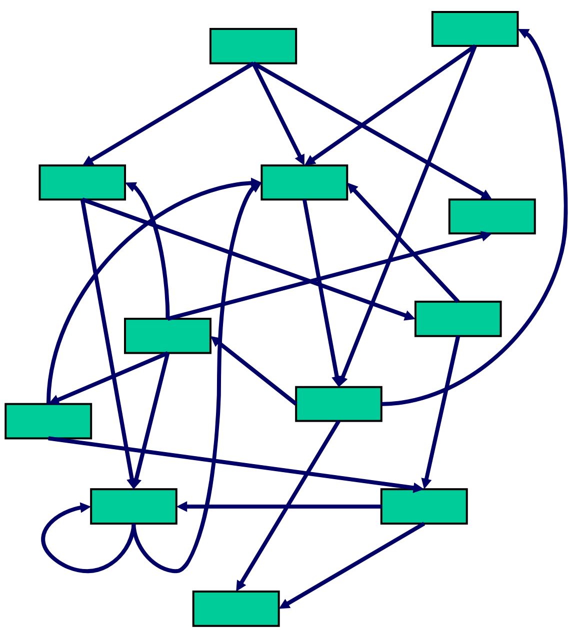
- **High-level, architecture-independent language**
  – Backend support for uniprocessors, multicores (Raw, SMP), cluster of workstations
- **Model of computation: synchronous dataflow** [Lee & Messerschmidt, 1987]
  – Program is a graph of independent *filters*
  – Filters have an atomic execution step with known input / output rates
  – Compiler is responsible for scheduling and buffer management



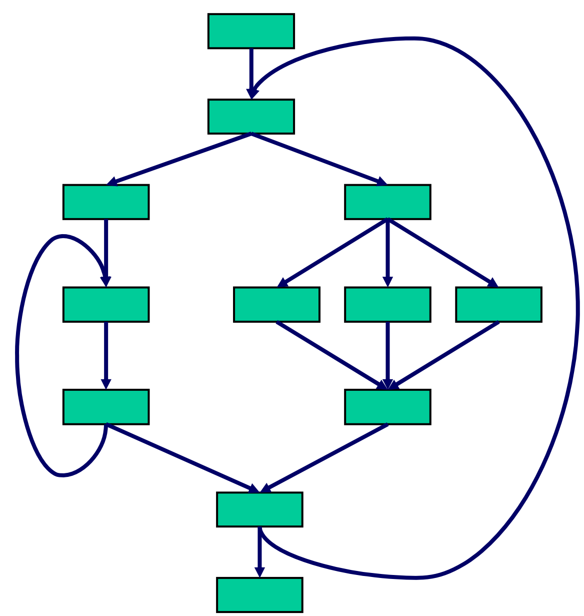

- **Extensions to synchronous dataflow**
  – Dynamic I/O rates
  – Support for sliding window operations
  – Teleport messaging [PPoPP'05]

# Representing Streams

- **Conventional wisdom: stream programs are graphs**
  - Graphs have no simple textual representation
  - Graphs are difficult to analyze and optimize
- **Insight: stream programs have structure**

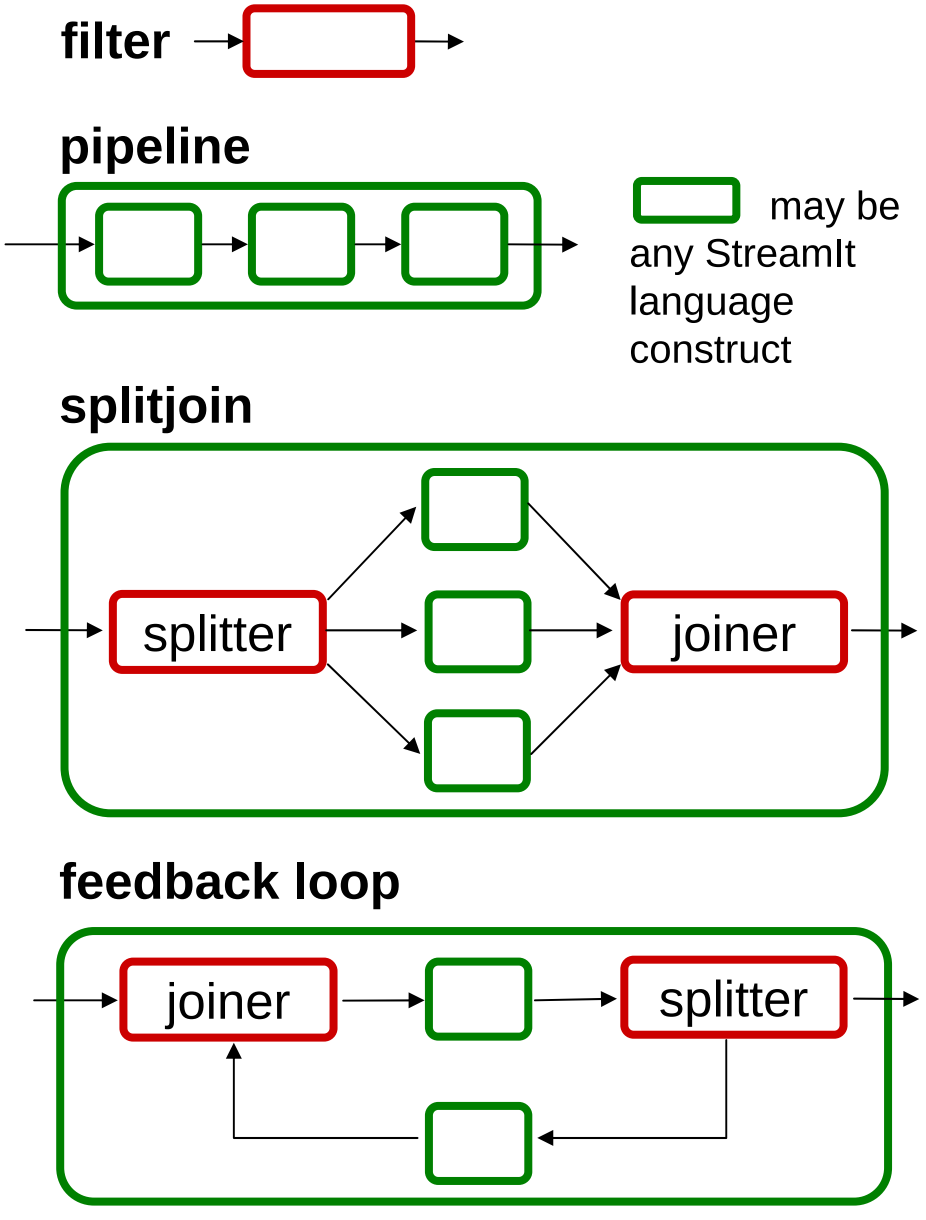


*unstructured*



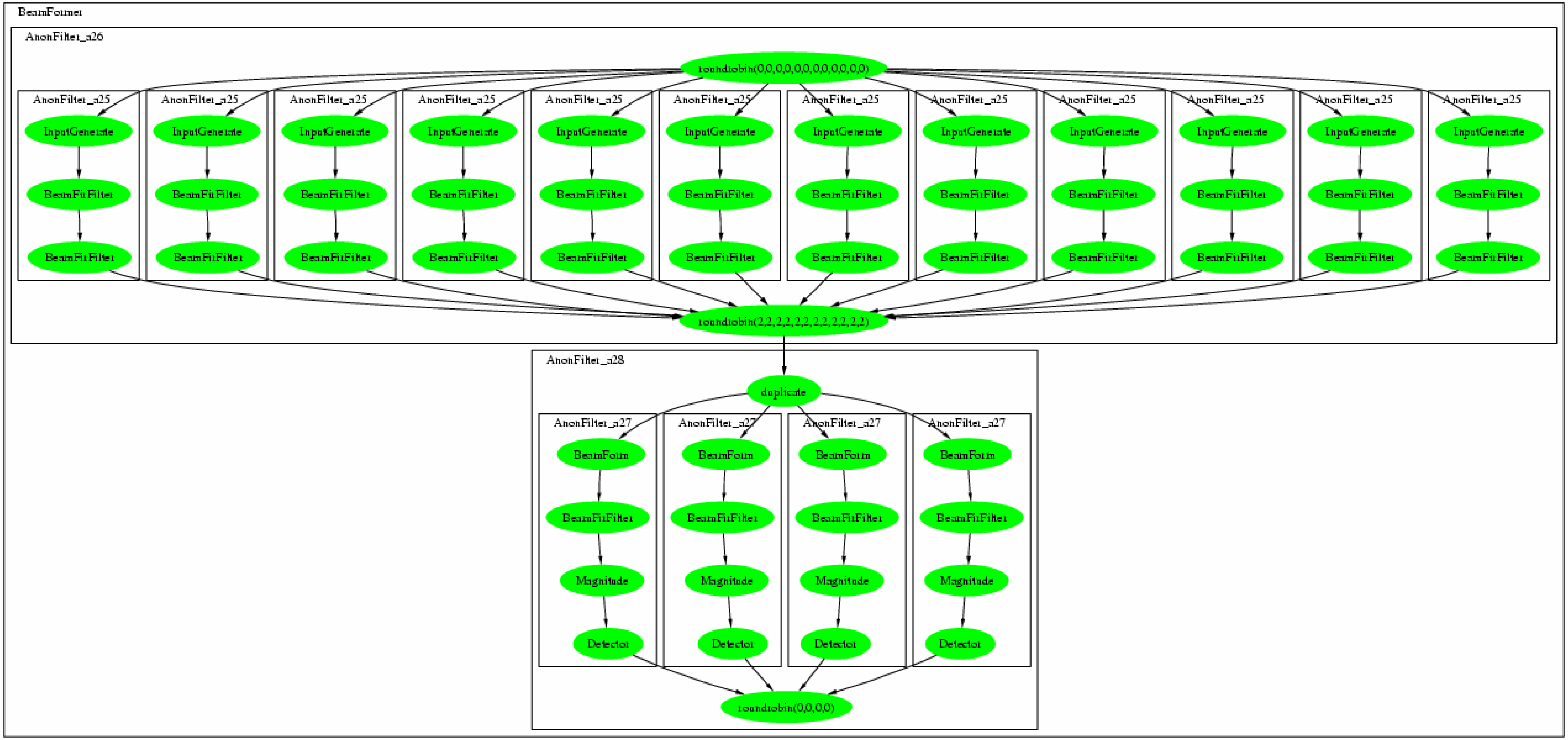
*structured*

# Structured Streams

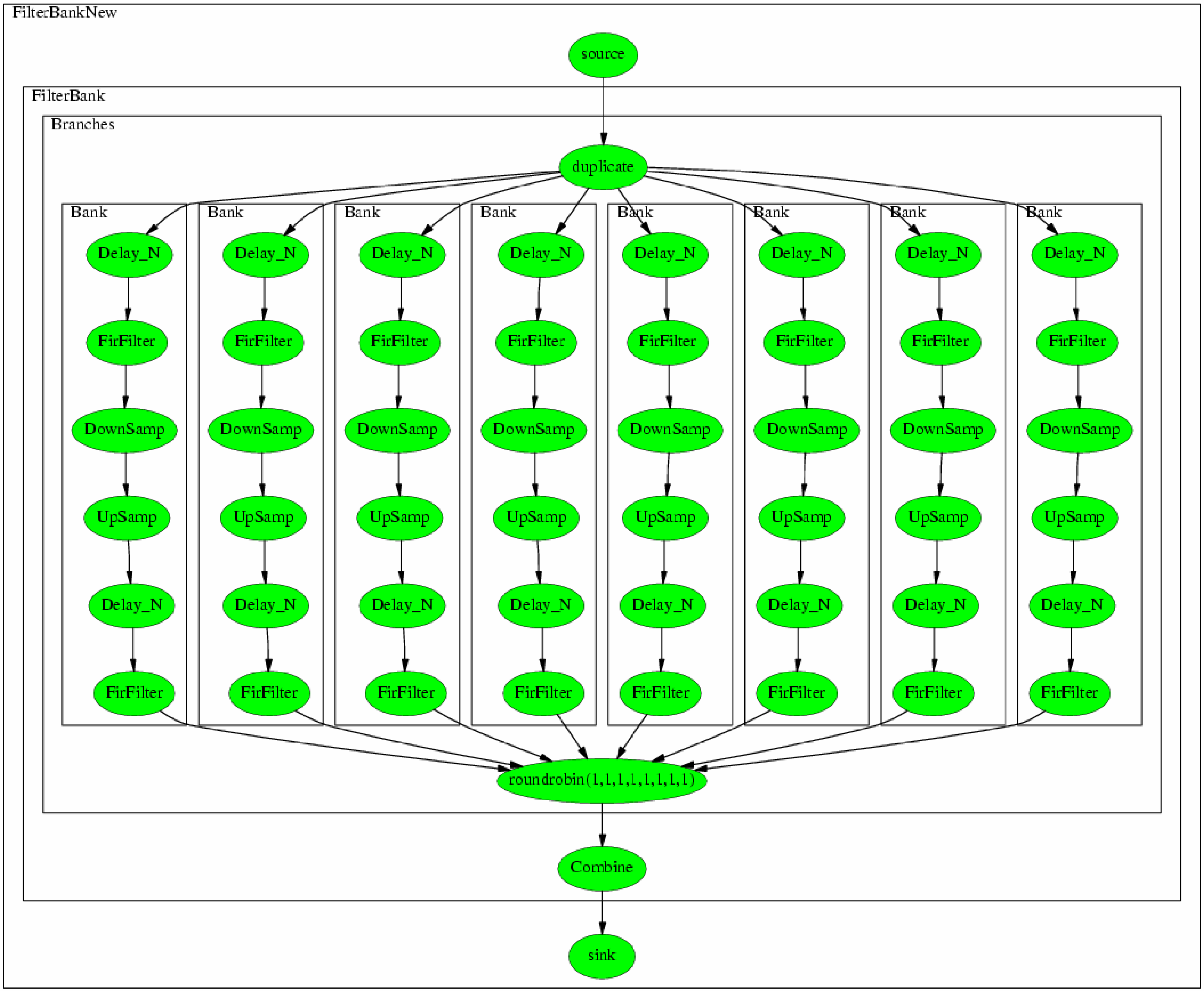



- **Each structure is single-input, single-output**
- **Hierarchical and composable**

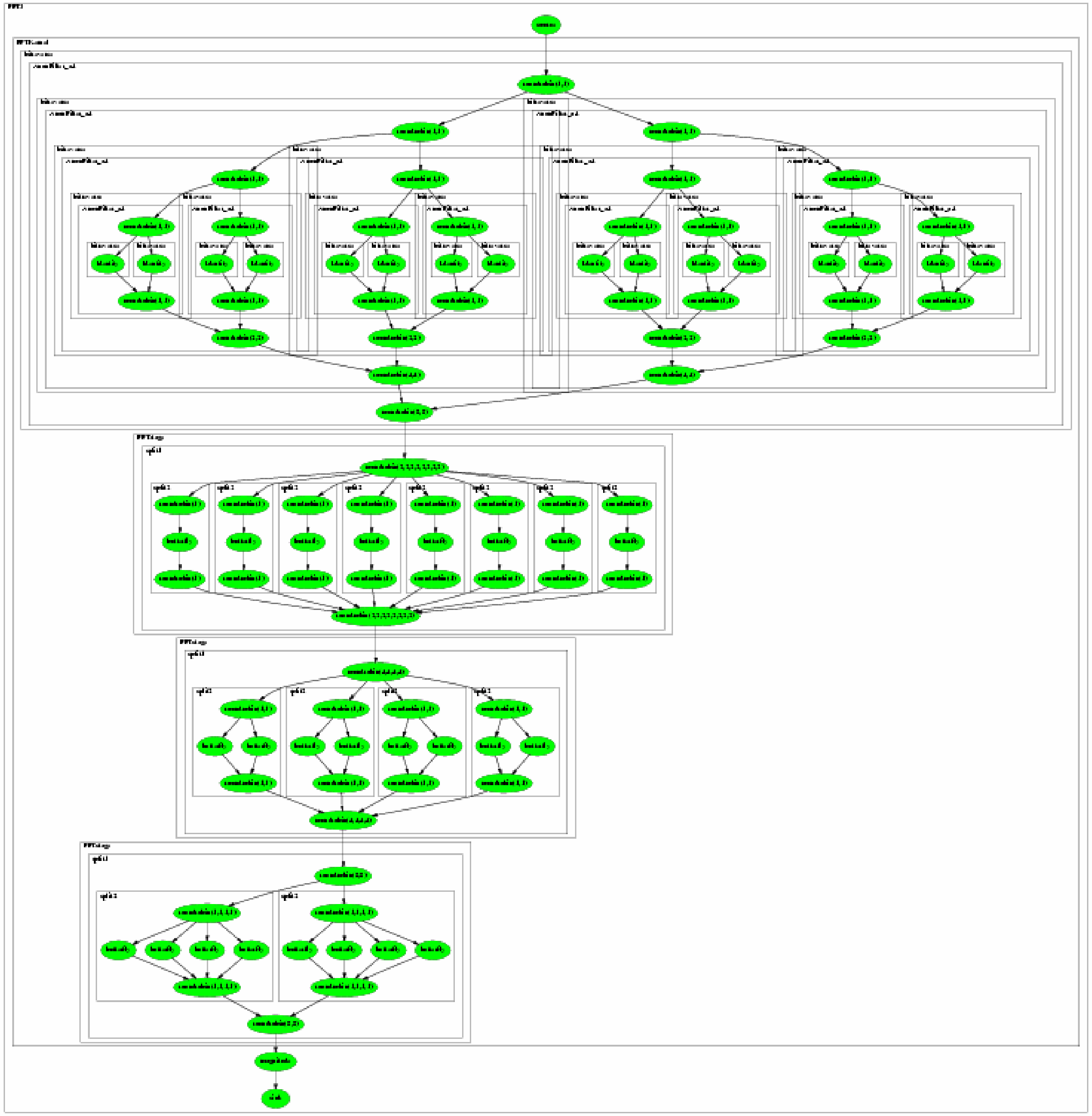
# Radar-Array Front End



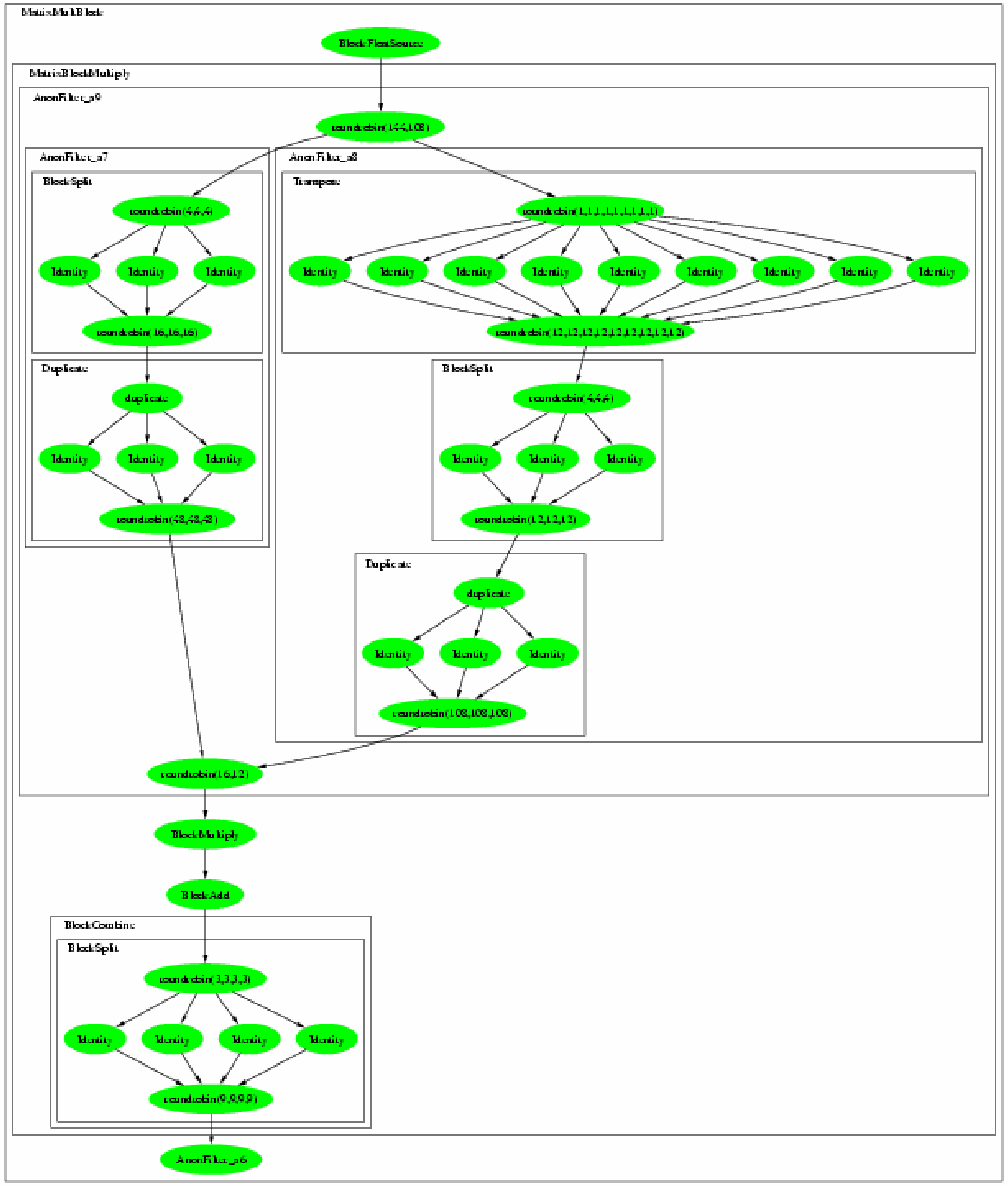
# Filterbank


FilterBankNew
source
FilterBank
Branches
duplicate
Bank
Delay_N
FirFilter
DownSamp
UpSamp
Delay_N
FirFilter
roundrobin(1,1,1,1,1,1,1,1)
Combine
sink

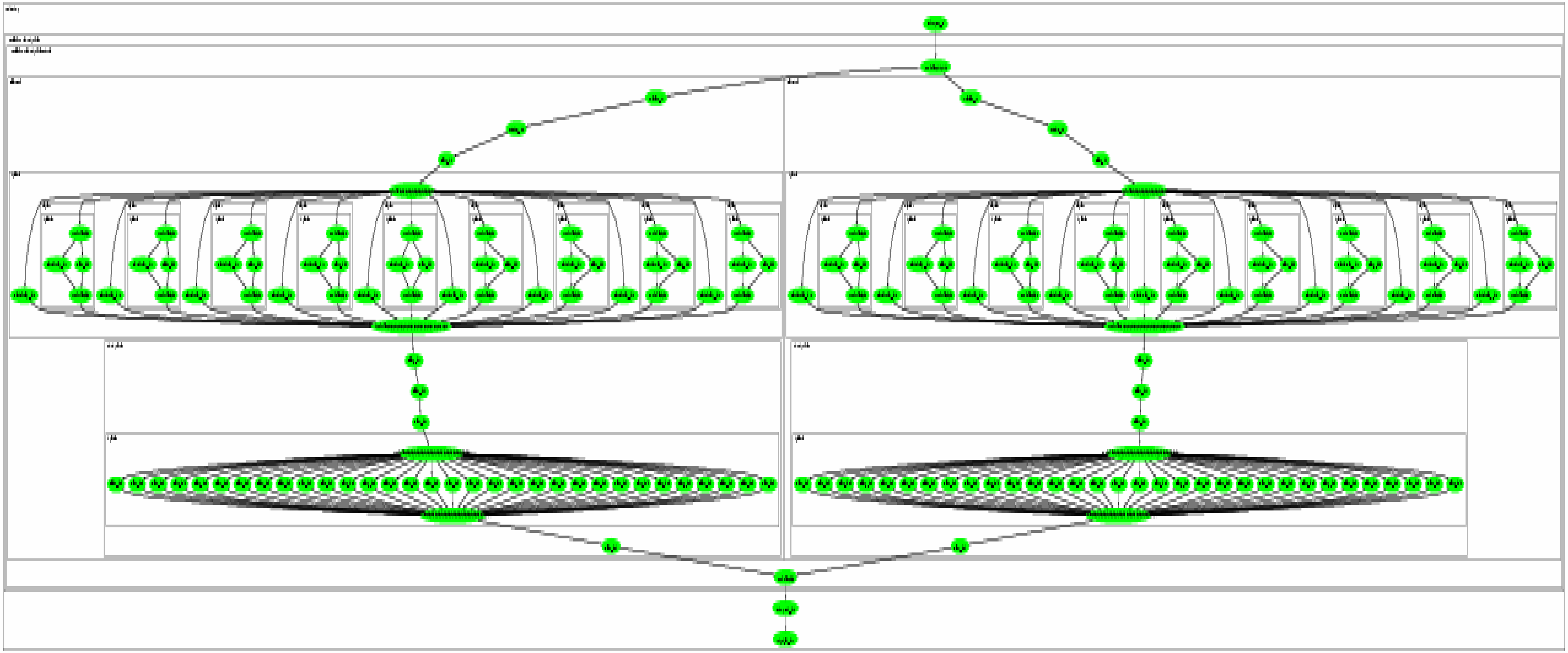
# FFT



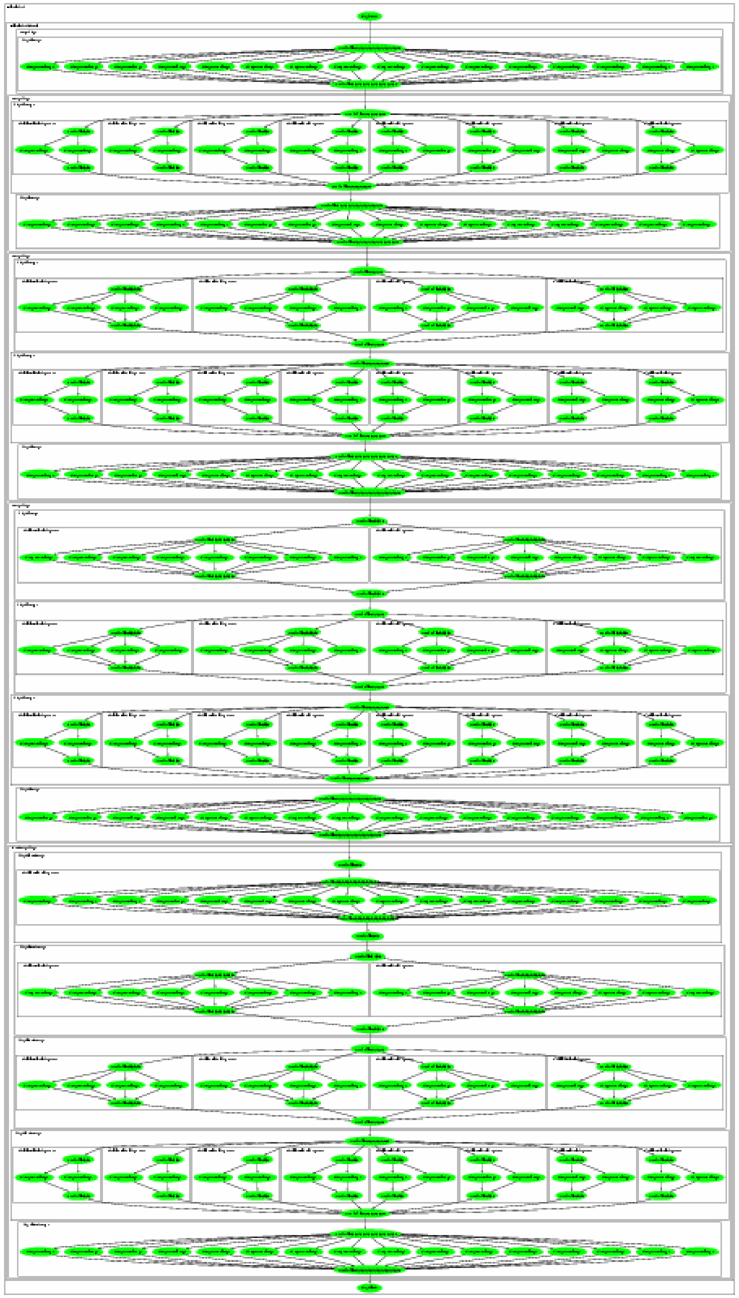
# Block Matrix Multiply



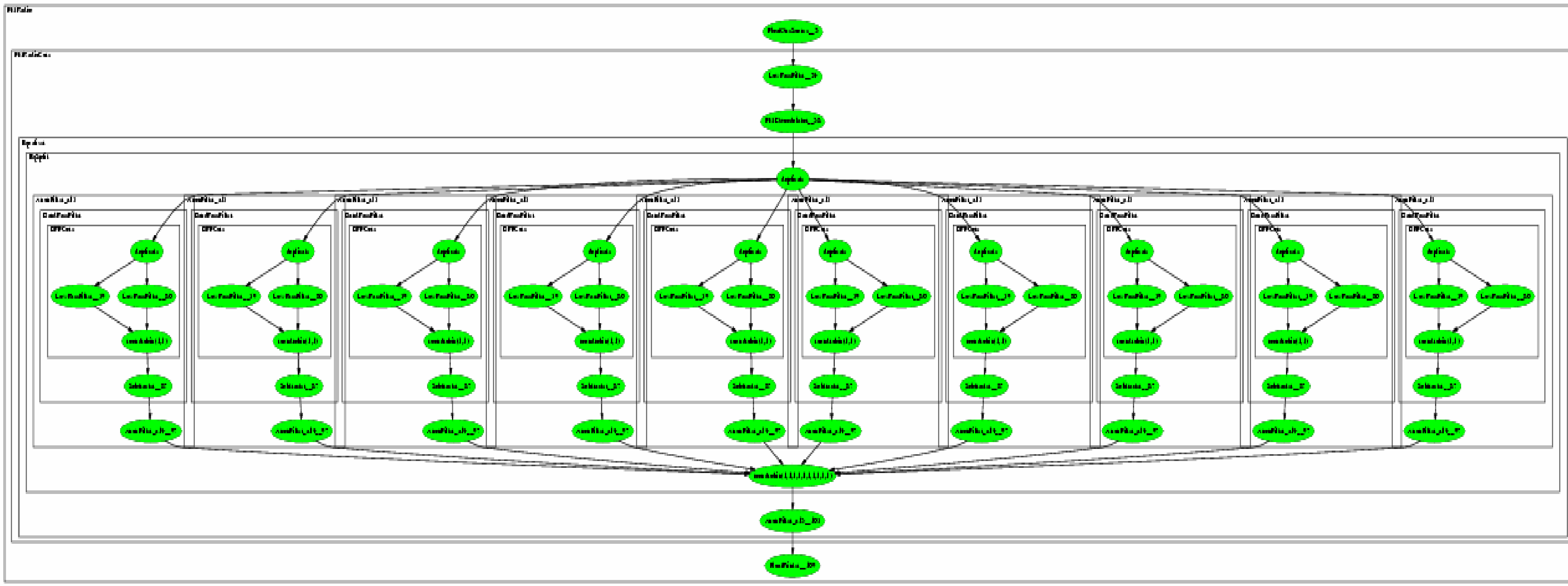
# MP3 Decoder



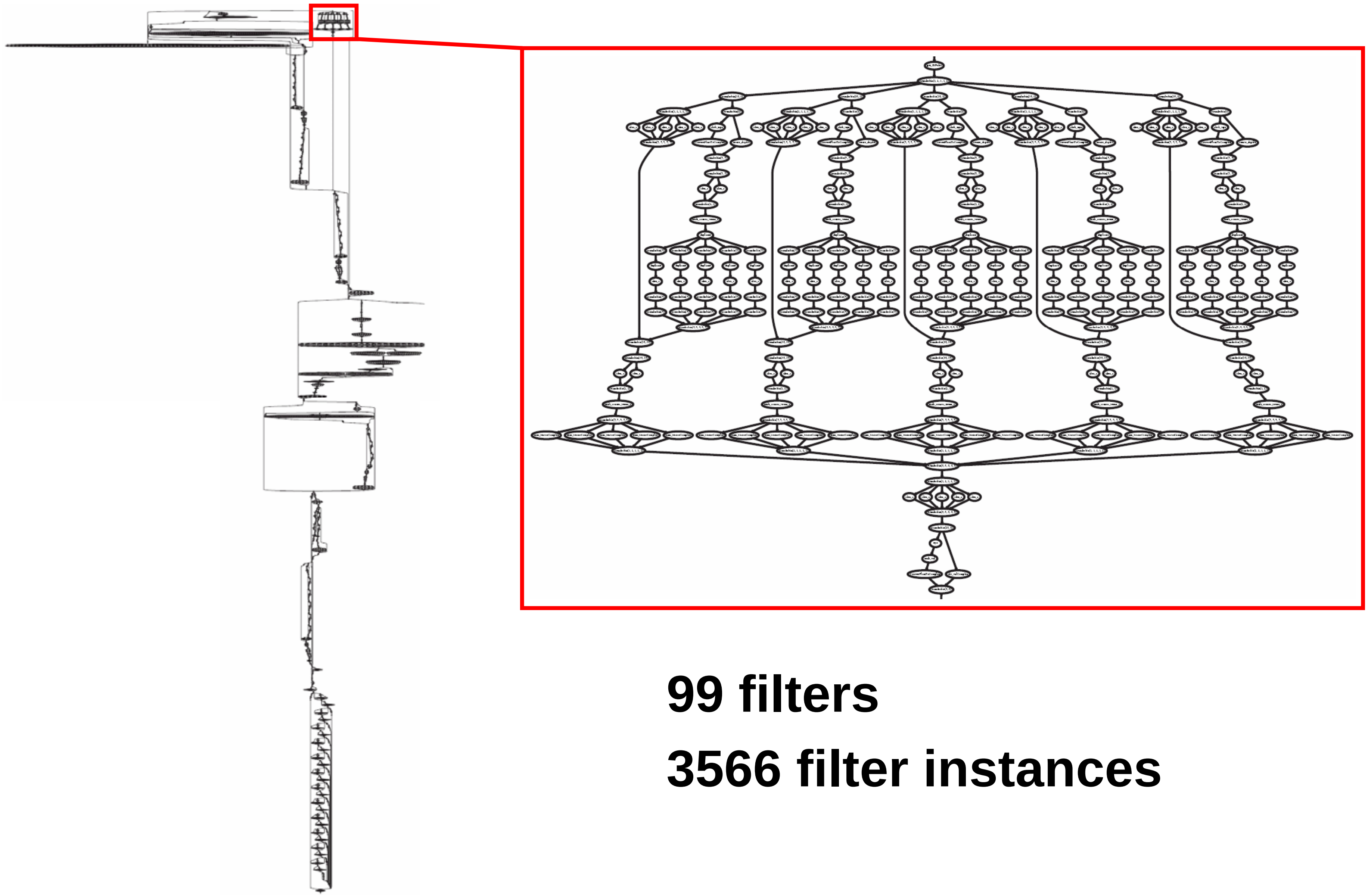
# Bitonic Sort



# FM Radio with Equalizer



# Ground Moving Target Indicator (GMTI)

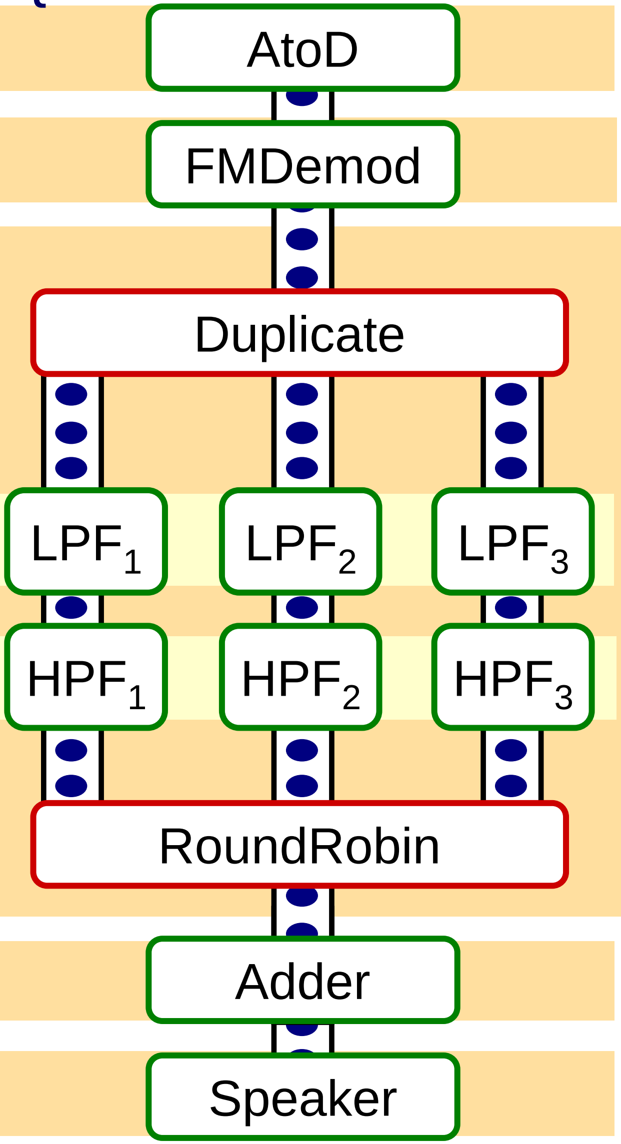


**99 filters**

**3566 filter instances**

# Example Syntax:  FMRadio

```
void->void pipeline FMRadio(int N, float lo, float hi) {
  add AtoD();
  add FMDemod();
  add splitjoin {
   split duplicate;
   for (int i=0; i<N; i++) {
     add pipeline {
       add LowPassFilter(lo + i*(hi - lo)/N);
       add HighPassFilter(lo + i*(hi - lo)/N);
     }
   }
   join roundrobin();
  }
  add Adder();
  add Speaker();
}
```



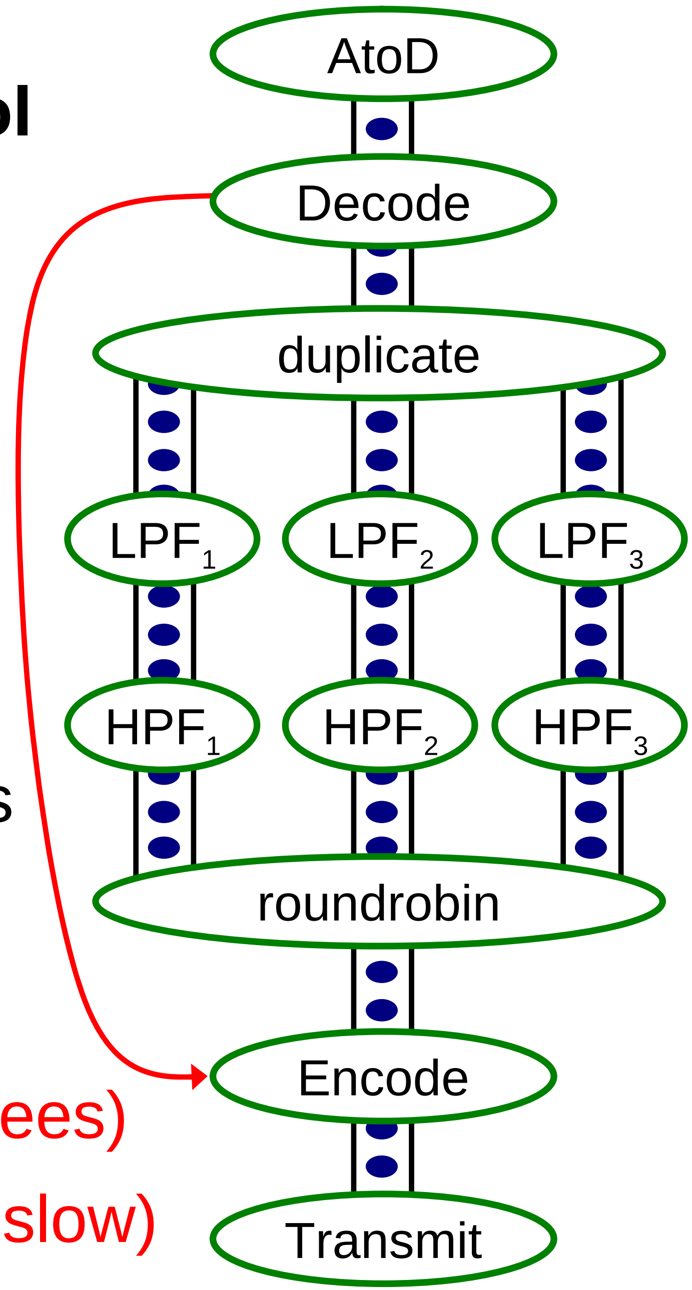
# StreamIt Application Suite

- Software radio
- Frequency hopping radio
- Acoustic beam former
- Vocoder
- FFTs and DCTs
- JPEG Encoder/Decoder
- MPEG-2 Encoder/Decoder
- MPEG-4 (fragments)
- Sorting algorithms
- GMTI (Ground Moving Target Indicator)
- DES and Serpent crypto algorithms
- SSCA#3 (HPCS scalable benchmark for synthetic aperture radar)
- Mosaic imaging using RANSAC algorithm

Total size: 60,000 lines of code

# Control Messages

- **Occasionally, low-bandwidth control messages are sent between actors**
- **Often demands precise timing**
  - Communications: adjust protocol, amplification, compression
  - Network router: cancel invalid packet
  - Adaptive beamformer: track a target
  - Respond to user input, runtime errors
  - Frequency hopping radio
- **Traditional techniques:**
  - Direct method call (no timing guarantees)
  - Embed message in stream (opaque, slow)

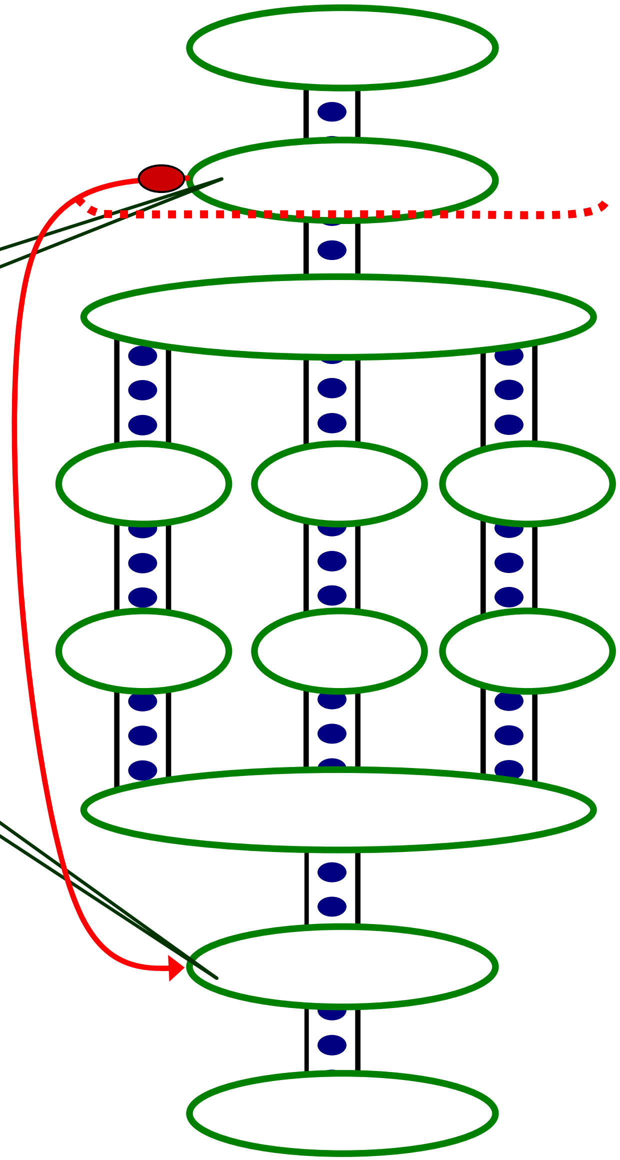


# Idea 2: Teleport Messaging

- **Looks like method call, but timed relative to data in the stream**

```
TargetFilter x;
if newProtocol(p) {
  x.setProtocol(p) @ 2;
}
```

```
void setProtocol(int p) {
  reconfig(p);
}
```



  – Exposes dependences to compiler
  – Simple and precise for user
    - Adjustable latency
    - Can send upstream or downstream

# Part 2:  Automatic Parallelization

*Michael I. Gordon, William Thies, Saman Amarasinghe (ASPLOS'06)*

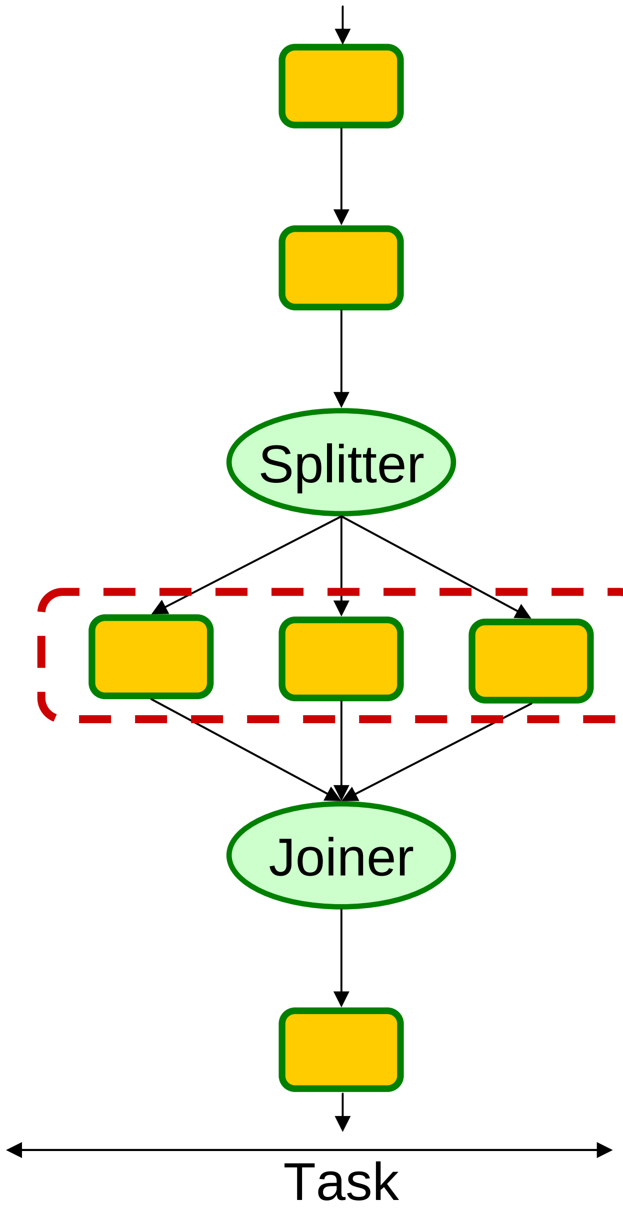
*Michael I. Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Andrew A. Lamb, Chris Leger, Jeremy Wong, Henry Hoffmann, David Maze, Saman Amarasinghe (ASPLOS'02)*

# Streaming is an Implicitly Parallel Model

- **Programmer thinks about functionality, not parallelism**
- **More explicit models may…**
  - Require knowledge of target [MPI] [cG]
  - Require parallelism annotations [OpenMP] [HPF] [Cilk] [Intel TBB]
- **Novelty over other implicit models?**
  [Erlang] [MapReduce] [Sequoia] [pH] [Occam] [Sisal] [Id] [VAL] [LUSTRE] [HAL] [THAL] [SALSA] [Rosette] [ABCL] [APL] [ZPL] [NESL] […]

  → **Exploiting streaming structure for robust performance**

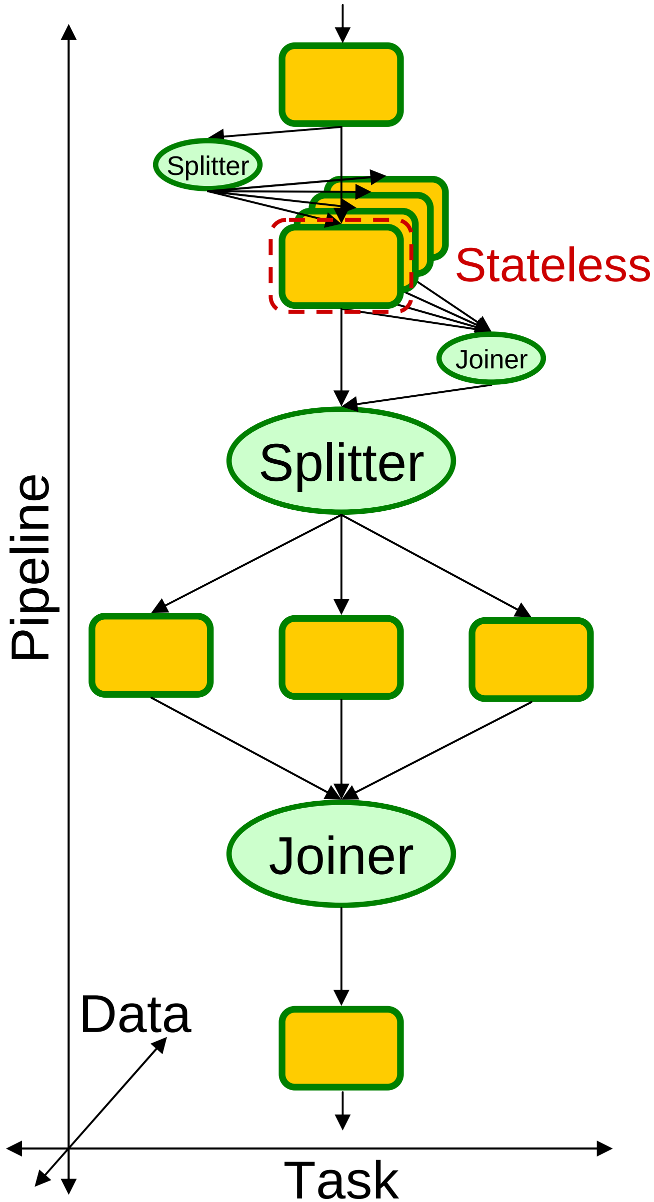
# Parallelism in Stream Programs




**Task parallelism**

– Analogous to thread (fork/join) parallelism

# Parallelism in Stream Programs




**Task parallelism**

– Analogous to thread (fork/join) parallelism

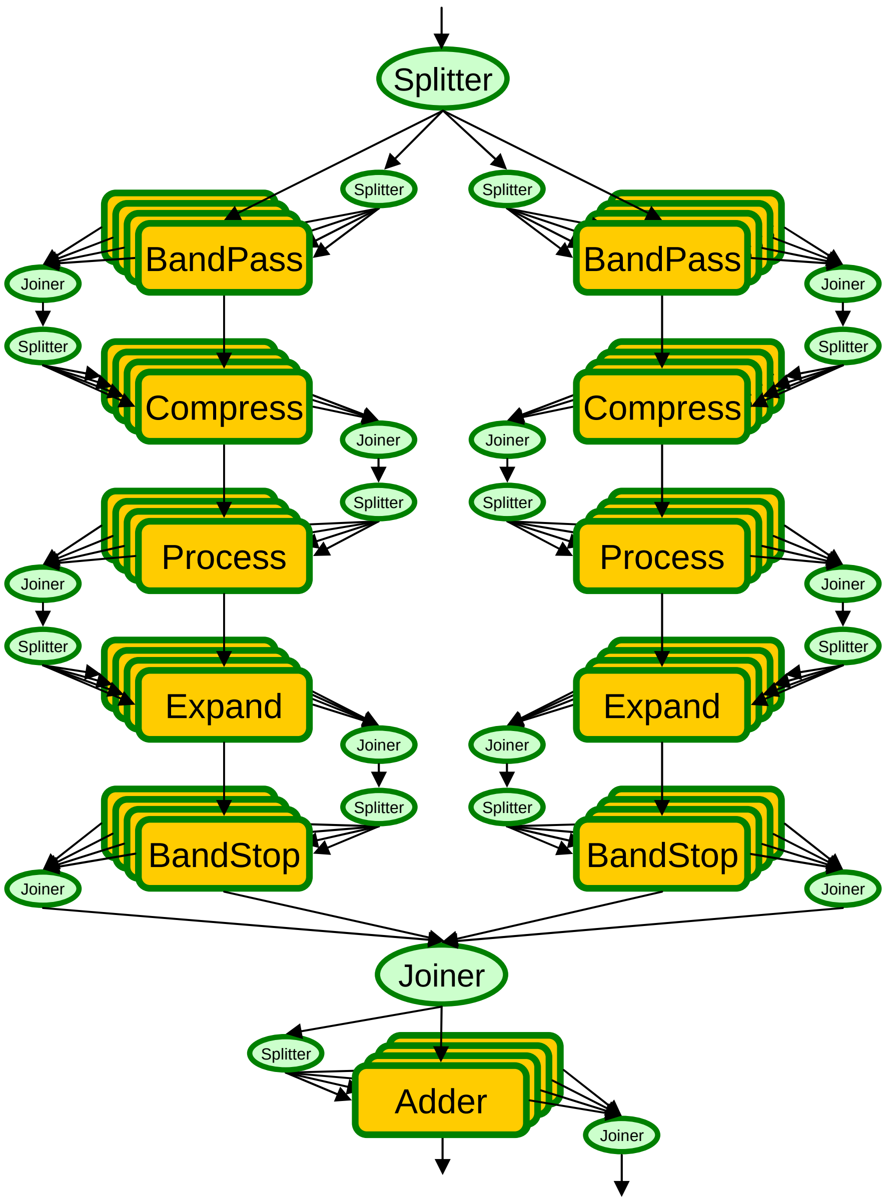
**Data parallelism**

– Analogous to DOALL loops

**Pipeline parallelism**

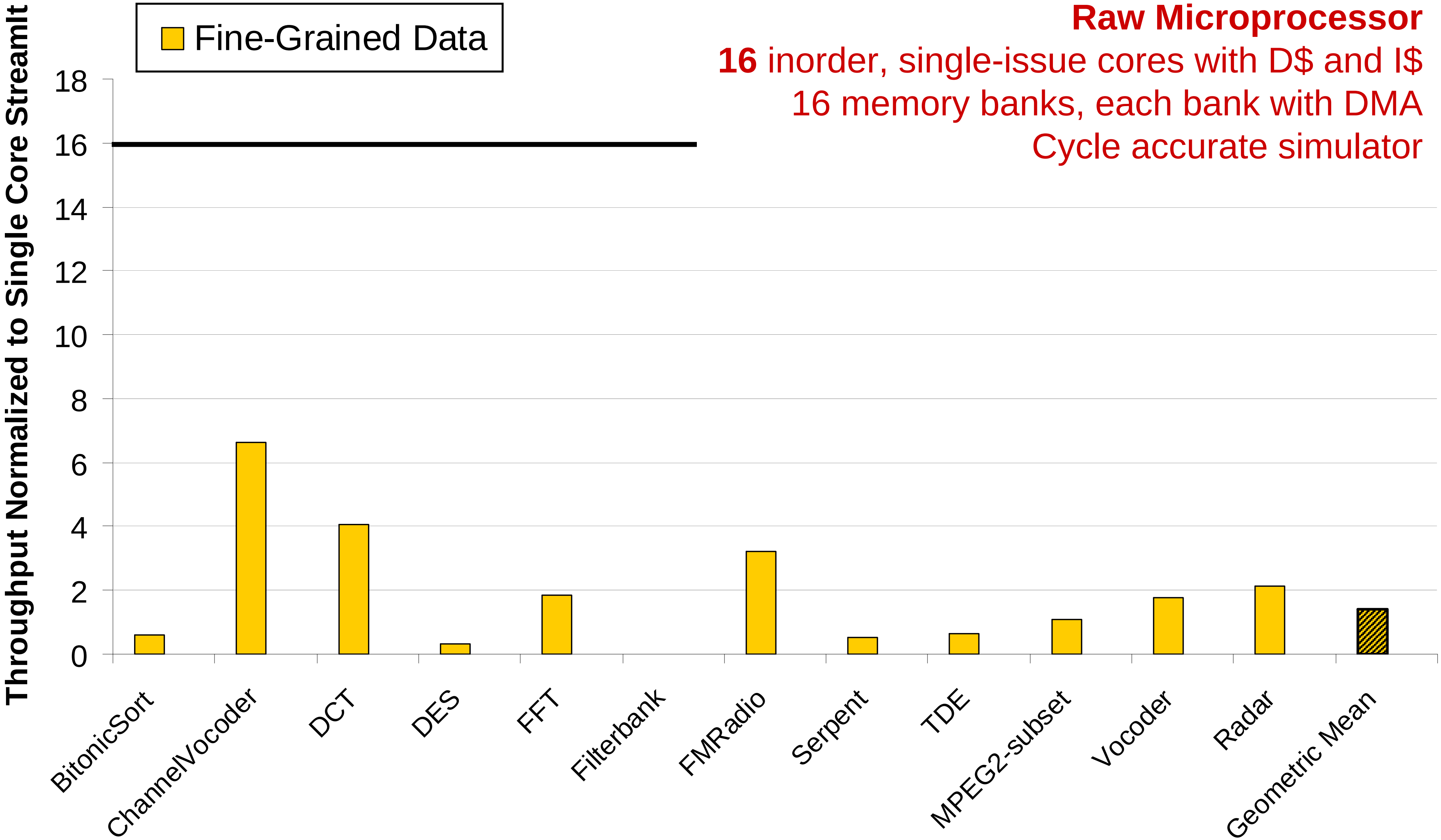
– Analogous to ILP that is exploited in hardware

# Baseline: Fine-Grained Data Parallelism

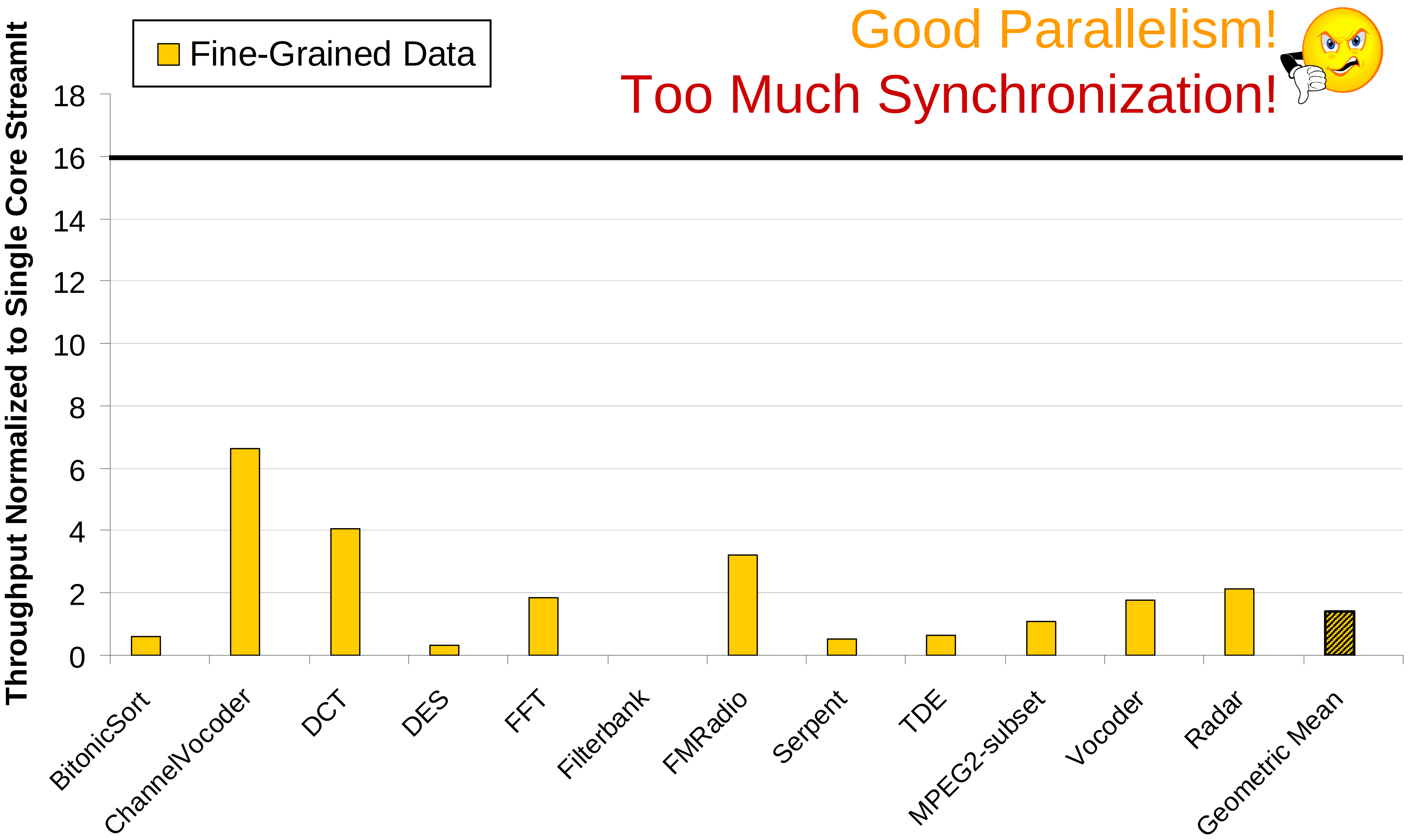


# Evaluation:
# Fine-Grained Data Parallelism

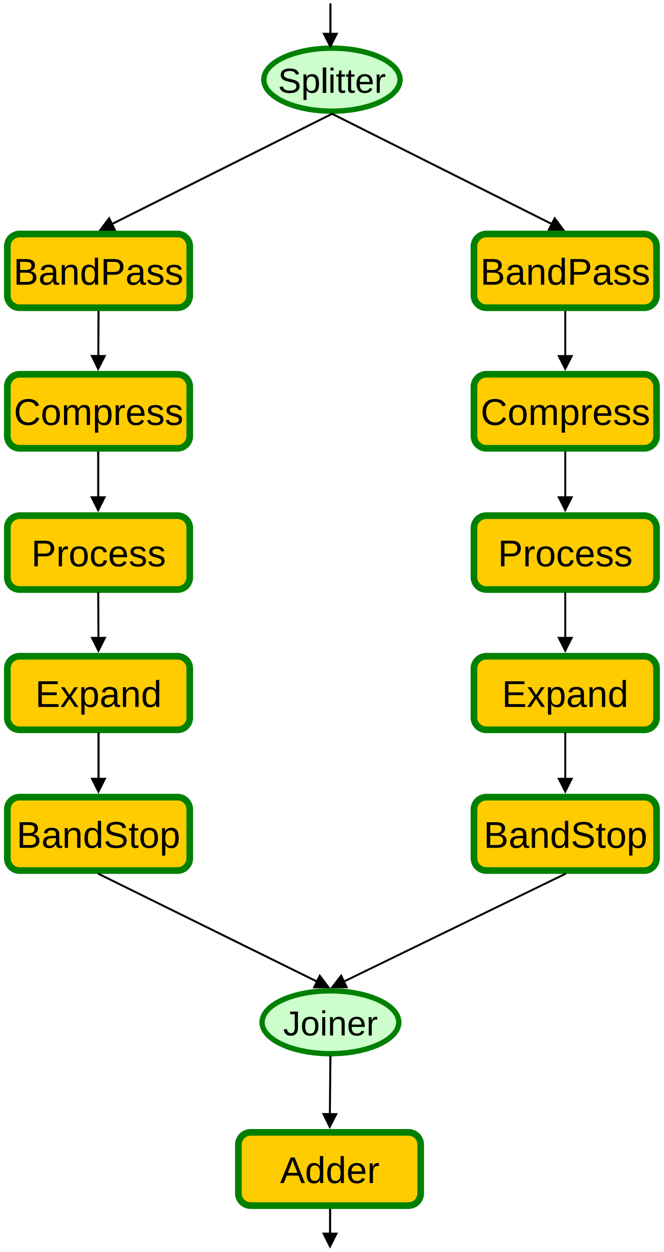
**Raw Microprocessor**
**16** inorder, single-issue cores with D$ and I$
16 memory banks, each bank with DMA
Cycle accurate simulator



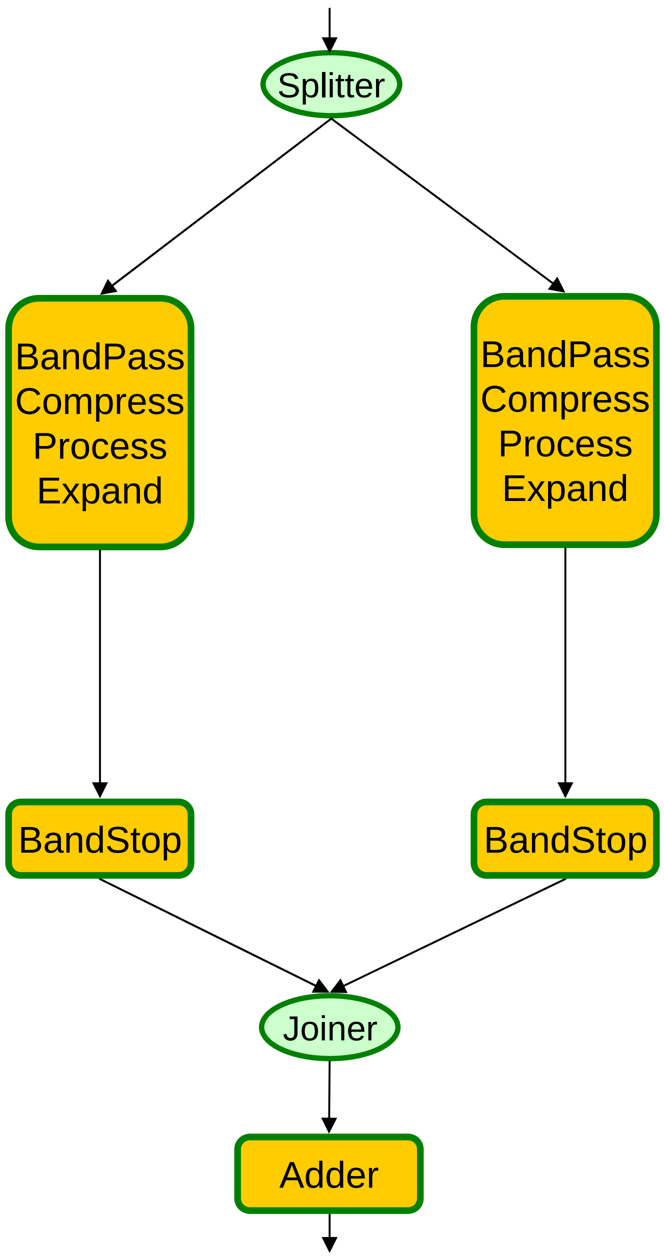
# Evaluation:
# Fine-Grained Data Parallelism



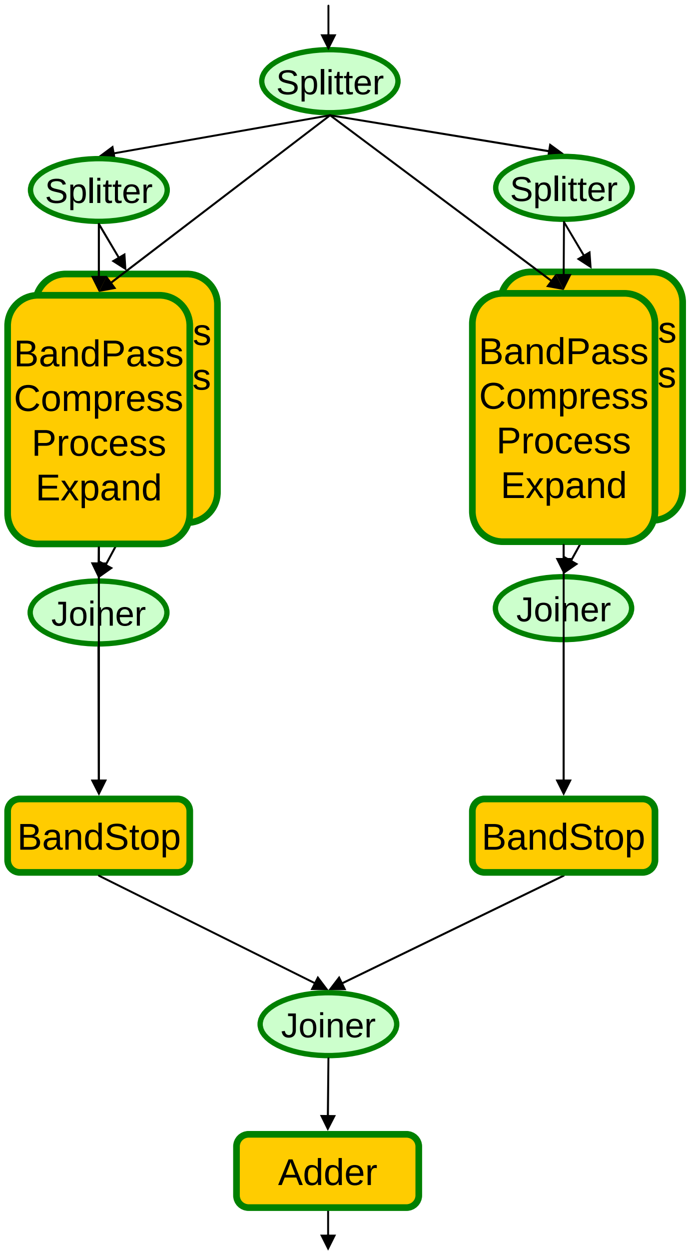
# Coarsening the Granularity



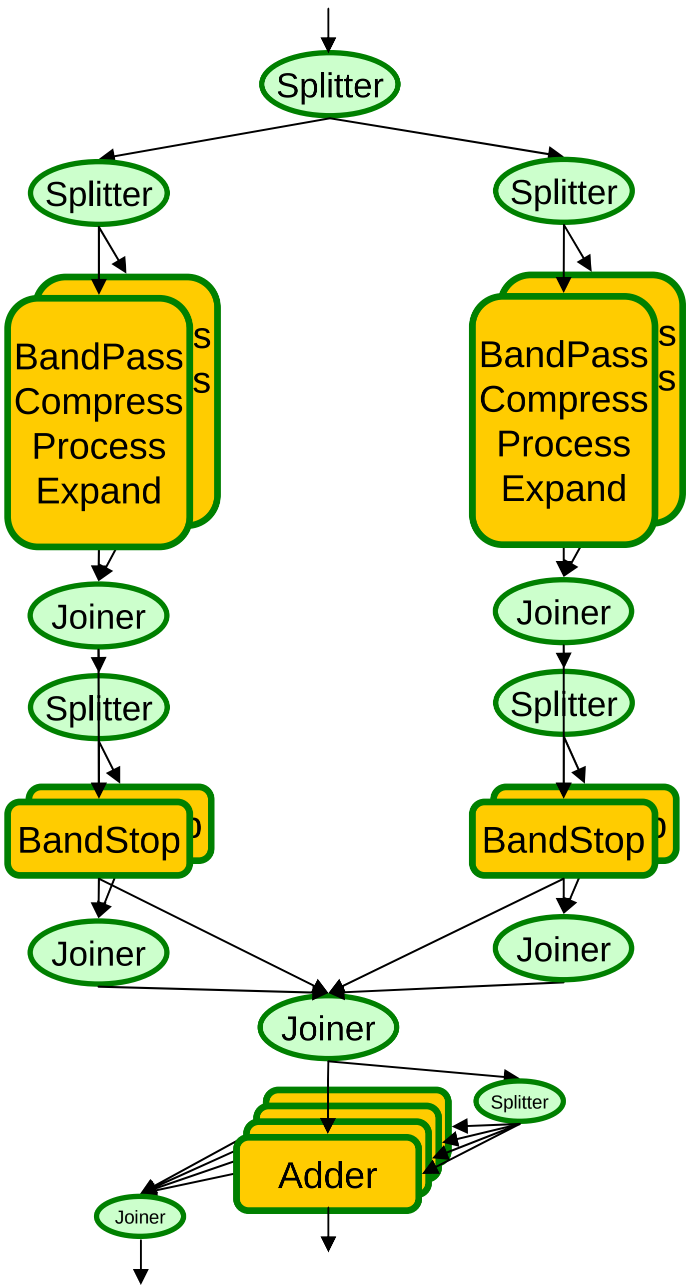
# Coarsening the Granularity



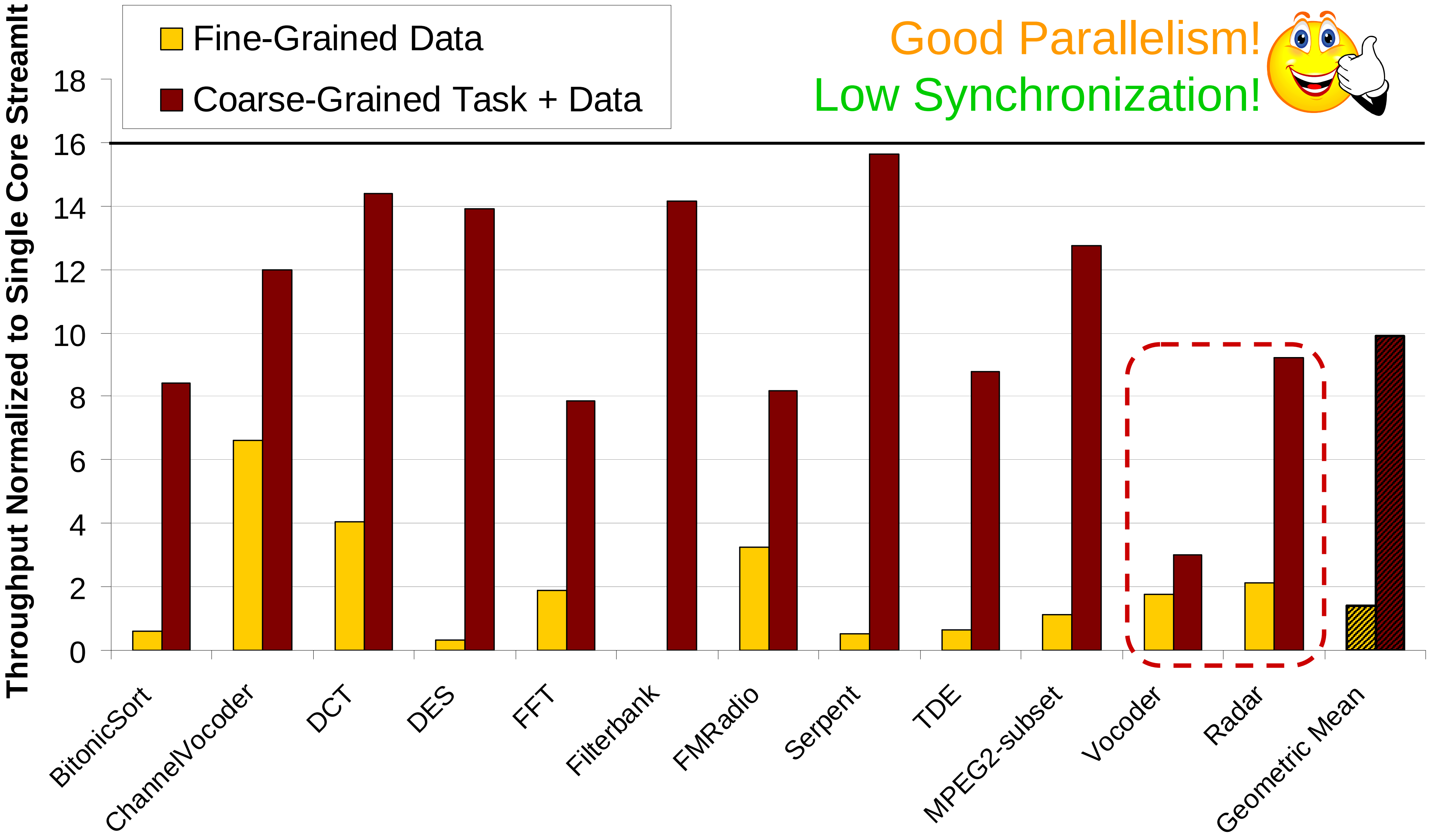
# Coarsening the Granularity



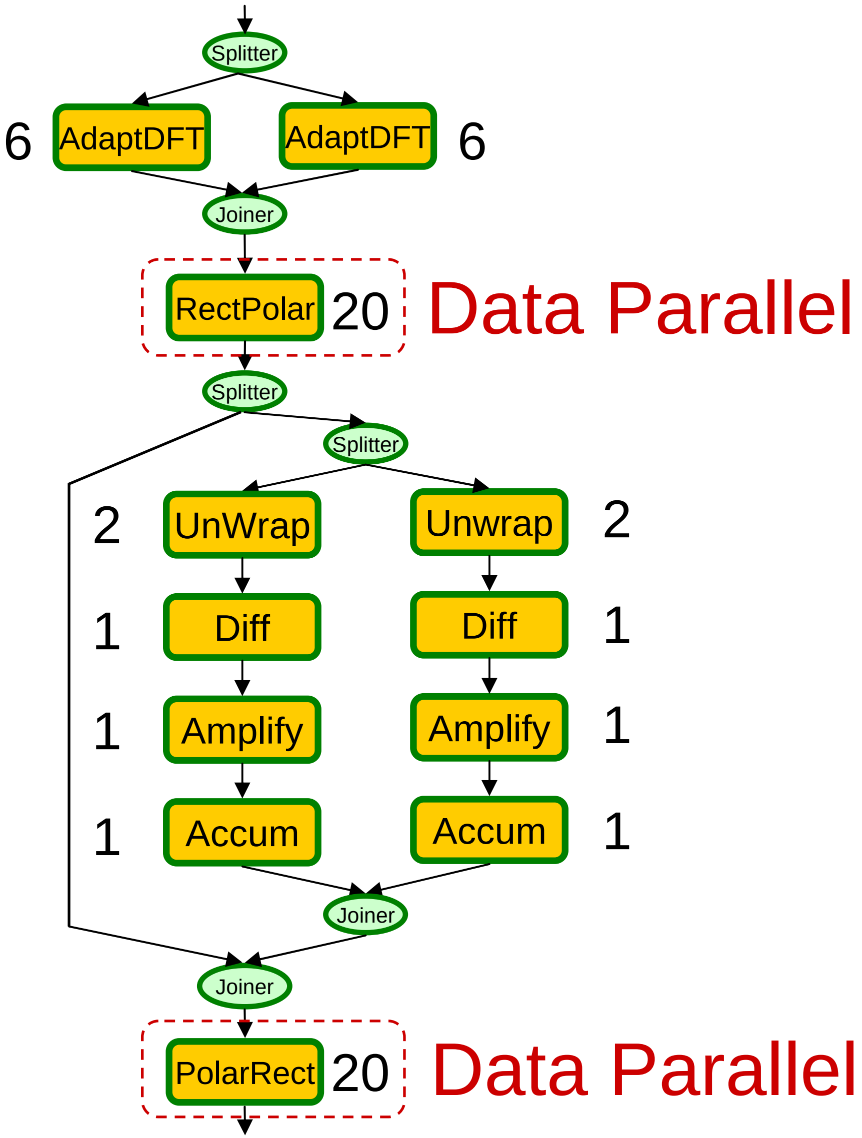
# Coarsening the Granularity



# Evaluation:
# Coarse-Grained Data Parallelism

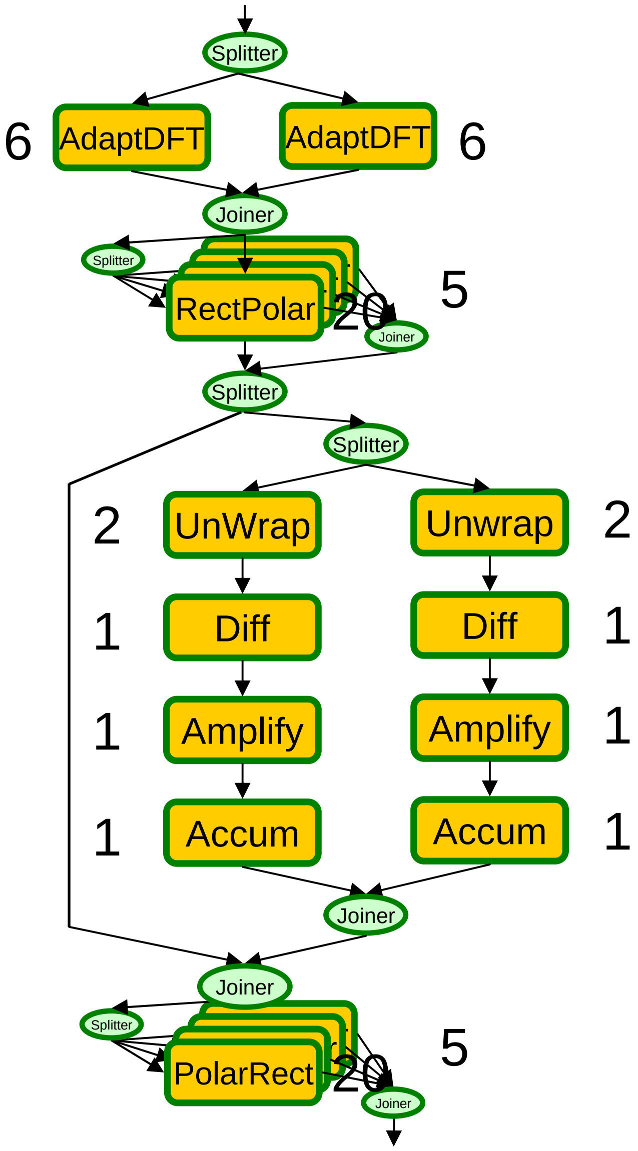


# Simplified Vocoder



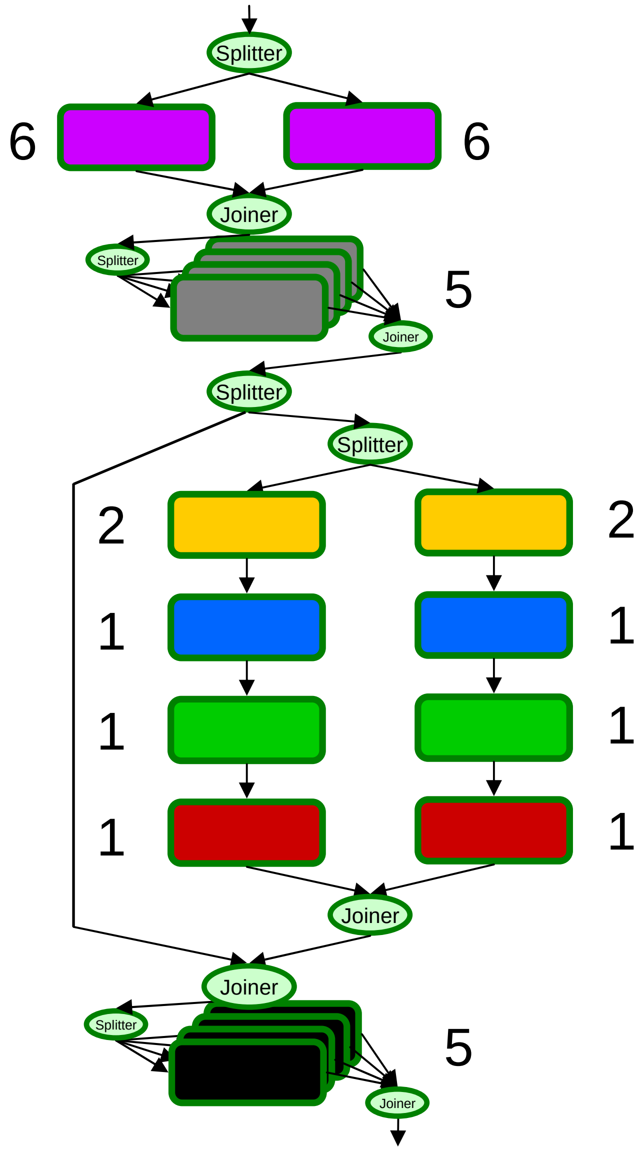

Target a 4-core machine

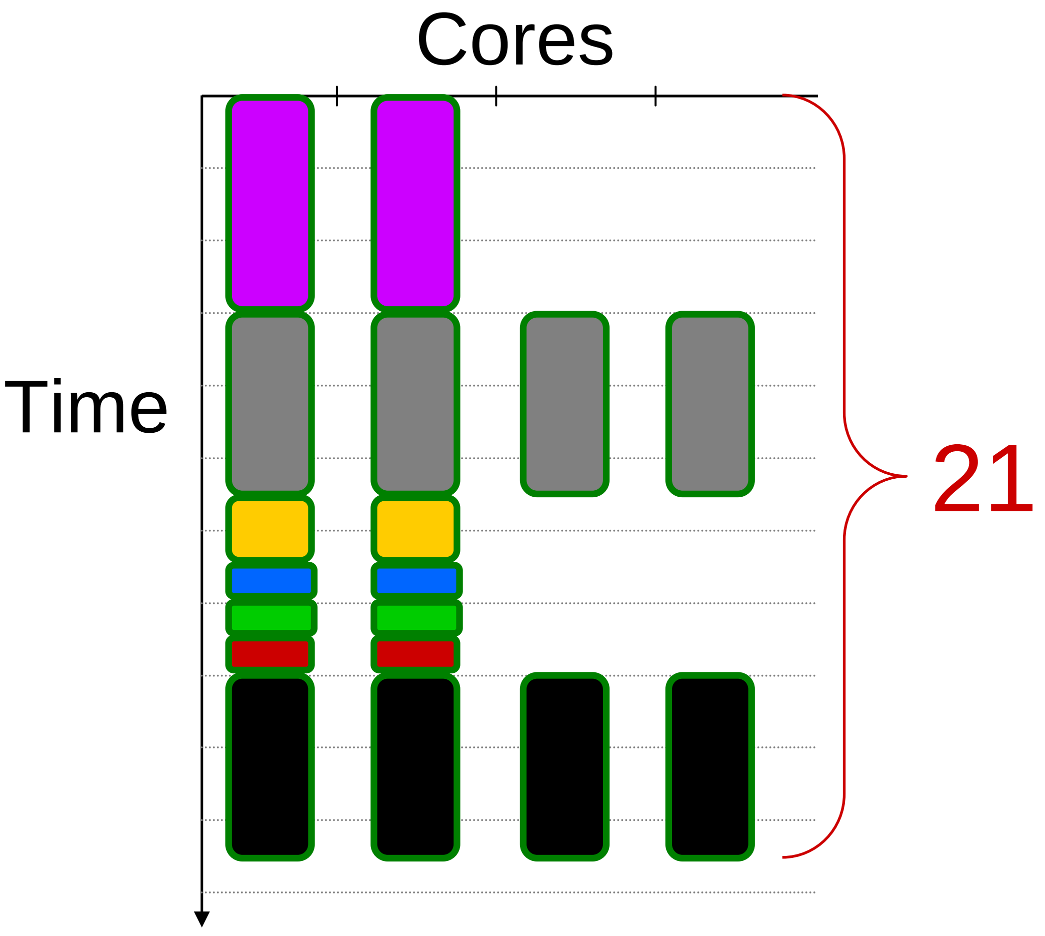
# Data Parallelize




Target a 4-core machine

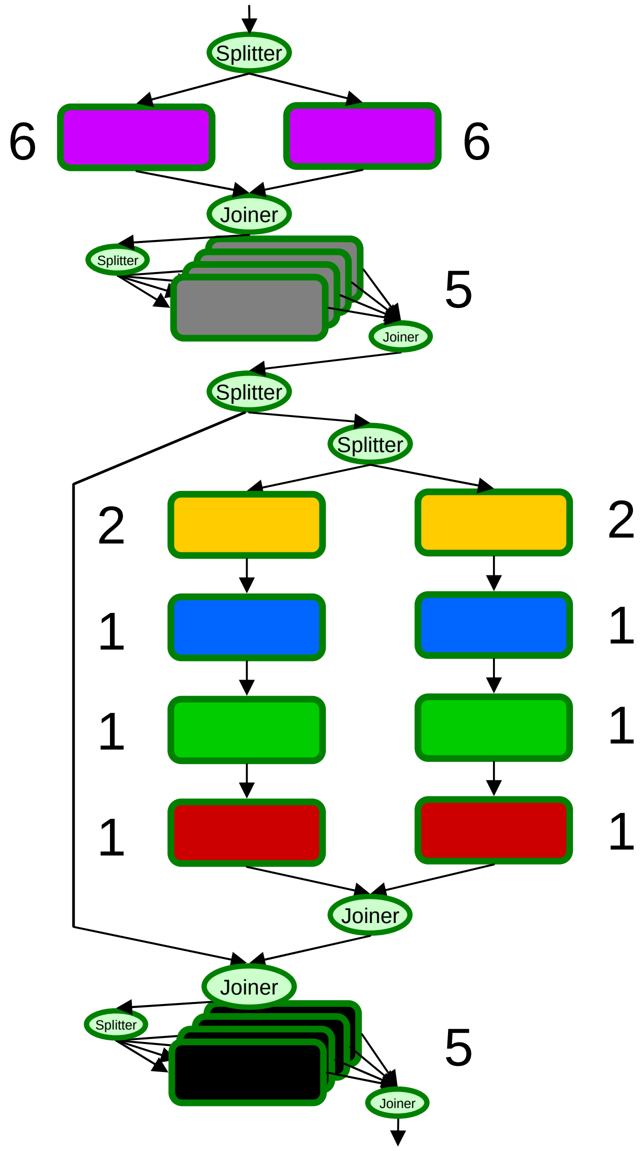
# Data + Task Parallel Execution

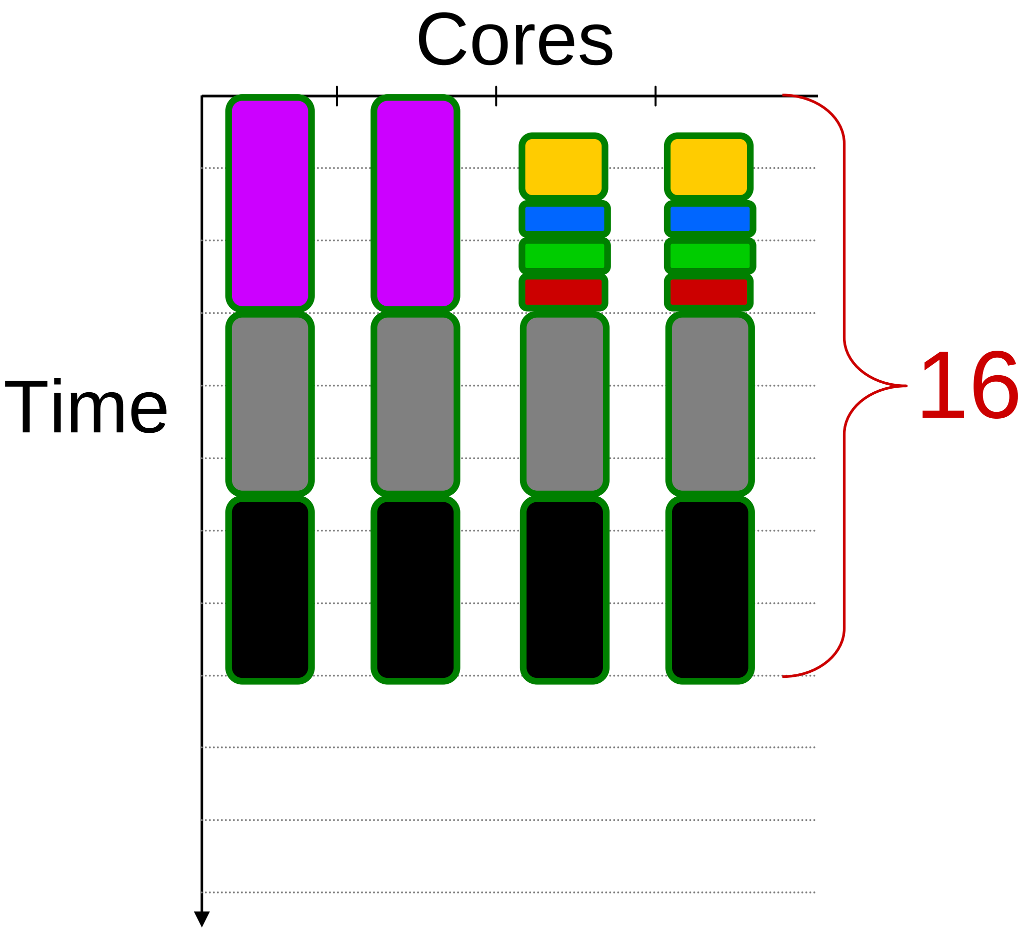



Target a 4-core machine

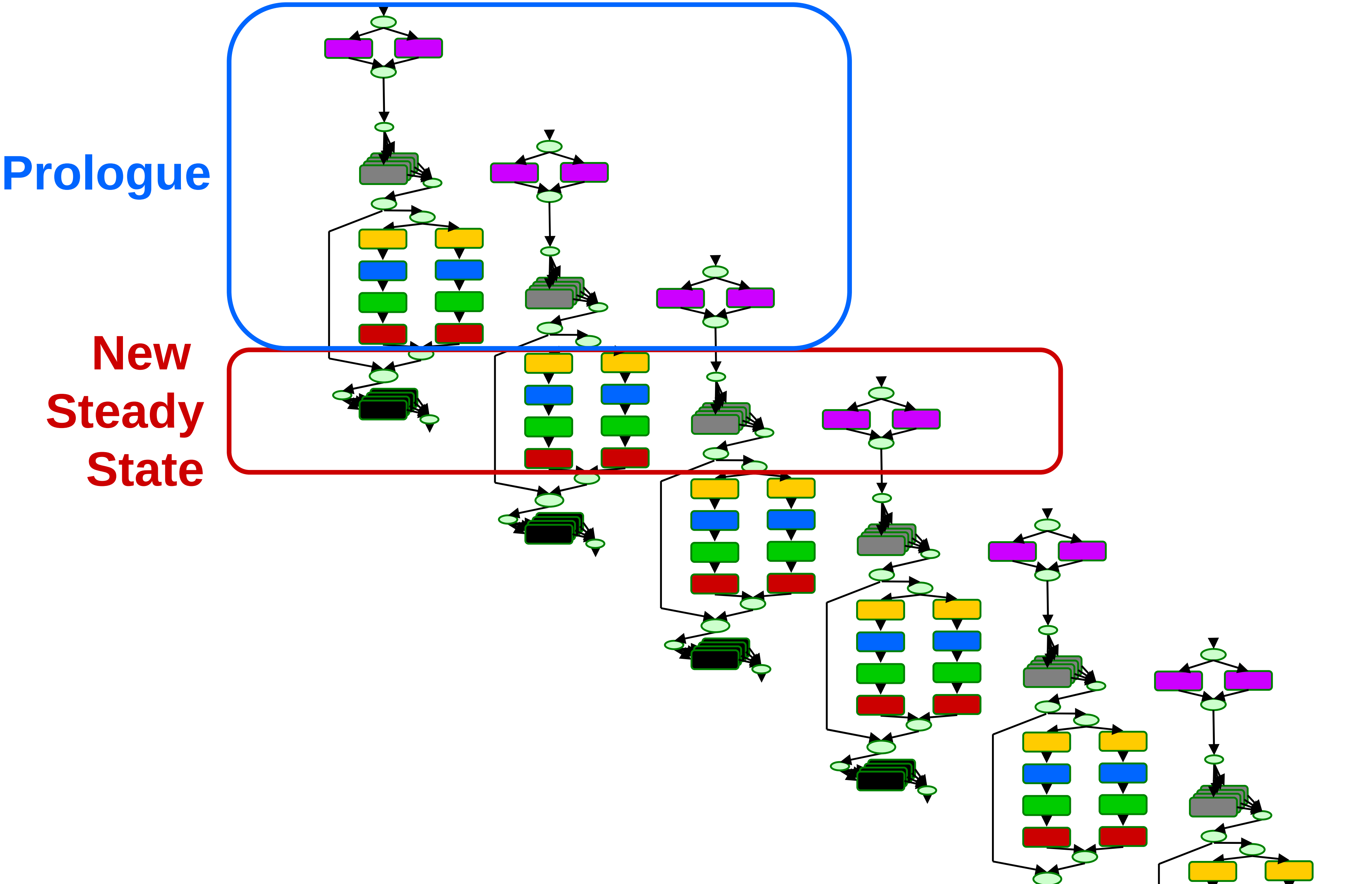
# We Can Do Better



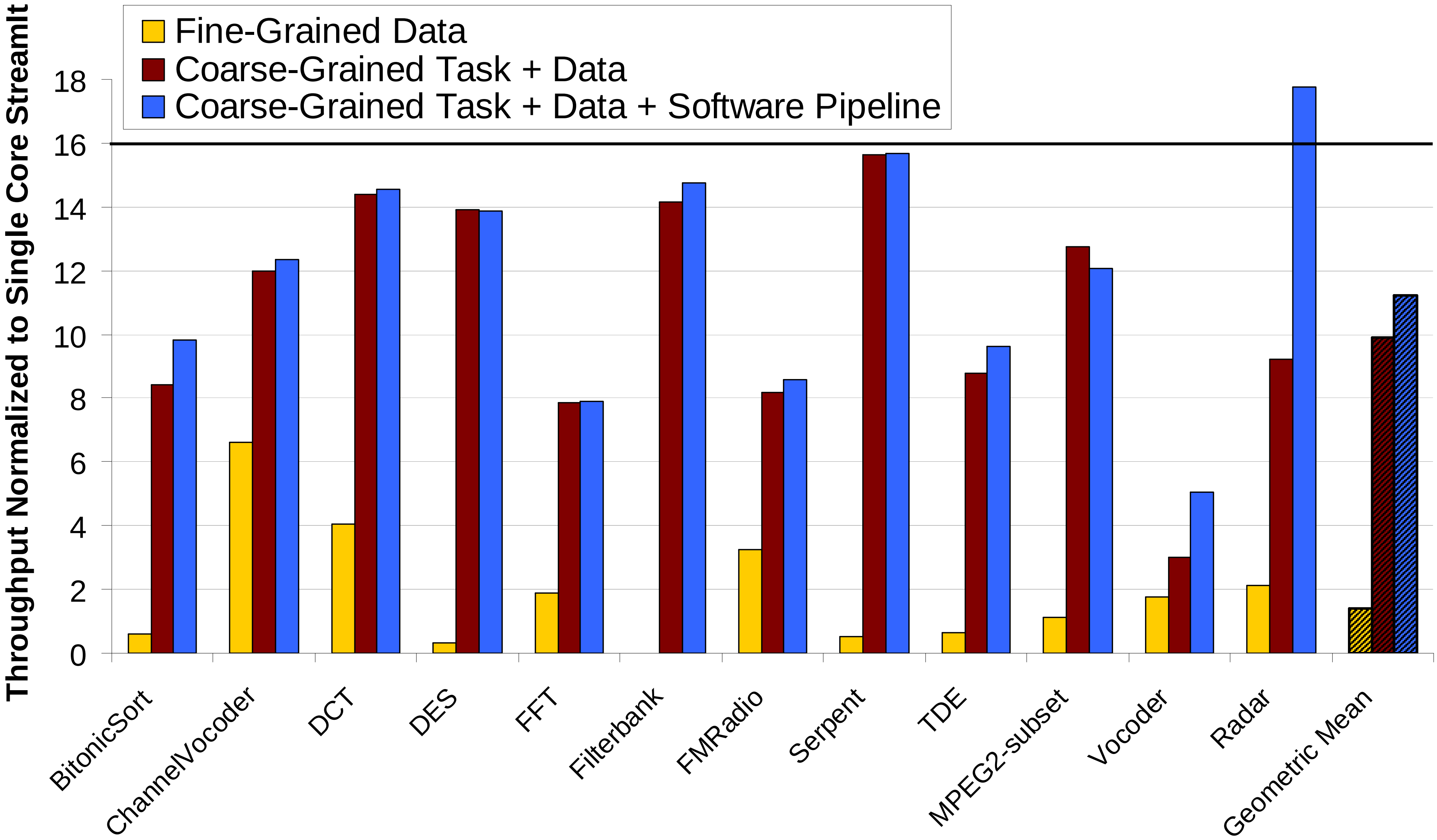




Target a 4-core machine

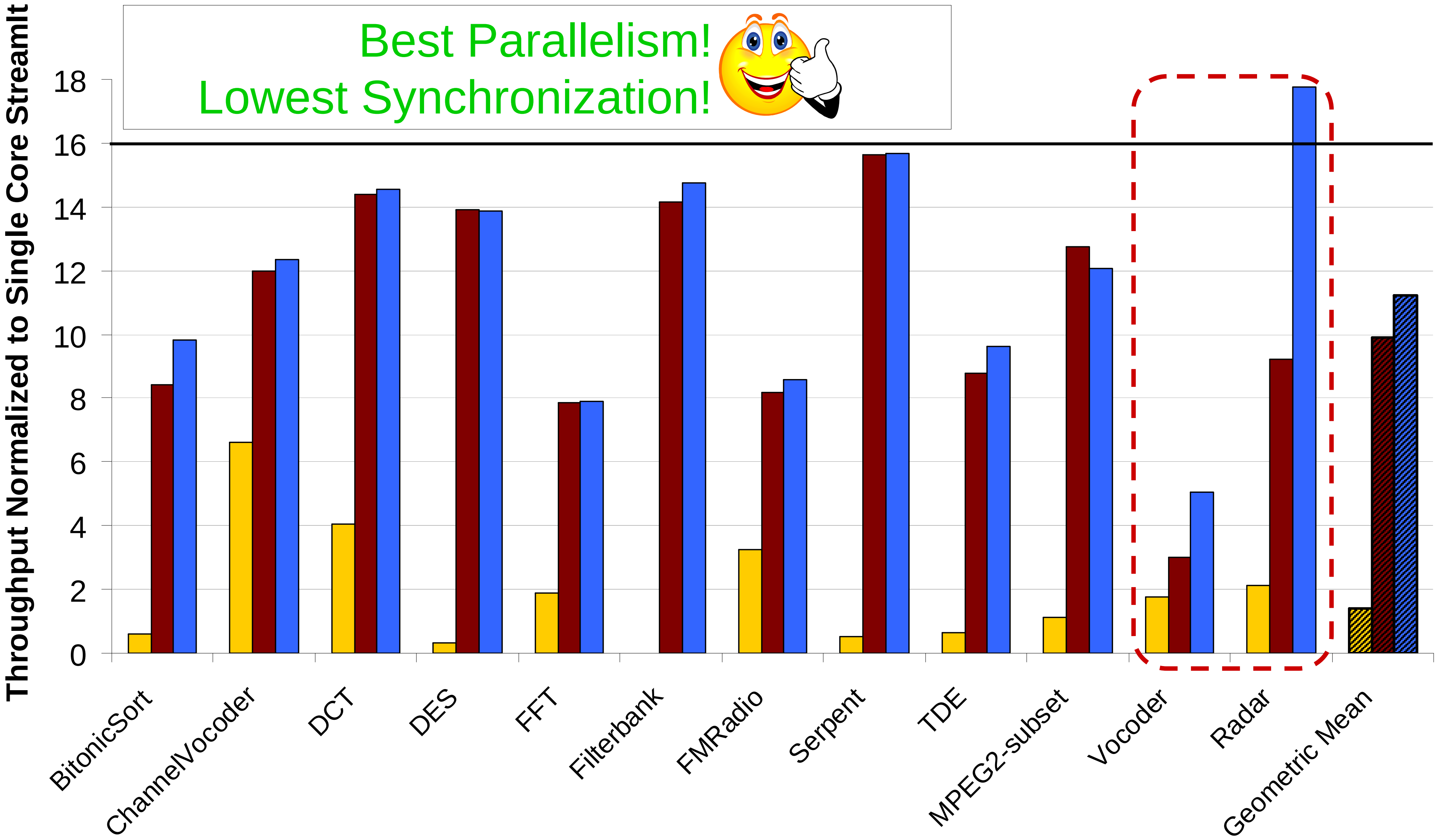
# Coarse-Grained Software Pipelining



# Evaluation: Coarse-Grained Task + Data + Software Pipelining

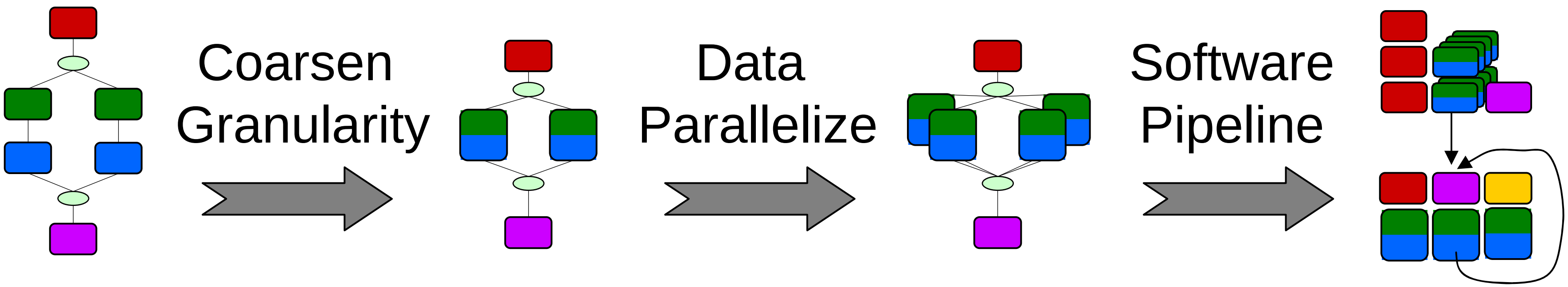


# Evaluation: Coarse-Grained Task + Data + Software Pipelining



# Parallelism:  Take Away

- **Stream programs have abundant parallelism**
  - However, parallelism is obfuscated in language like C
- **Stream languages enable new & effective mapping**




  - In C, analogous transformations impossibly complex
  - In StreamC or Brook, similar transformations possible
    [Khailany et al., IEEE Micro’01] [Buck et al., SIGGRAPH’04] [Das et al., PACT’06] […]
- **Results should extend to other multicores**
  - Parameters:  local memory, comm.-to-comp. cost
  - Preliminary results on Cell are promising [Zhang, dasCMP’07]

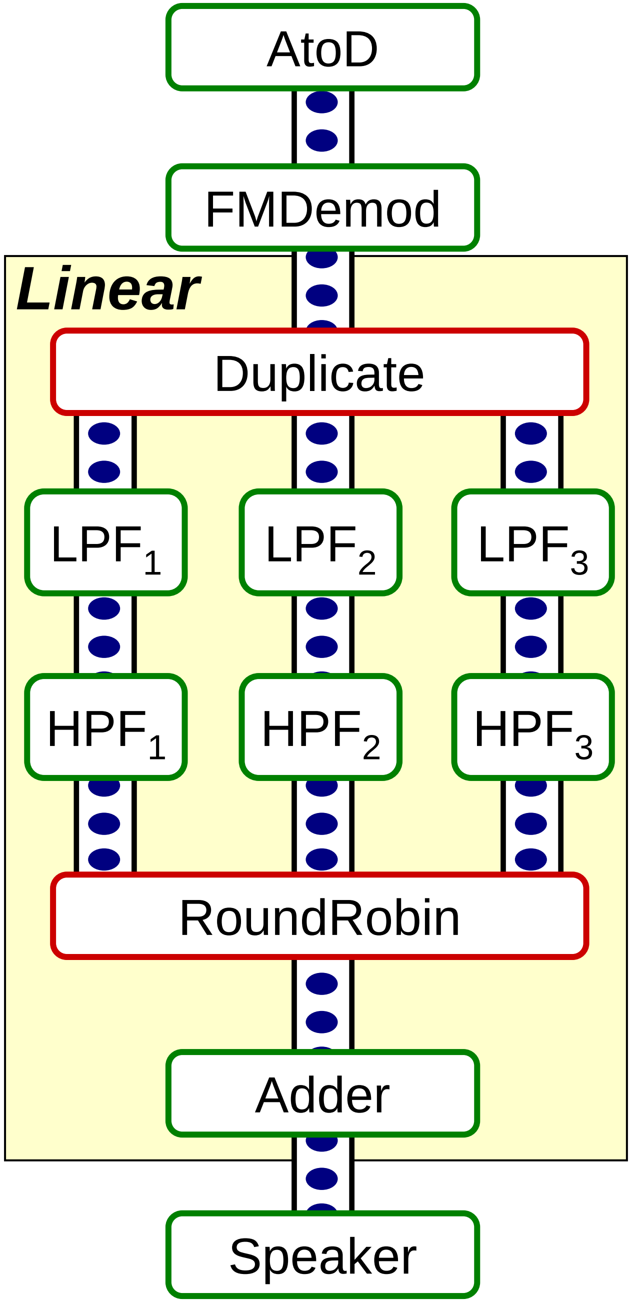
# Part 3:  Domain-Specific Optimizations

*Andrew Lamb, William Thies, Saman Amarasinghe (PLDI'03)*

*Sitij Agrawal, William Thies, Saman Amarasinghe (CASES'05)*

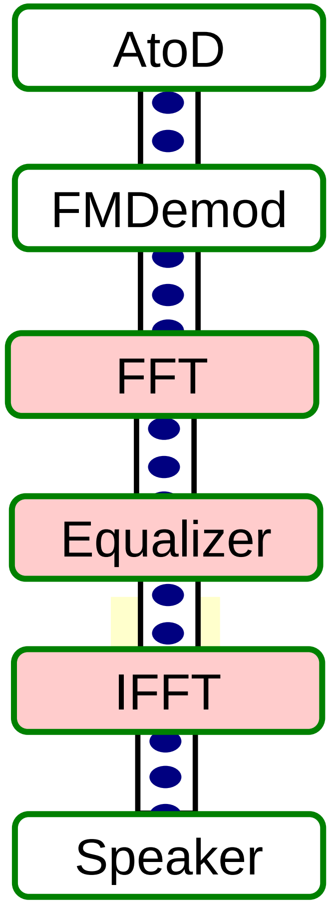
# DSP Optimization Process

- **Given specification of algorithm, minimize the computation cost**



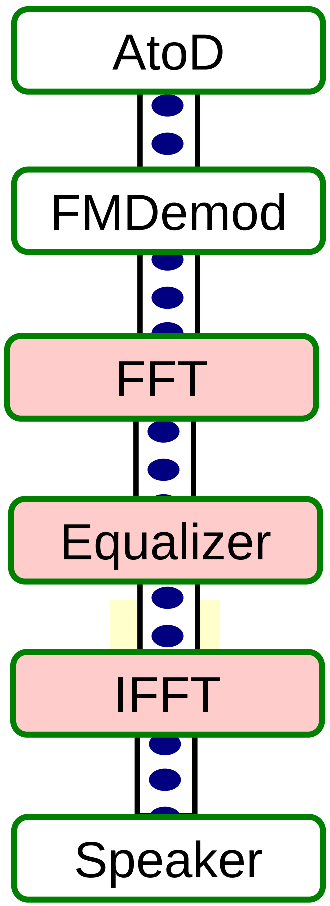
# DSP Optimization Process

- **Given specification of algorithm, minimize the computation cost**
  - Currently done by hand (MATLAB)



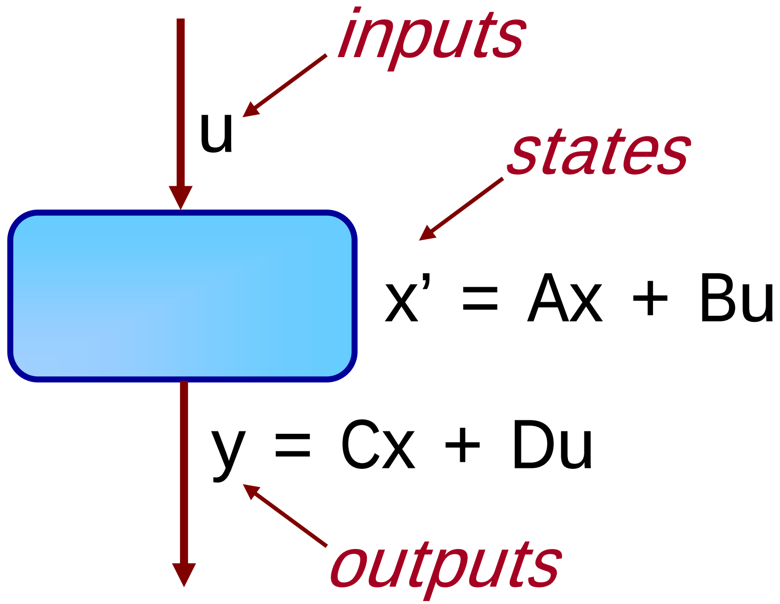
# DSP Optimization Process

- **Given specification of algorithm, minimize the computation cost**
  - Currently done by hand (MATLAB)
- **Can compiler replace DSP expert?**
  - Library generators limited [Spiral] [FFTW] [ATLAS]
  - Enable unified development environment

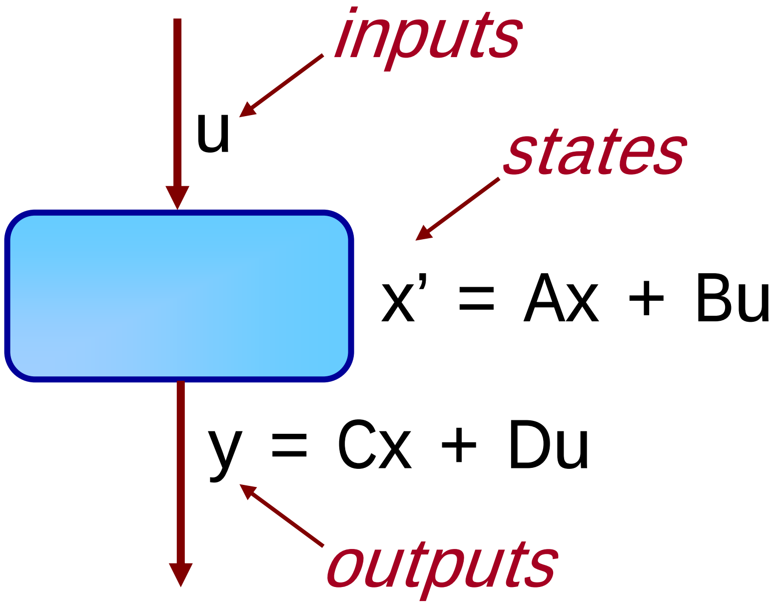


# Focus:  Linear State Space Filters

- **Properties:**
  - Outputs are linear function of inputs and states
  - New states are linear function of inputs and states
- **Most common target of DSP optimizations**
  - FIR / IIR filters
  - Linear difference equations
  - Upsamplers / downsamplers
  - DCTs



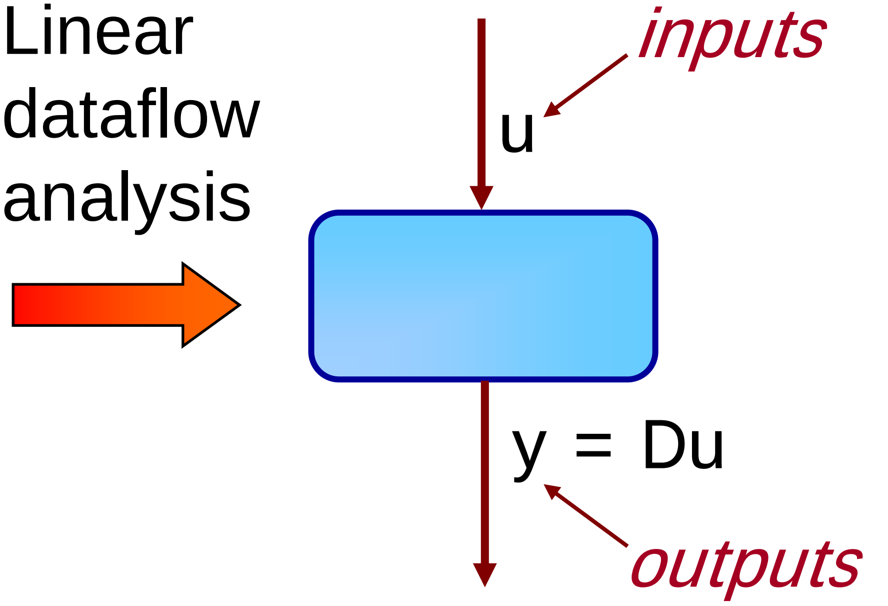
# Focus: Linear State Space Filters



# Focus: Linear Filters

```
float->float filter Scale {
  work push 2 pop 1 {
    float u = pop();
    push(u);
    push(2*u);
  }
}
```

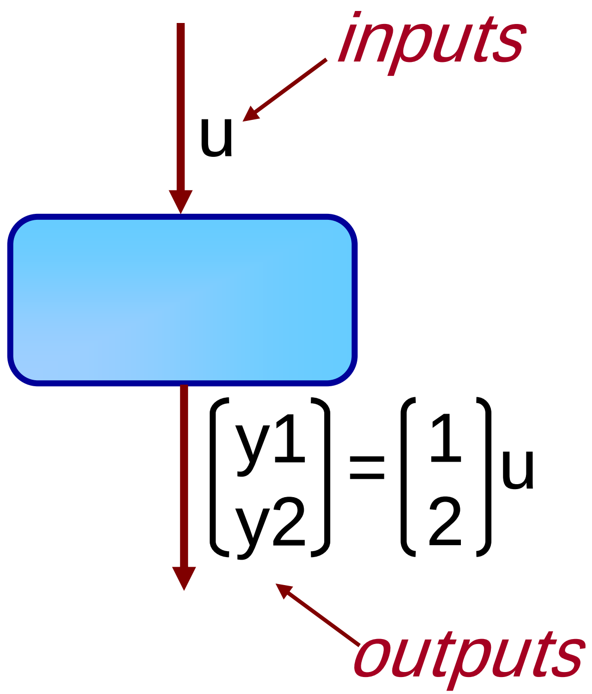
Linear dataflow analysis



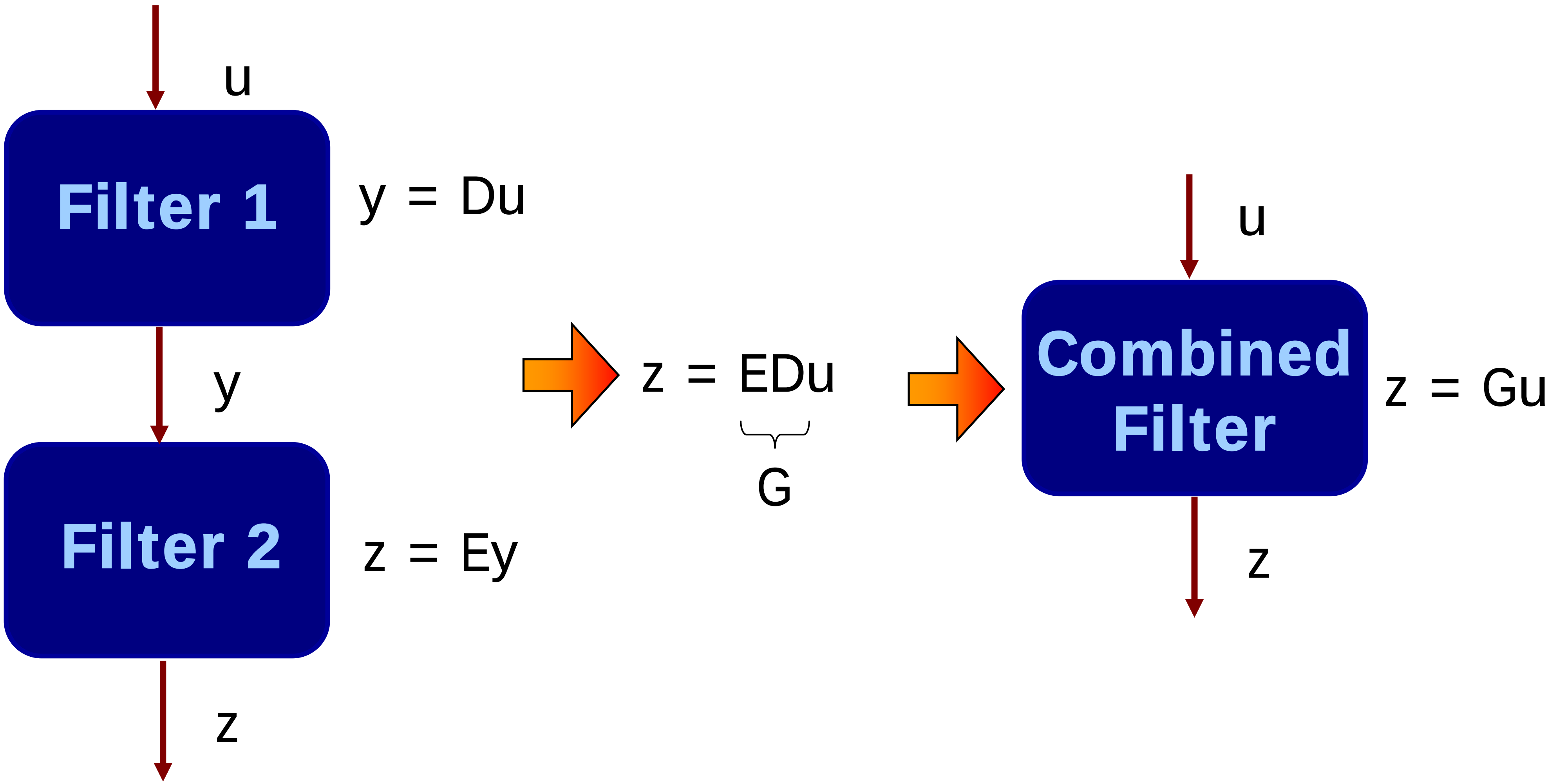
# Focus: Linear Filters

```
float->float filter Scale {
  work push 2 pop 1 {
    float u = pop();
    push(u);
    push(2*u);
  }
}
```

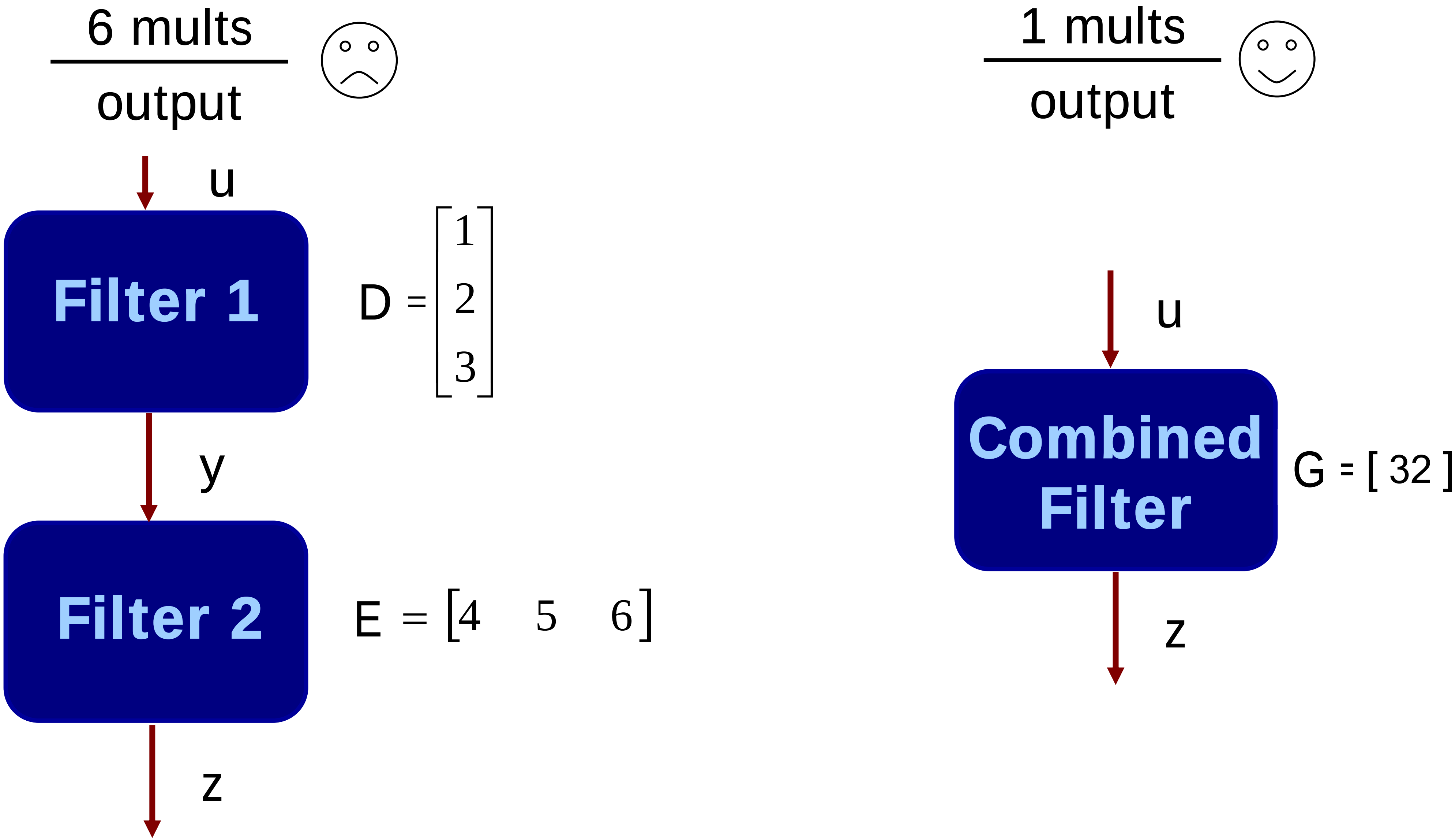
Linear dataflow analysis



# Combining Adjacent Filters

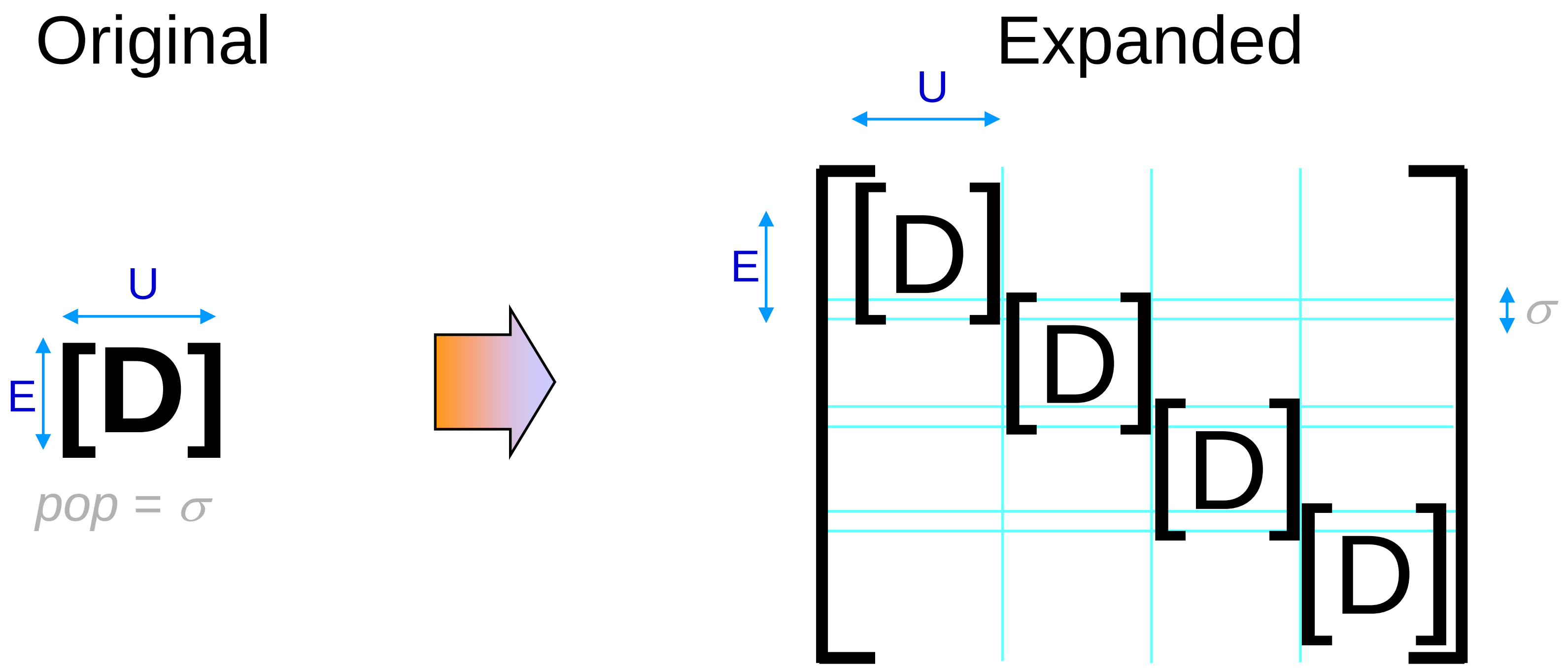


# Combination Example



# The General Case

- **If matrix dimensions mis-match? Matrix *expansion*:**



# The General Case

- **If matrix dimensions mis-match? Matrix *expansion*:**

$$
\begin{aligned}
\mathbf{A^e} &= \mathbf{A}^n \mathbf{A_{pre}} \\
\mathbf{B^e} &= \begin{bmatrix} \mathbf{A^n B_{pre}} & \mathbf{A^{n-1} B} & \mathbf{A^{n-2} B} & \ldots & \mathbf{B} \end{bmatrix} \\
\mathbf{C^e} &= \begin{bmatrix} \mathbf{CA_{pre}} \\ \mathbf{CAA_{pre}} \\ \ldots \\ \mathbf{CA^{n-1} A_{pre}} \end{bmatrix}
\end{aligned}
$$

$$
\mathbf{D^e} = \begin{bmatrix}
\mathbf{CB_{pre}} & \mathbf{D} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\
\mathbf{CAB_{pre}} & \mathbf{CB} & \mathbf{D} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\
\mathbf{CA^2 B_{pre}} & \mathbf{CAB} & \mathbf{CB} & \mathbf{D} & \ldots & \mathbf{0} & \mathbf{0} \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\
\mathbf{CA^{n-1} B_{pre}} & \mathbf{CA}^{n-2}\mathbf{B} & \mathbf{CA}^{n-3}\mathbf{B} & \mathbf{CA}^{n-3}\mathbf{B} & \ldots & \mathbf{CB} & \mathbf{D}
\end{bmatrix}
$$

# The General Case

### *Pipelines*

$$
\begin{aligned}
A &= \begin{bmatrix} A_1 & 0 \\ B_2C_1 & A_2 \end{bmatrix} \\
B &= \begin{bmatrix} B_1 \\ B_2D_1 \end{bmatrix} \\
C &= \begin{bmatrix} D_2C_1 & C_2 \end{bmatrix} \\
D &= D_2D_1
\end{aligned}
$$

$$
\begin{aligned}
A_{pre} &= \begin{bmatrix} {A_1}^e & 0 \\ B_{pre2}{C_1}^e & A_{pre2} \end{bmatrix} \\
B_{pre} &= \begin{bmatrix} {B_1}^e \\ B_{pre2}{D_1}^e \end{bmatrix} \\
\overrightarrow{\mathbf{initVec}} &= \begin{bmatrix} \overrightarrow{initVec_1} \\ \overrightarrow{initVec_2} \end{bmatrix}
\end{aligned}
$$

### *Feedback Loops*

$$
\begin{aligned}
\vec{x_1} &= A_1\vec{x_1} + B_1\vec{u_1} = A_1\vec{x_1} + B_1\vec{y} = A_1\vec{x_1} + B_1(C_2\vec{x_2} + D_{2\_1}\vec{u} + D_{2\_2}C_3\vec{x_3}) \\
&= A_1\vec{x_1} + B_1C_2\vec{x_2} + B_1D_{2\_1}\vec{u} + B_1D_{2\_2}C_3\vec{x_3}
\end{aligned}
$$

$$
\begin{aligned}
\vec{x_2} &= A_2\vec{x_2} + B_2\vec{u_2} = A_2\vec{x_2} + B_{2\_1}\vec{u} + B_{2\_2}\vec{y_3} = A_2\vec{x_2} + B_{2\_1}\vec{u} + B_{2\_2}C_3\vec{x_3} \\
\vec{y_2} &= C_2\vec{x_2} + D_2\vec{u_2} = C_2\vec{x_2} + D_{2\_1}\vec{u} + D_{2\_2}\vec{y_3} = C_2\vec{x_2} + D_{2\_1}\vec{u} + D_{2\_2}C_3\vec{x_3}
\end{aligned}
$$

$$
\begin{aligned}
\vec{x_3} &= A_3\vec{x_3} + B_3\vec{u_3} = A_3\vec{x_3} + B_3\vec{y_1} = A_3\vec{x_3} + B_3(C_1\vec{x_1} + D_1\vec{u_1}) \\
&= A_3\vec{x_3} + B_3(C_1\vec{x_1} + D_1\vec{y}) = A_3\vec{x_3} + B_3(C_1\vec{x_1} + D_1(C_2\vec{x_2} + D_{2\_1}\vec{u} + D_{2\_2}C_3\vec{x_3})) \\
&= A_3\vec{x_3} + B_3C_1\vec{x_1} + B_3D_1C_2\vec{x_2} + B_3D_1D_{2\_1}\vec{u} + B_3D_1D_{2\_2}C_3\vec{x_3}
\end{aligned}
$$

# The General Case

***Splitjoins***

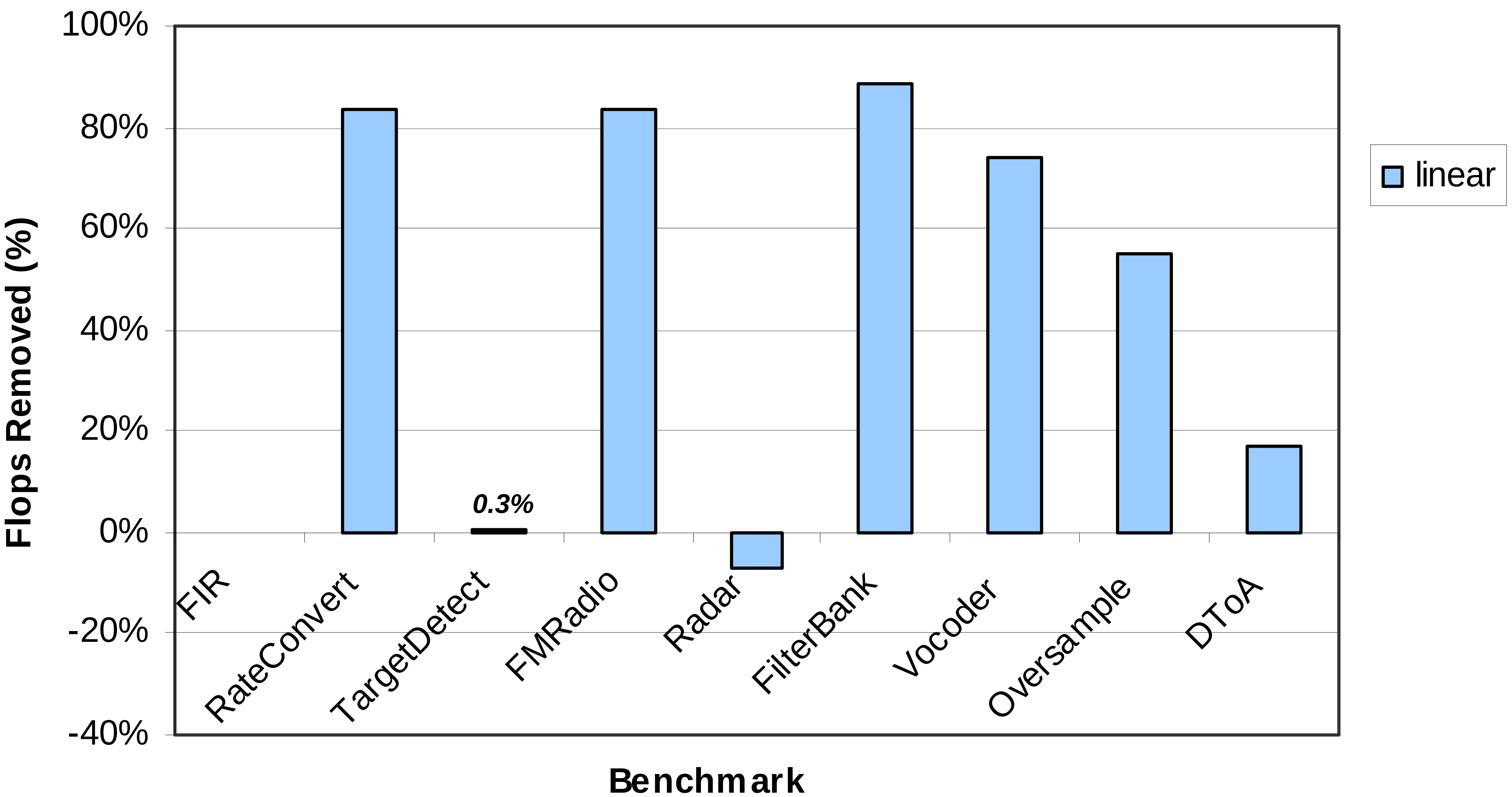
$$A = \begin{bmatrix} A_s & 0 & 0 & \dots & 0 \\ A_{1rs} & A_{1rr} & 0 & \dots & 0 \\ A_{2rs} & 0 & A_{2rr} & \dots & 0 \\ \dots & \dots & \dots & \dots & \\ A_{krr} & 0 & 0 & \dots & A_{krs} \end{bmatrix} B = \begin{bmatrix} B_s \\ B_{1r} \\ B_{2r} \\ \dots \\ B_{kr} \end{bmatrix}$$

$$C_i = \begin{bmatrix} C_{is1} & C_{ir1} \\ C_{is2} & C_{ir2} \\ \dots & \dots \\ C_{isexecutions} & C_{irexecutions} \end{bmatrix} \quad D_i = \begin{bmatrix} D_{i1} \\ D_{i2} \\ \dots \\ D_{iexecutions} \end{bmatrix}$$

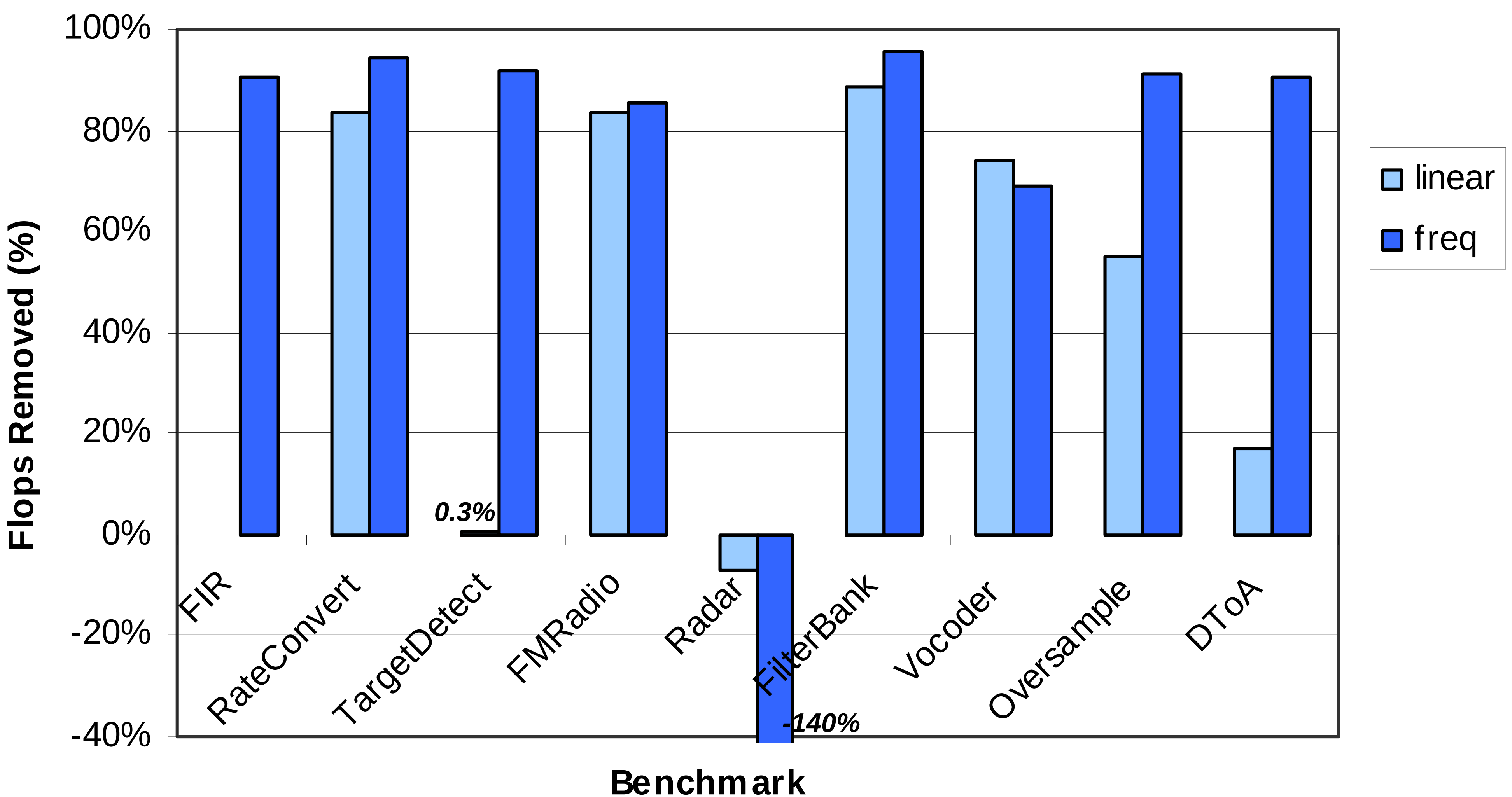
$$C = \begin{bmatrix} C_{1s1} & C_{1r1} & 0 & \dots & 0 \\ C_{2s1} & C_{2r1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ C_{ks1} & 0 & 0 & \dots & C_{kr1} \\ \dots & \dots & \dots & \dots & \dots \\ C_{1sk} & C_{1rk} & 0 & \dots & 0 \\ C_{2sk} & C_{2rk} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \\ C_{ksk} & 0 & 0 & \dots & C_{krk} \end{bmatrix} \quad D = \begin{bmatrix} D_{11} \\ D_{21} \\ \dots \\ D_{k1} \\ \dots \\ D_{1k} \\ D_{2k} \\ \dots \\ D_{kk} \end{bmatrix}$$

$$A_{pre} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & A_{pre1rr} & 0 & \dots & 0 \\ 0 & 0 & A_{pre2rr} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & A_{prekrr} \end{bmatrix} \quad B_{pre} = \begin{bmatrix} B_{pres} \\ B_{pre1r} \\ B_{pre2r} \\ \dots \\ B_{prekr} \end{bmatrix} \quad \overrightarrow{initVec} = \begin{bmatrix} \vec{0} \\ \overrightarrow{initVec_{1r}} \\ \overrightarrow{initVec_{2r}} \\ \dots \\ \overrightarrow{initVec_{kr}} \end{bmatrix}$$

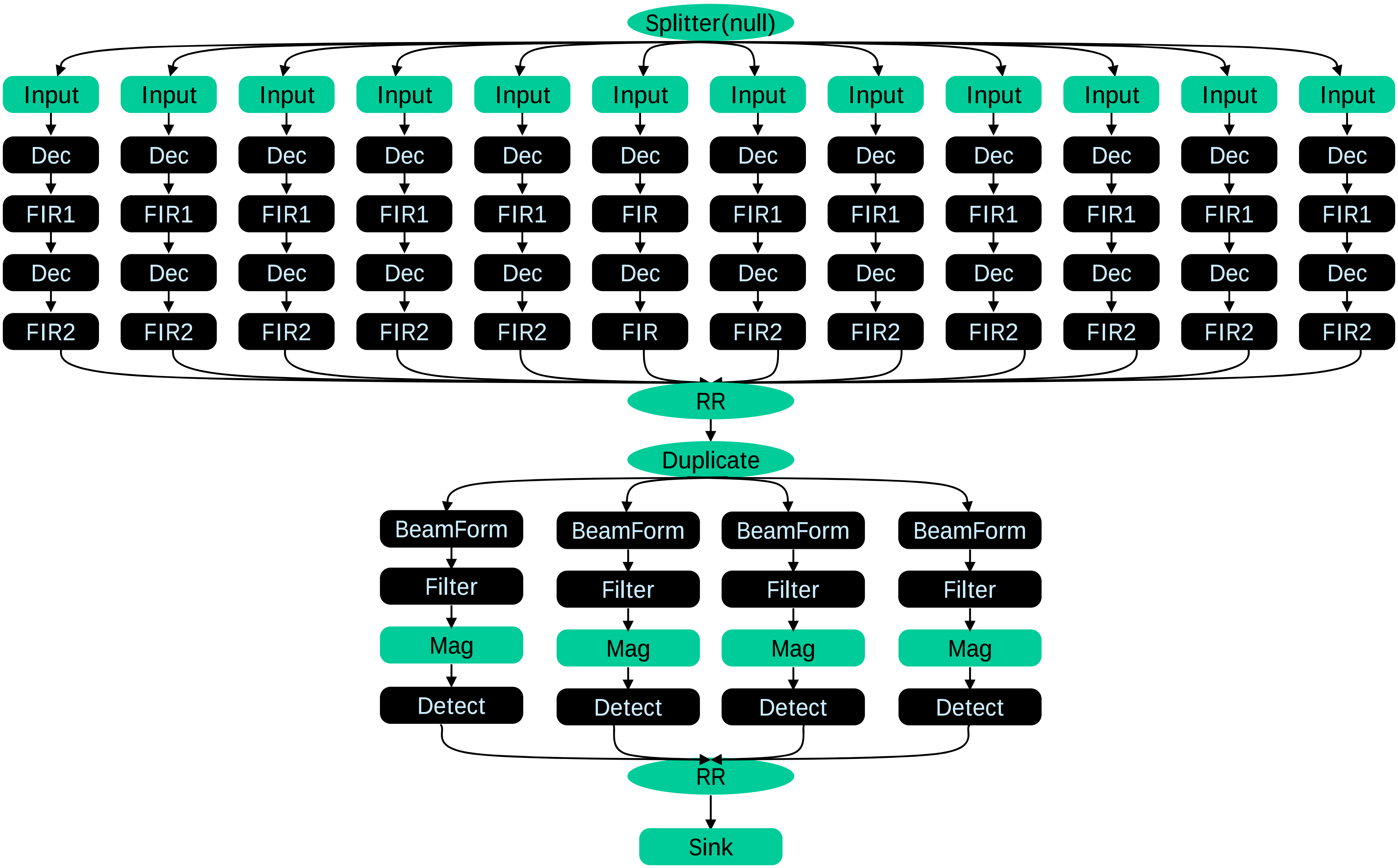
# Floating-Point Operations Reduction



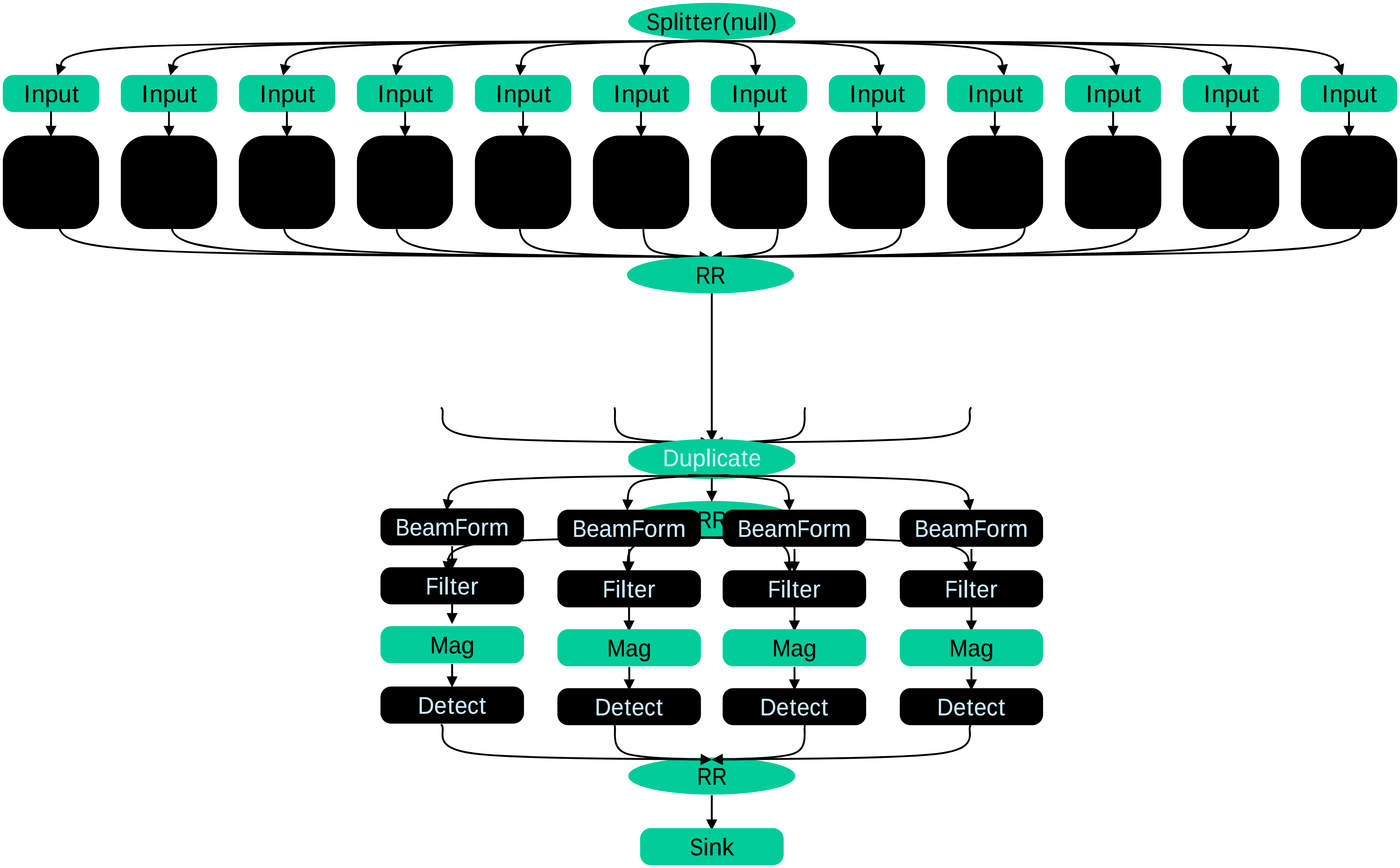
# Floating-Point Operations Reduction



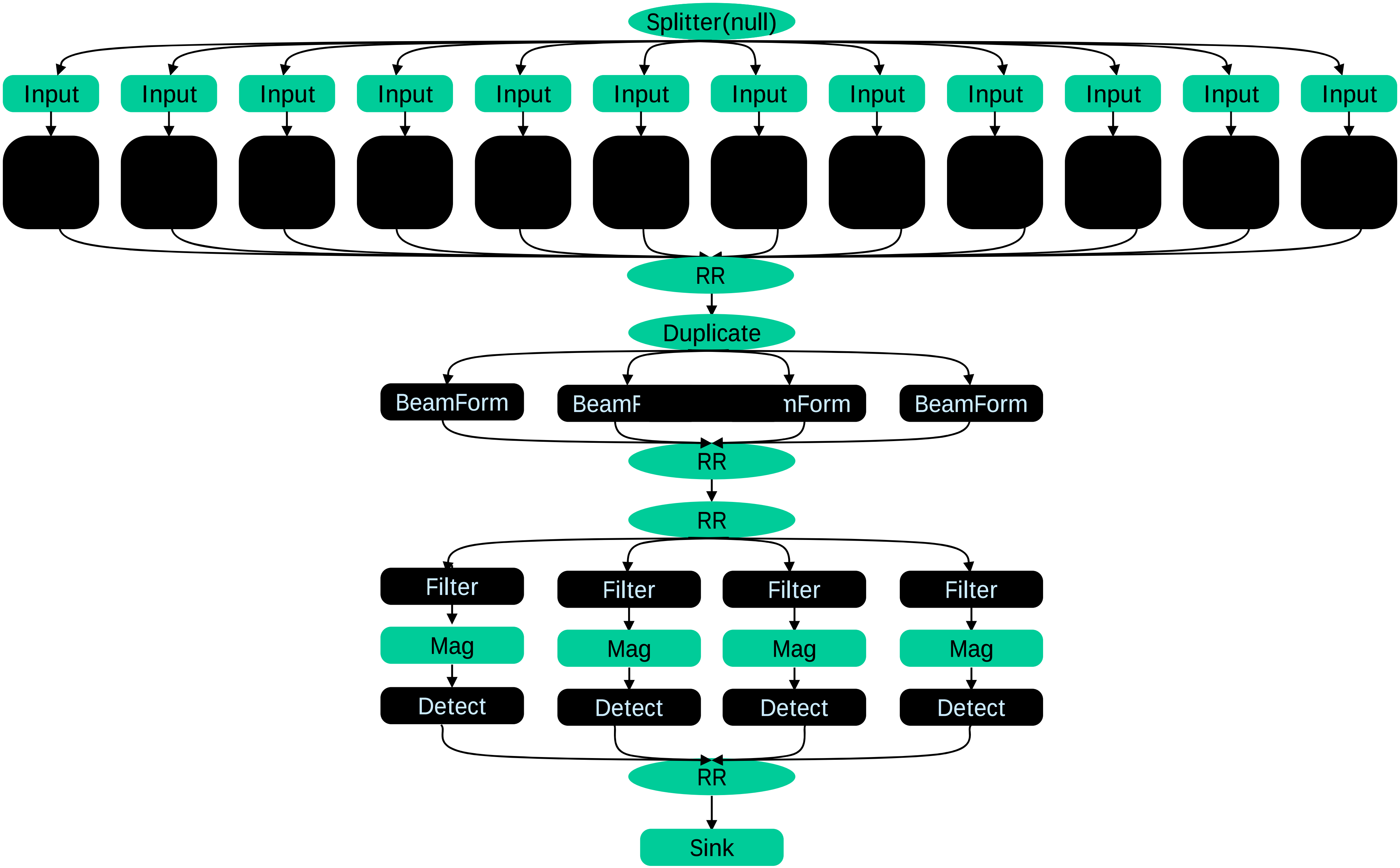
# Radar (Transformation Selection)



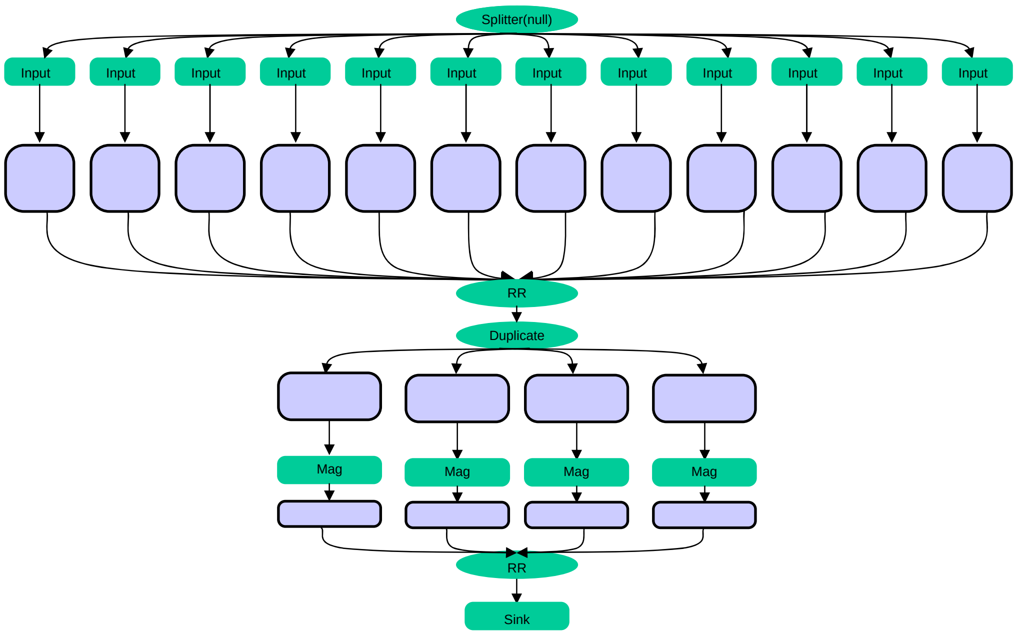
# Radar (Transformation Selection)



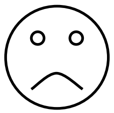
# Radar (Transformation Selection)



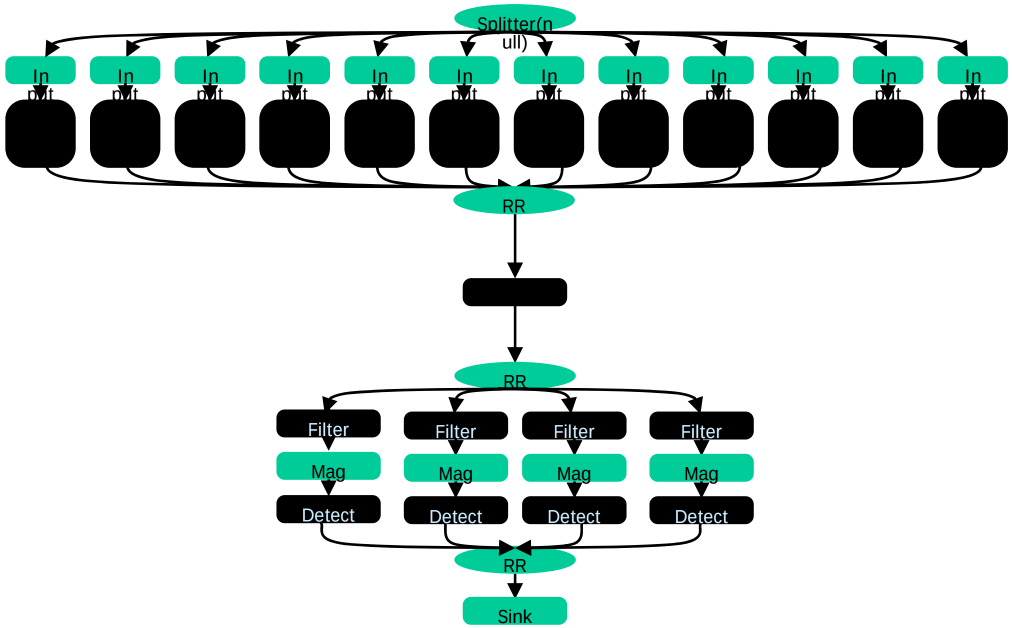
# Radar



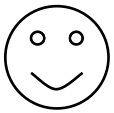

Maximal Combination and Shifting to Frequency Domain



2.4 times as many FLOPS

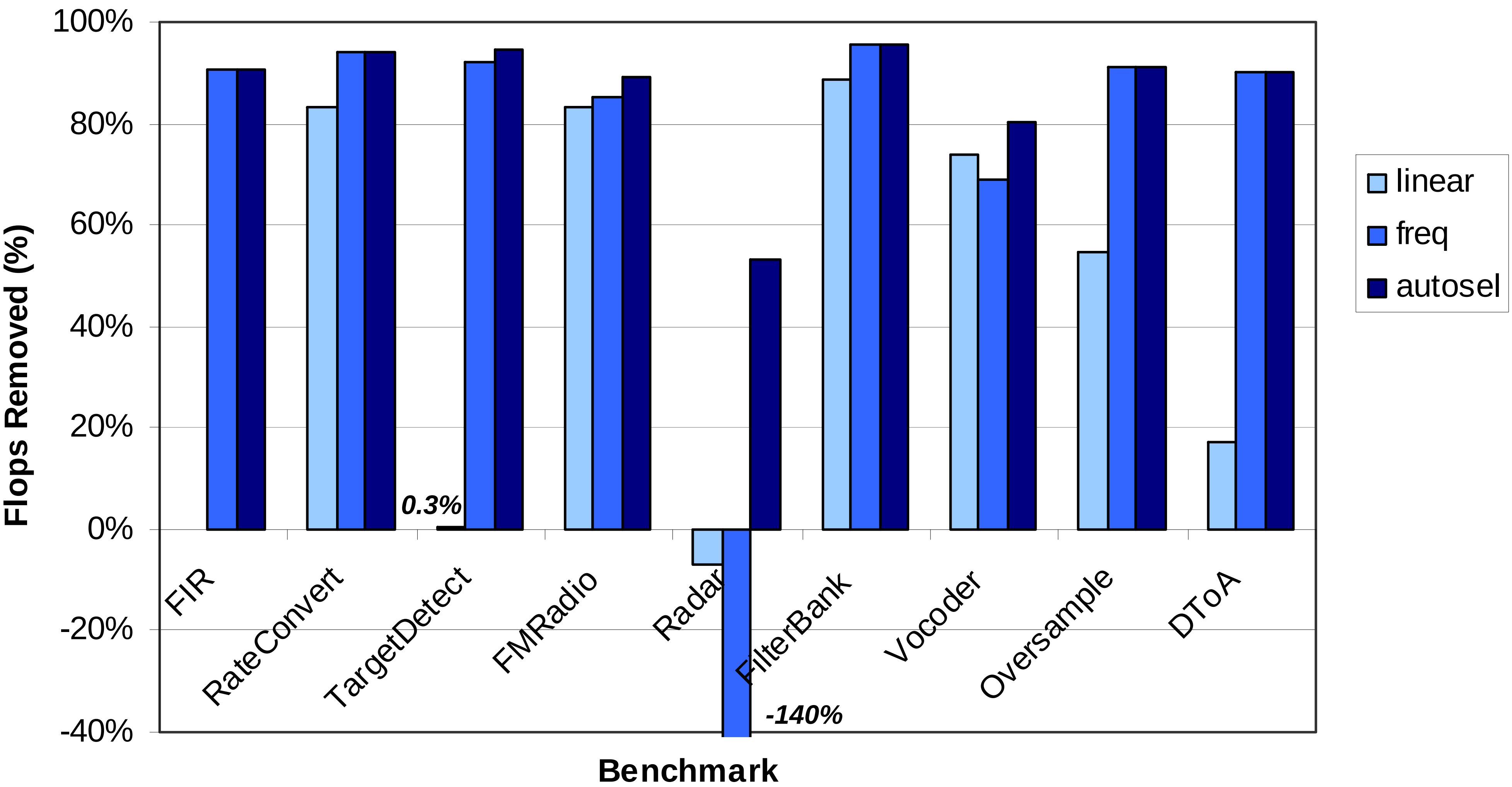



Using Transformation Selection

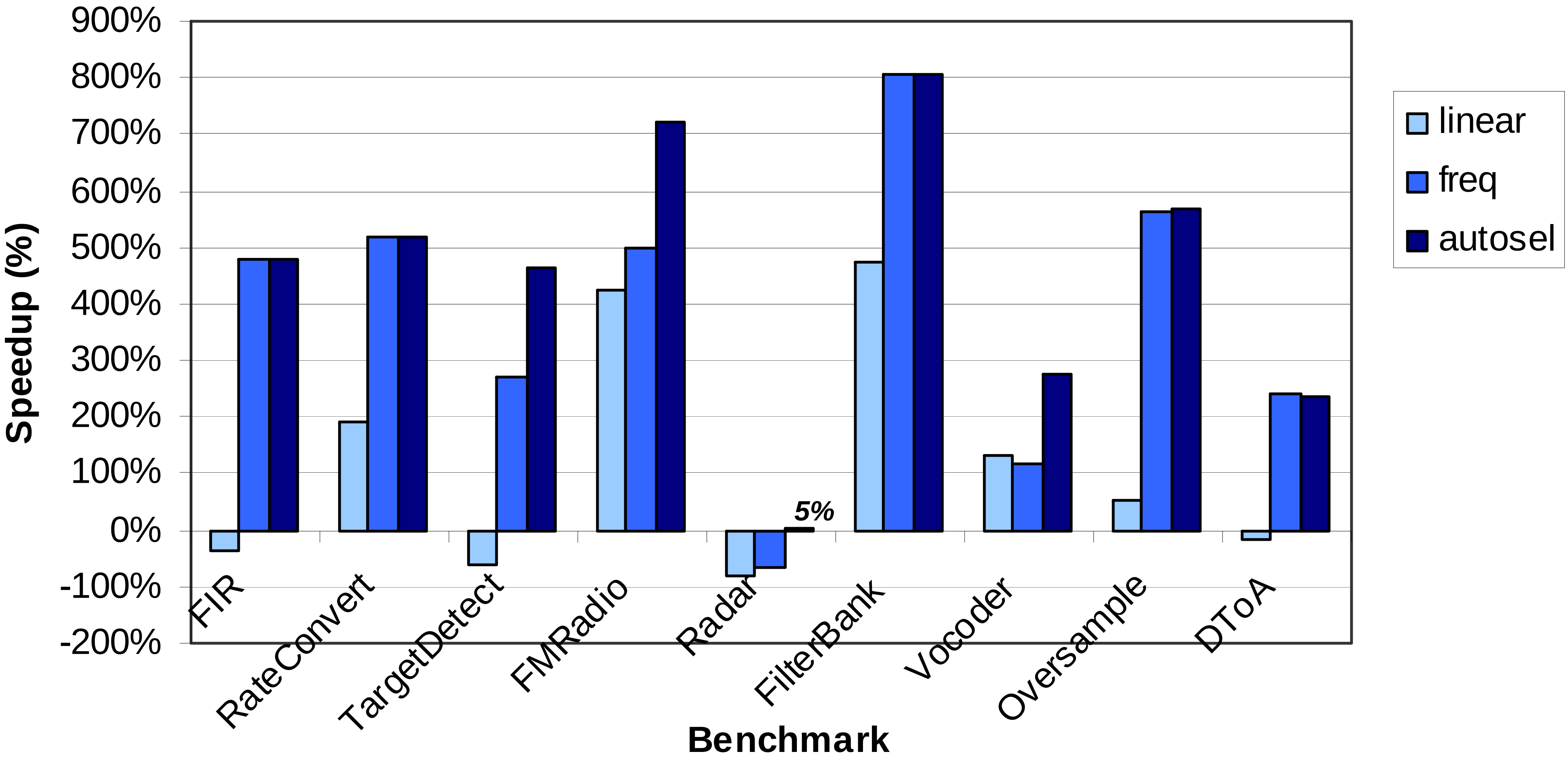


half as many FLOPS

# Floating Point Operations Reduction

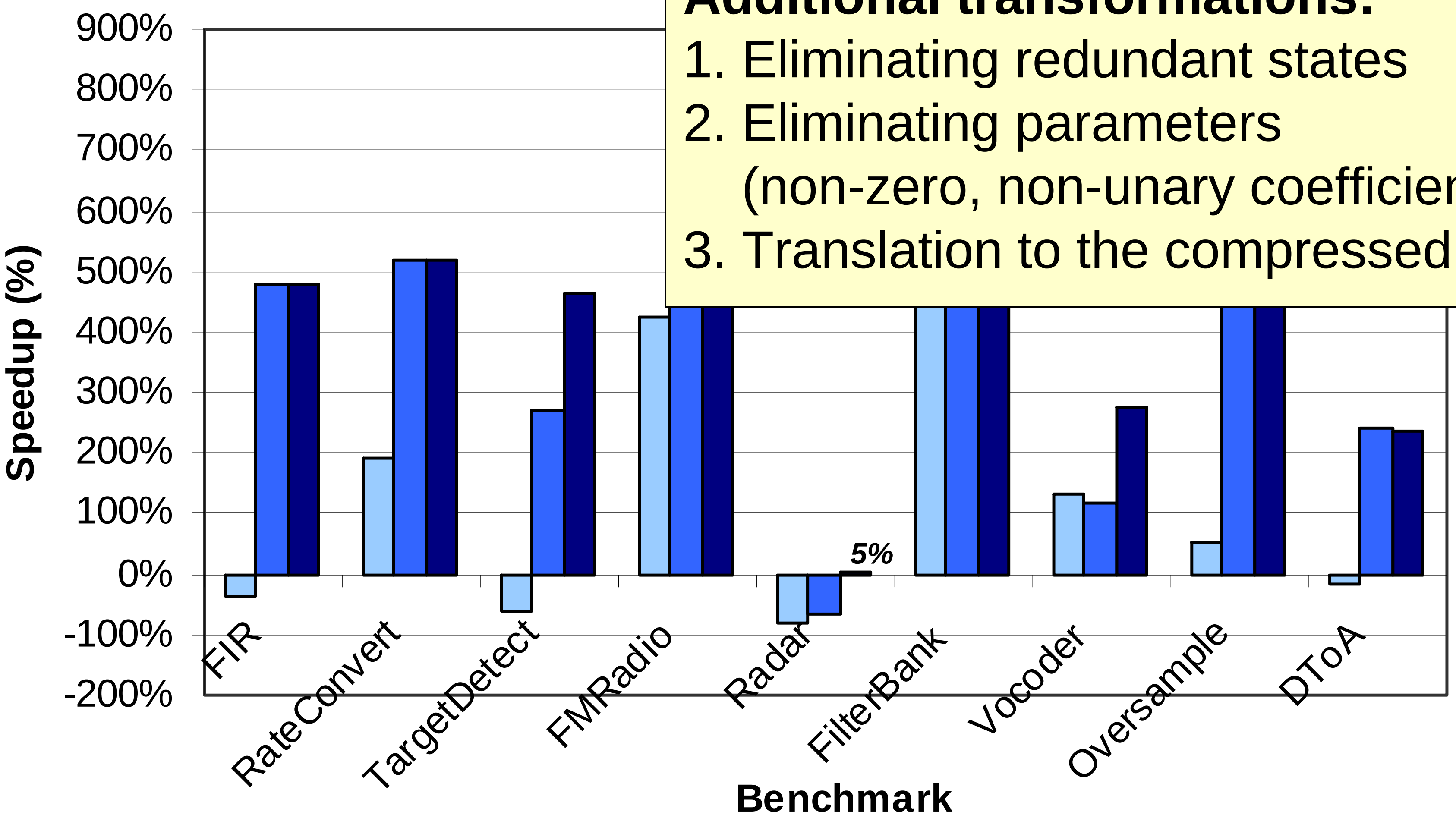


# Execution Speedup




On a Pentium IV

# Execution Speedup



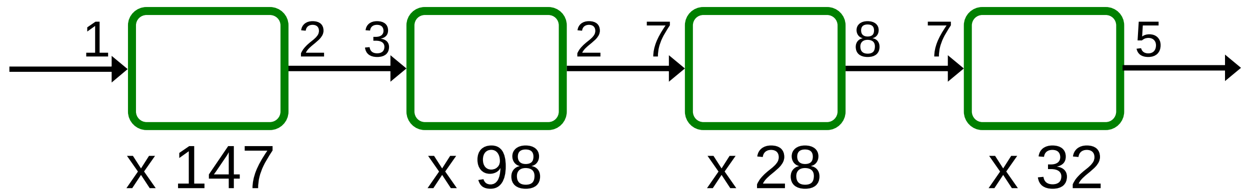

**Additional transformations:**

1. Eliminating redundant states
2. Eliminating parameters
   (non-zero, non-unary coefficients)
3. Translation to the compressed domain

On a Pentium IV

# StreamIt:  Lessons Learned

- **In practice, I/O rates of filters are often matched** [LCTES'03]
  - Over 30 publications study an uncommon case (CD-DAT)




- **Multi-phase filters complicate programs, compilers**
  - Should maintain simplicity of only one atomic step per filter

- **Programmers accidentally introduce mutable filter state**

```
void>int filter SquareWave() {
   work push 2 {
      push(0);
      push(1);
   }
}
```

*stateless*

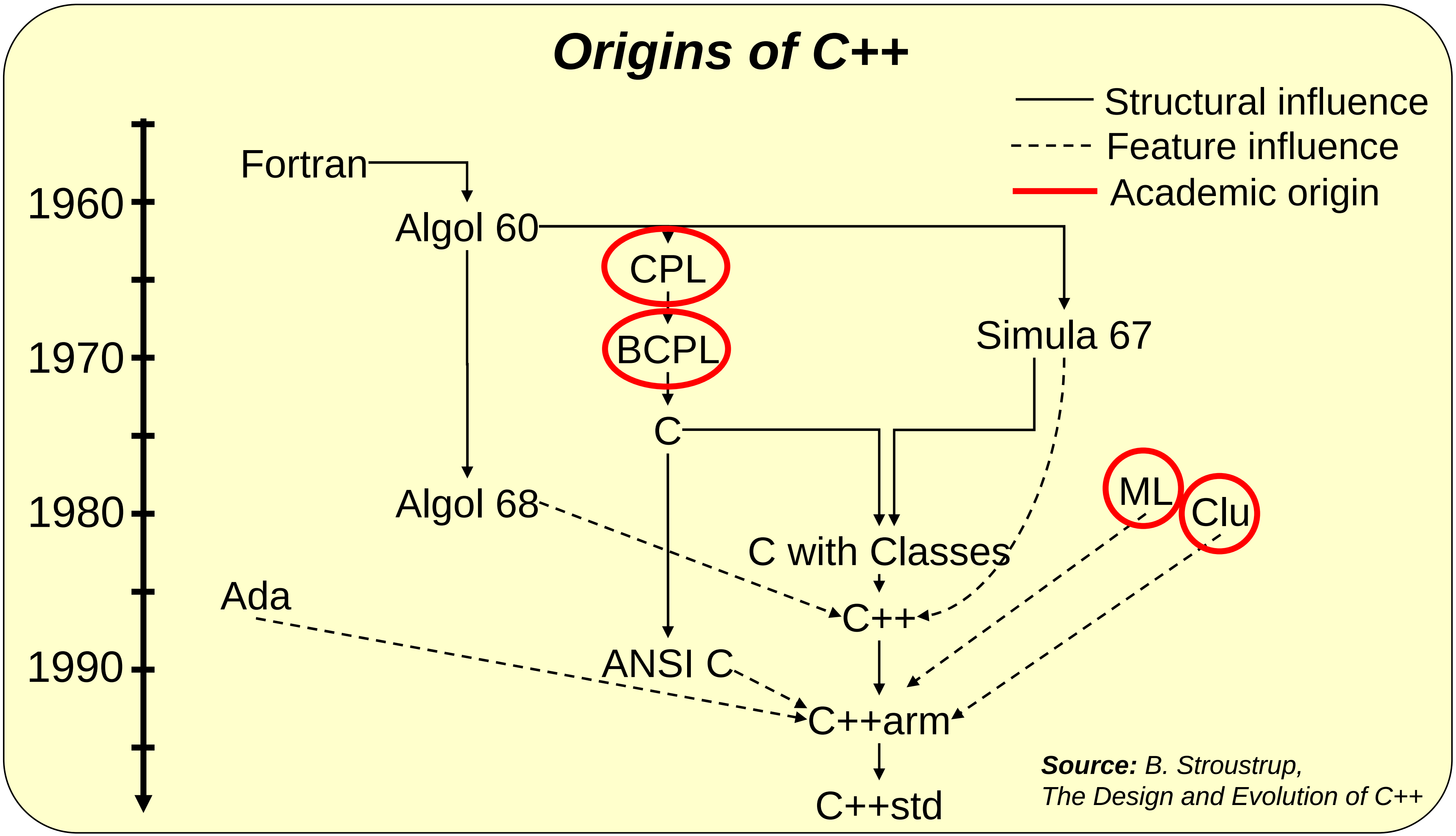
```
void>int filter SquareWave() {
   int x = 0;

   work push 1 {
      push(x);
      x = 1 - x;
} }
```

*stateful*

# Future of StreamIt

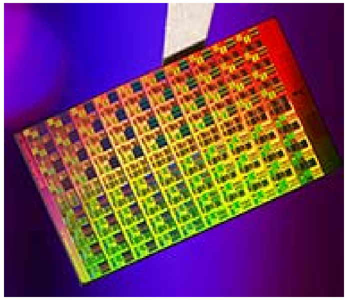
- **Goal: influence the next big language**



# Research Trajectory

- **Vision: Make emerging computational substrates universally accessible and useful**

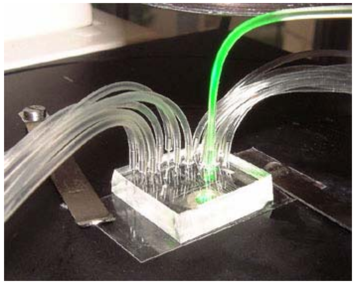
**1. Languages, compilers, & tools for multicores**



– I believe new language / compiler technology can enable scalable and robust performance

– Next inroads: expose & exploit flexibility in programs

**2. Programmable microfluidics**



– We have developed programming languages, tools, and flexible new devices for microfluidics

– Potential to revolutionize biology experimentation

**3. Technologies for the developing world**

– TEK: enable Internet experience over email account

– Audio Wiki: publish content from a low-cost phone

– uBox / uPhone: monitor & improve rural healthcare

# Conclusions

- **A parallel programming model will succeed only by luring programmers, making them do less, not more**
- **Stream programming lures programmers with:**
  - Elegant programming primitives
  - Domain-specific optimizations
- **Meanwhile, streaming is implicitly parallel**
  - Robust performance via task, data, & pipeline parallelism

***Contributions***

- Structured streams
- Teleport messaging
- Unified algorithm for task, data, pipeline parallelism
- Software pipelining of whole procedures
- Algebraic simplification of whole procedures
- Translation from time to frequency
- Selection of best DSP transforms

- **We believe stream programming will play a key role in enabling a transition to multicore processors**