

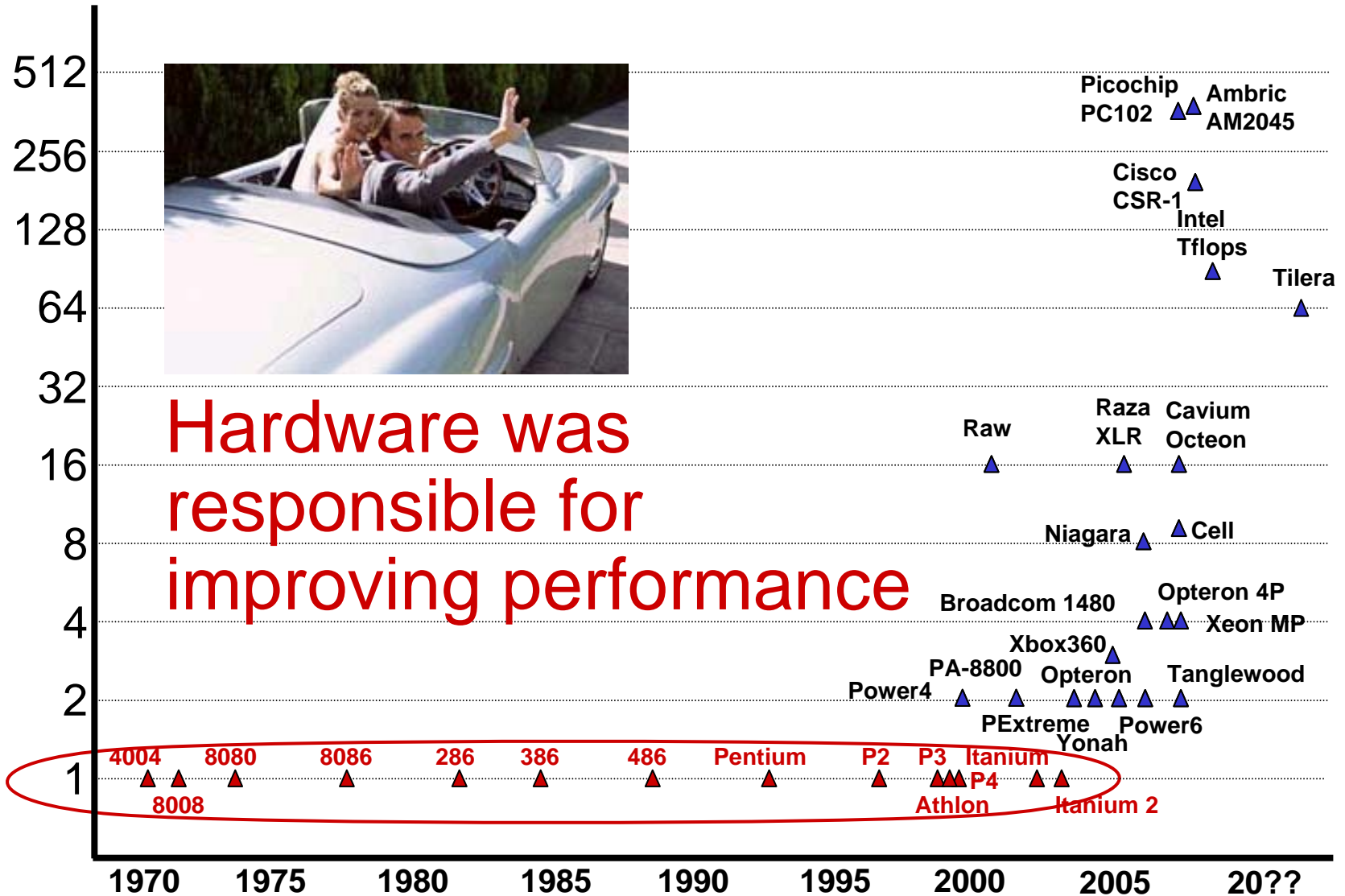
Stream Programming: Luring Programmers into the Multicore Era

Bill Thies

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Spring 2008

Multicores are Here

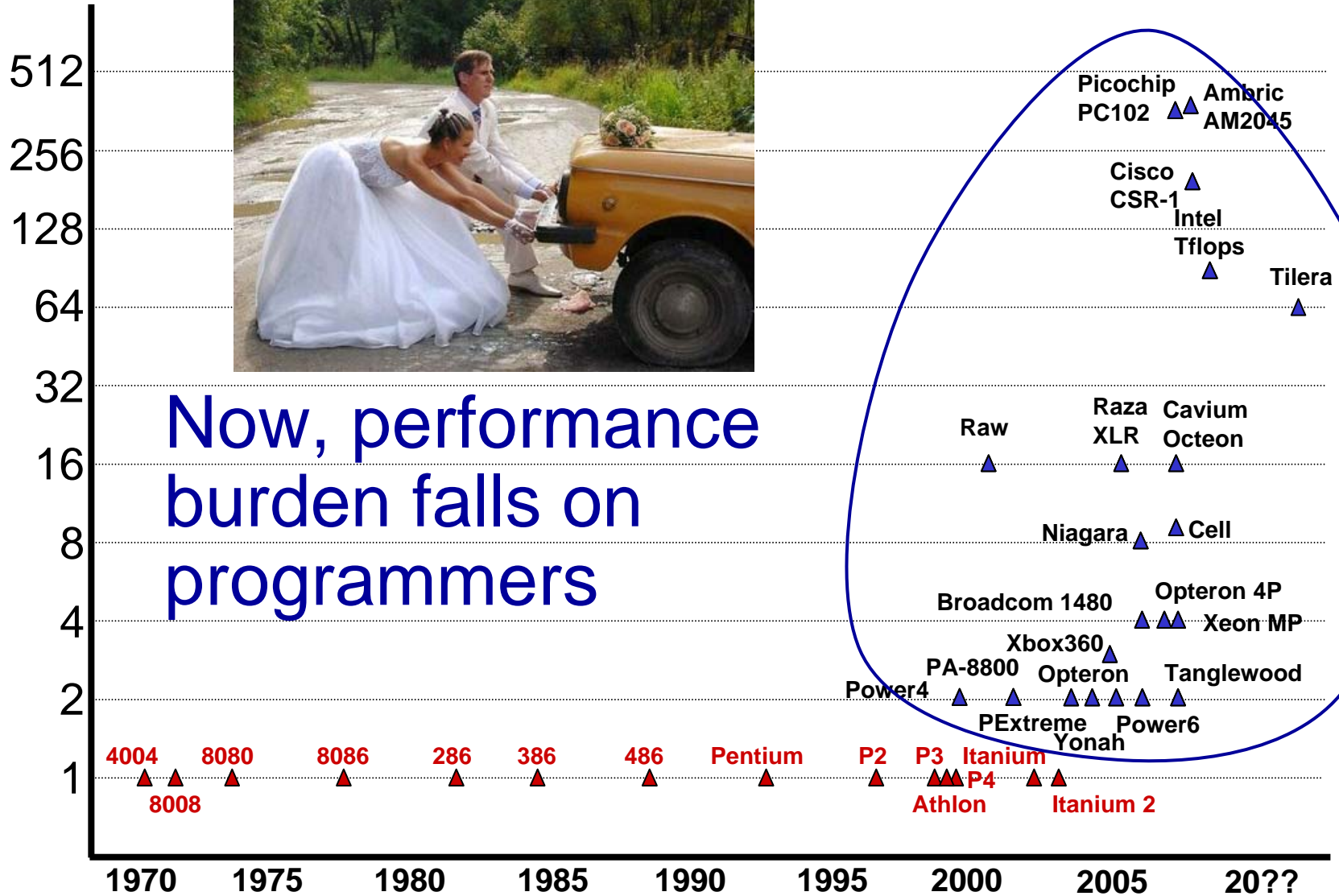


Multicores are Here



Now, performance burden falls on programmers

of cores

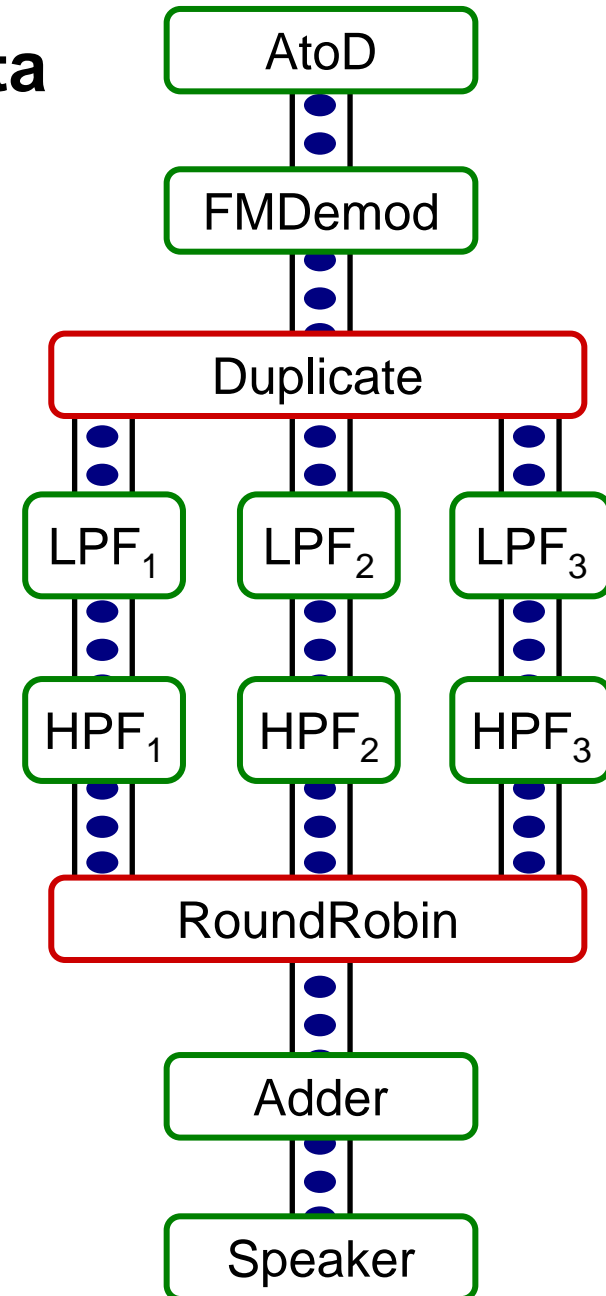


Is Parallel Programming a New Problem?

- **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

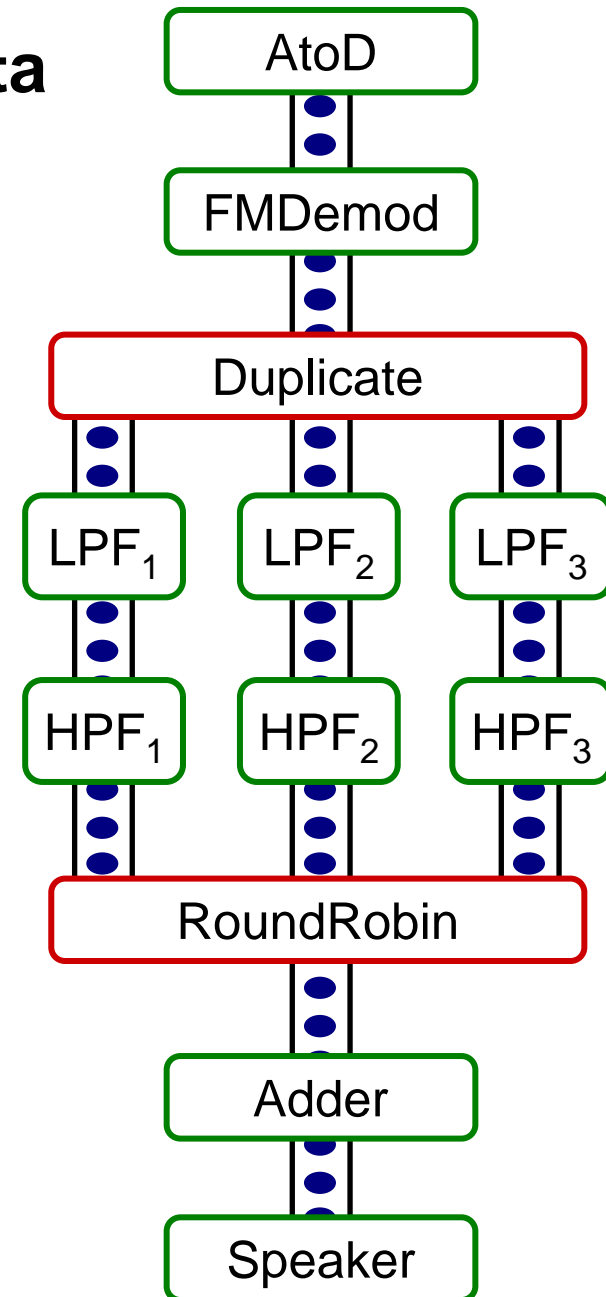
Streaming Application Domain

- **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

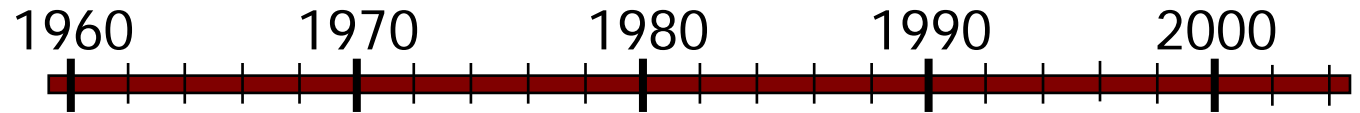


Streaming Application Domain

- **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



Models of Computation

Petri Nets
Comp. Graphs

Kahn Proc. Networks
Communicating Sequential Processes

Synchronous Dataflow

Modeling Environments

Ptolemy
Gabriel

Matlab/Simulink
Grape-II

etc.

Languages / Compilers

Lucid
C

Id
lazy

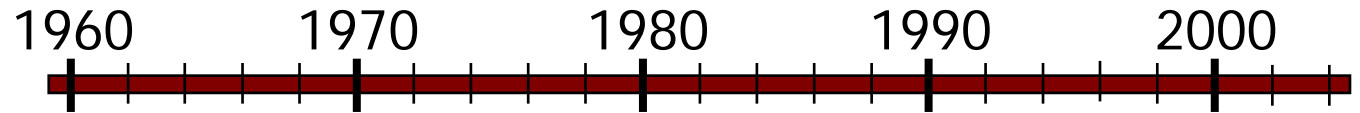
Sisal
VAL

Erlang
Occam

Esterel
LUSTRE

pH

Brief History of Streaming



Models of Computation

Petri Nets
Comp. Graphs

Kahn Proc. Networks
Communicating Sequential Processes

Synchronous Dataflow

Modeling Environments

Ptolemy
Gabriel

Matlab/Simulink
Grape-II

etc.

Languages / Compilers

Lucid
C

Id
lazy

Sisal
VAL

Erlang
Occam

Esterel
LUSTRE

pH

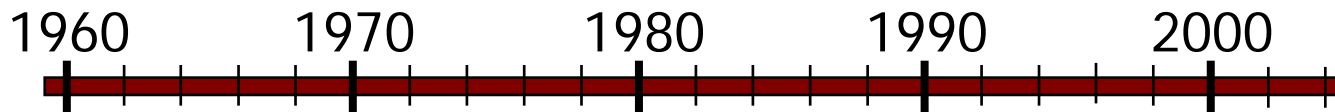
Strengths

- Elegance
- Generality

Weaknesses

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

Brief History of Streaming



Models of Computation	Petri Nets Comp. Graphs	Kahn Proc. Communicating	Networks Synchronous Dataflow Sequential Processes
Modeling Environments	Ptolemy Gabriel Matlab/Simulink Grape-II etc.		
Languages / Compilers Strengths <ul style="list-style-type: none"> Elegance Generality 	Weaknesses <ul style="list-style-type: none"> Unsuitable for static analysis Cannot leverage deep results from DSP / modeling community 		Lucid C Id lazy Sisal VAL Erlang Occam Esterel LUSTRE pH StreamIt Cg Brook StreamC <i>"Stream Programming"</i>

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

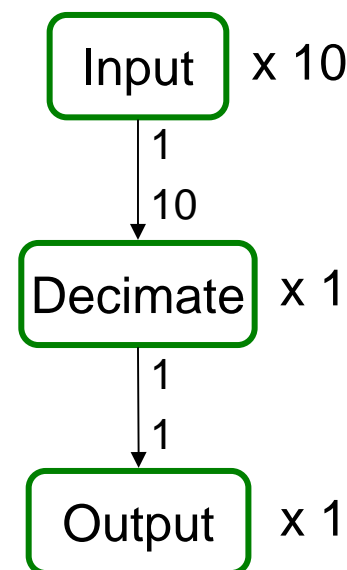
- **Model of computation: synchronous dataflow**

- 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**

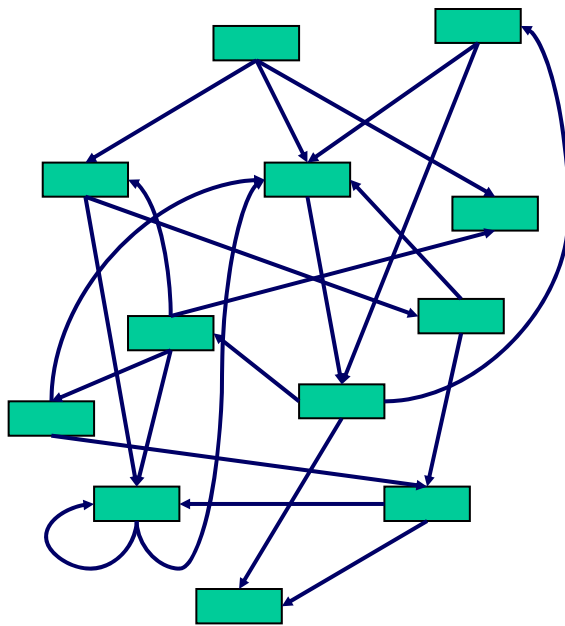
- Dynamic I/O rates
- Support for sliding window operations
- Teleport messaging [PPoPP'05]

[Lee & Messerschmidt, 1987]

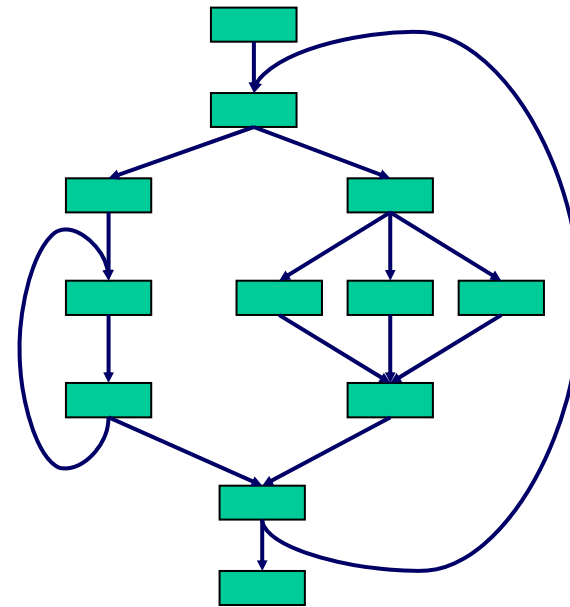


Representing Streams

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

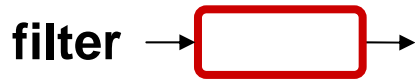


unstructured

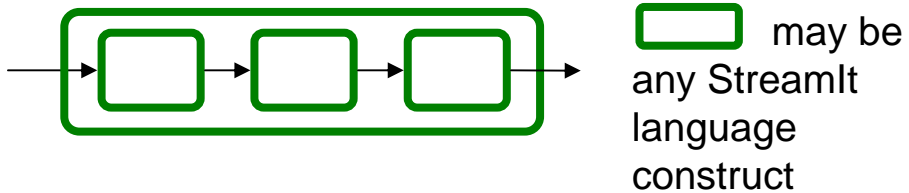


structured

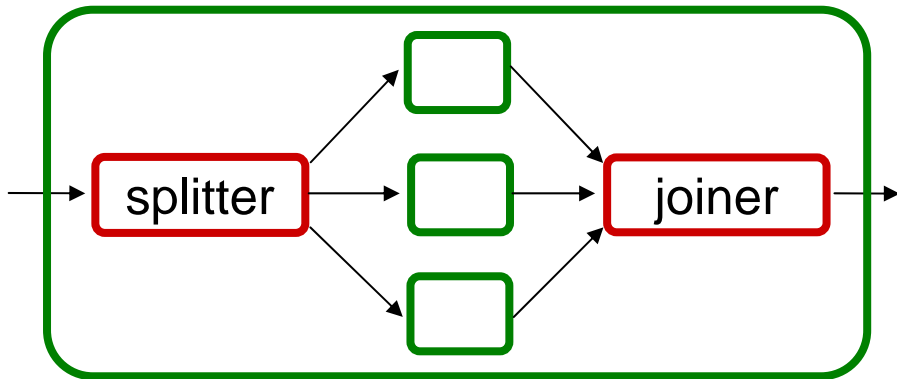
Structured Streams



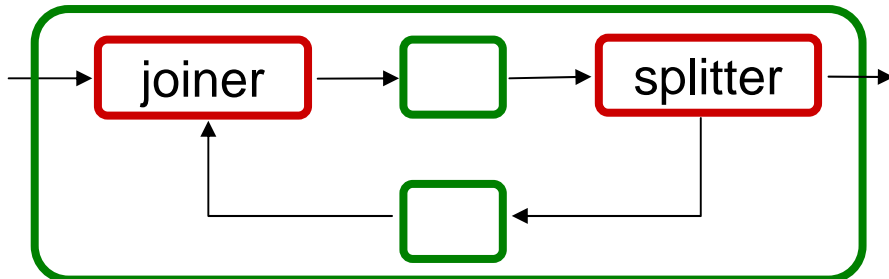
pipeline



splitjoin

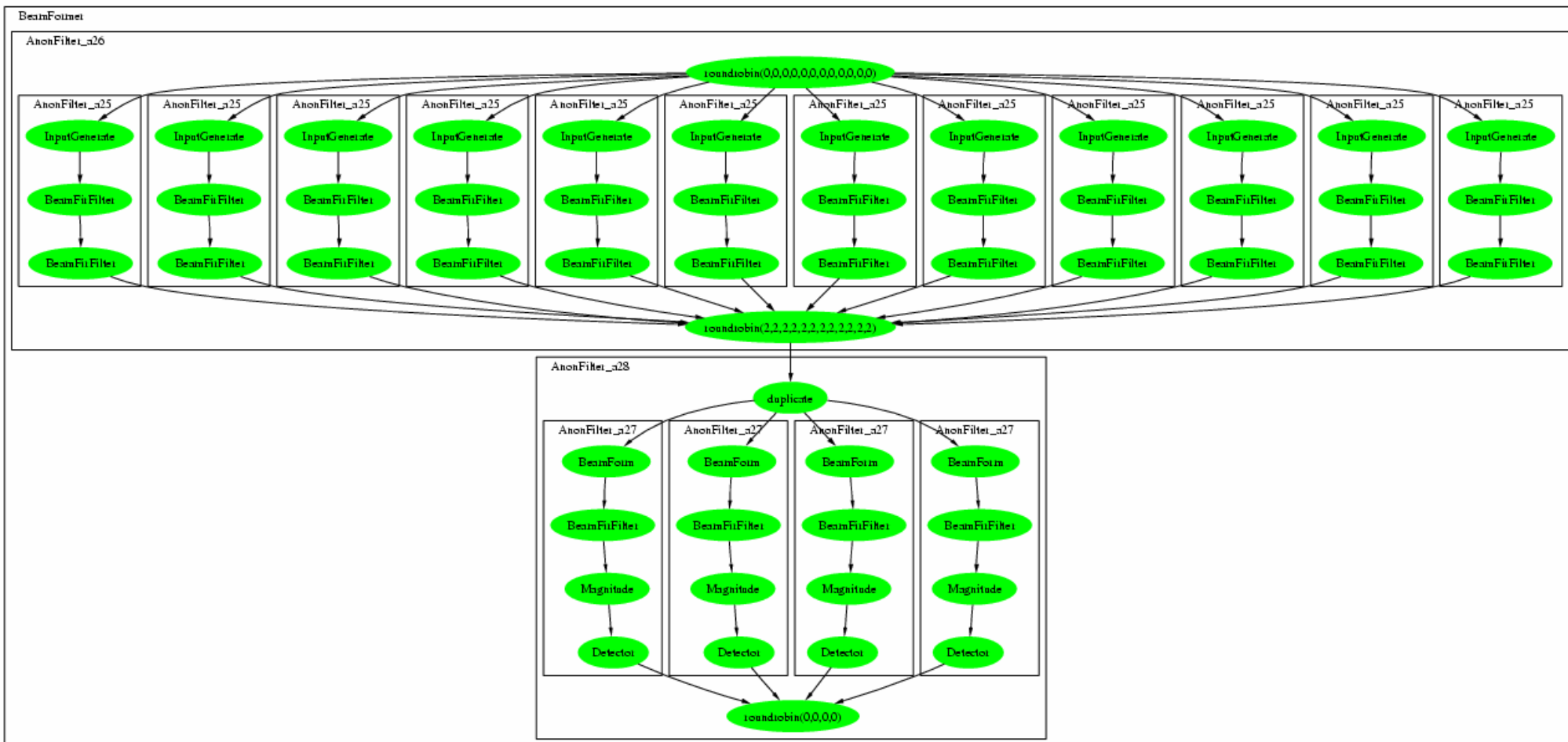


feedback loop

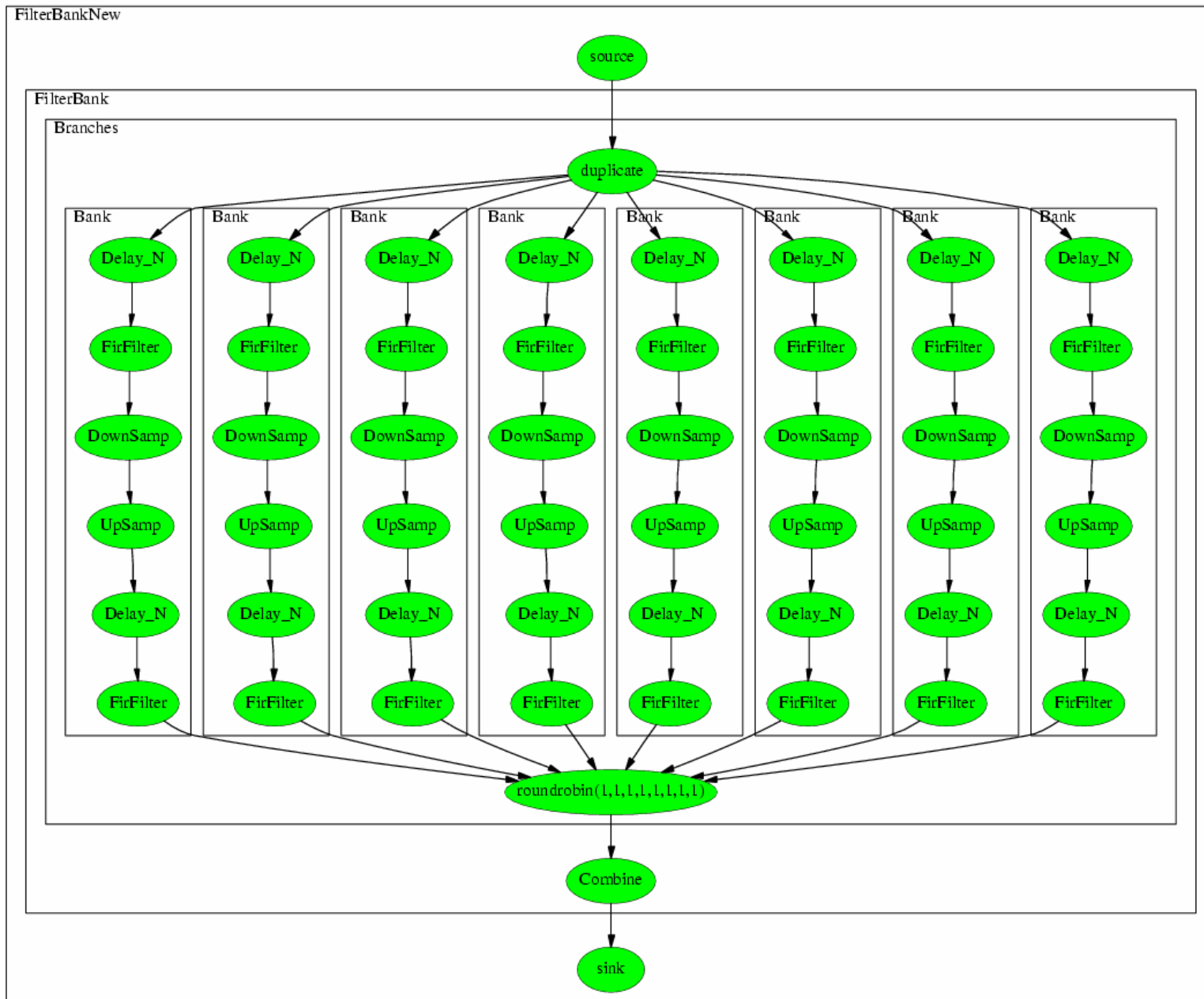


- Each structure is single-input, single-output
- Hierarchical and composable

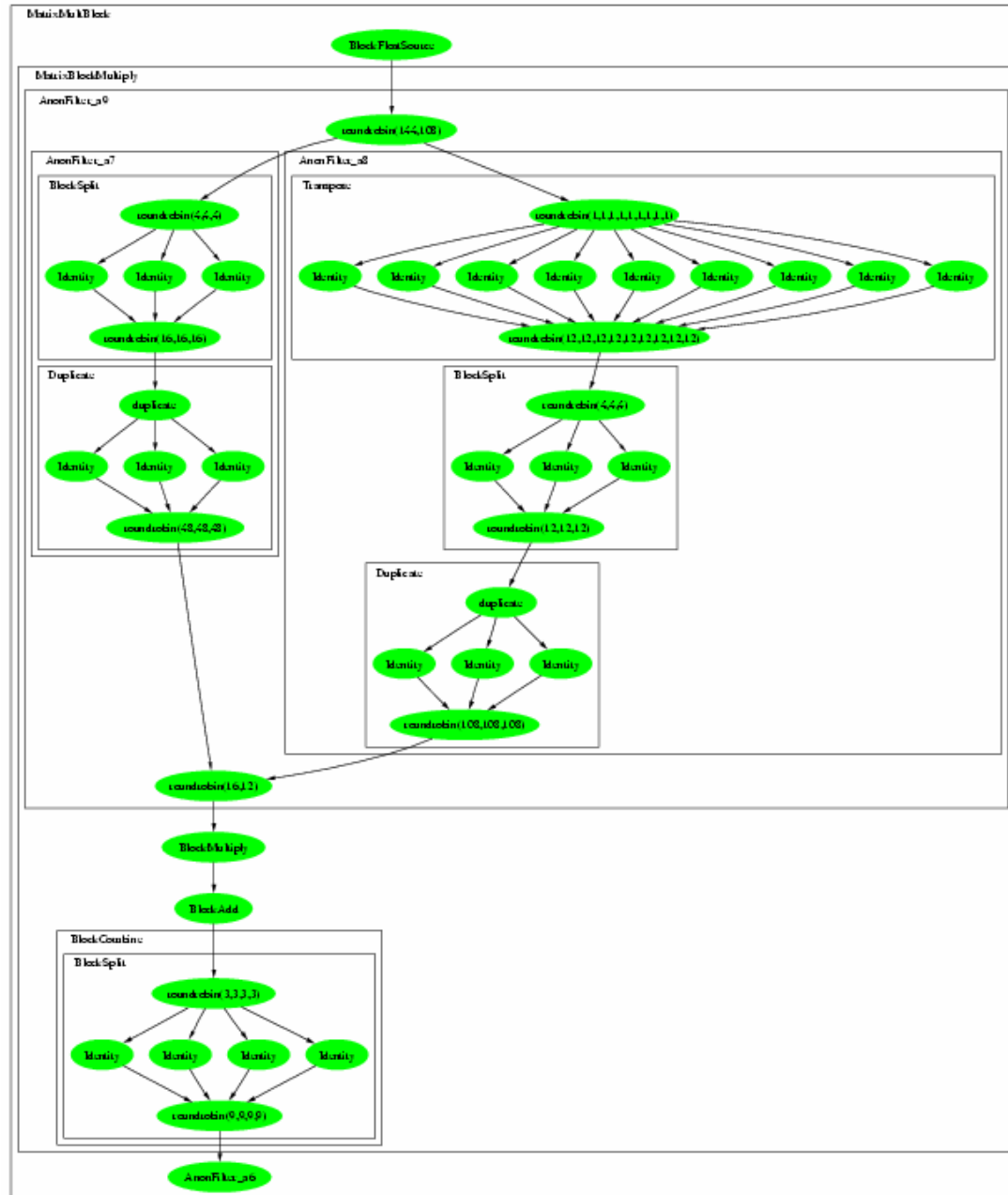
Radar-Array Front End



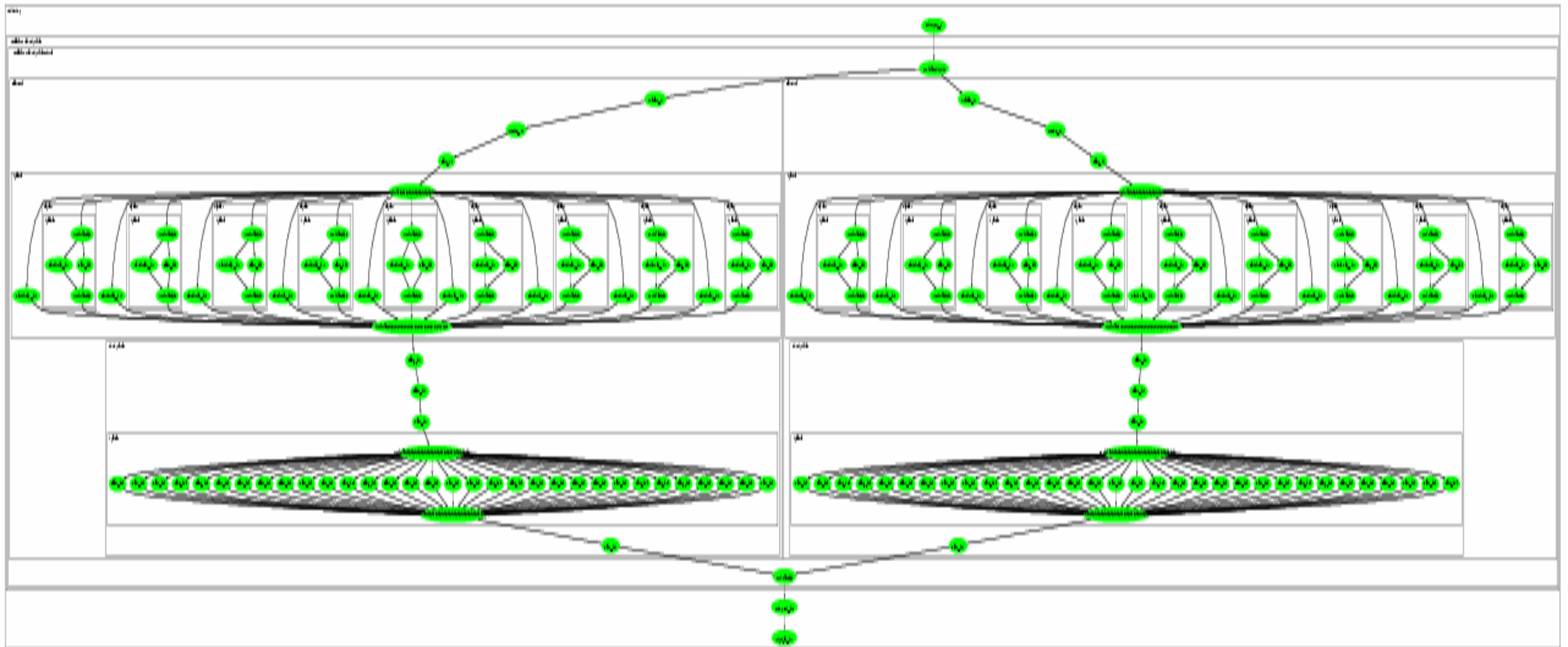
Filterbank



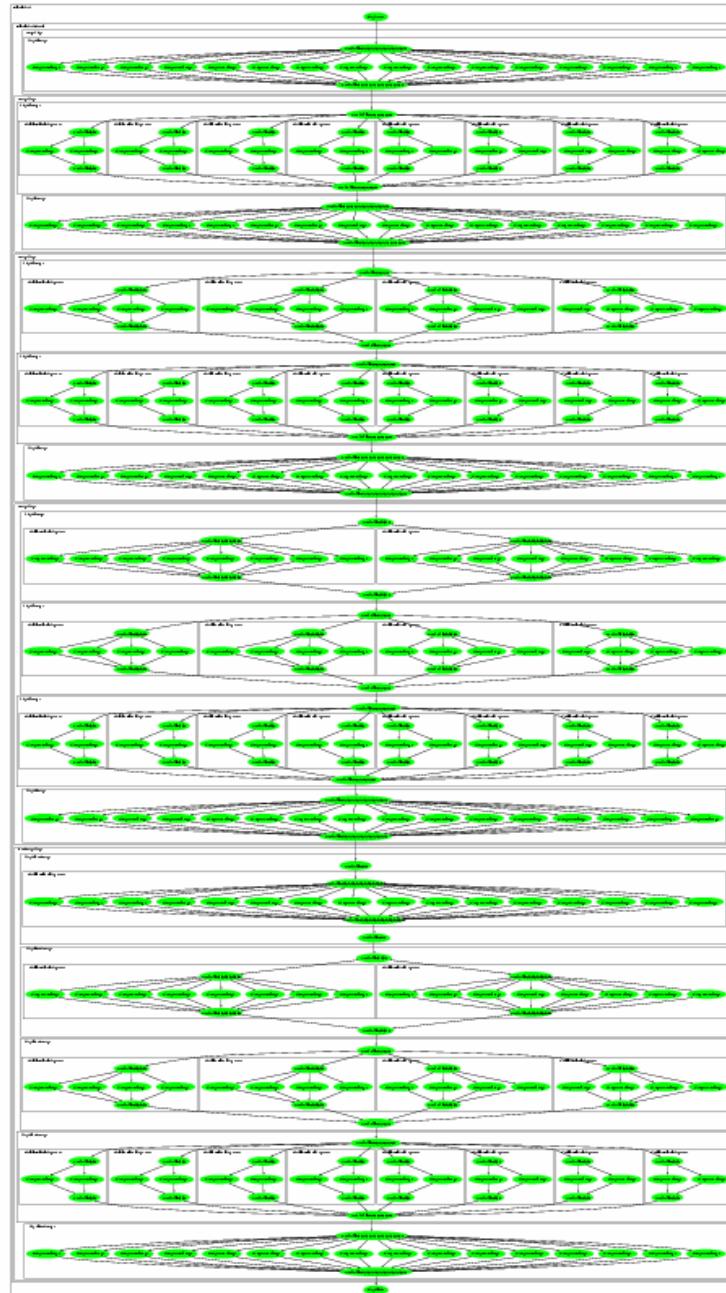
Block Matrix Multiply



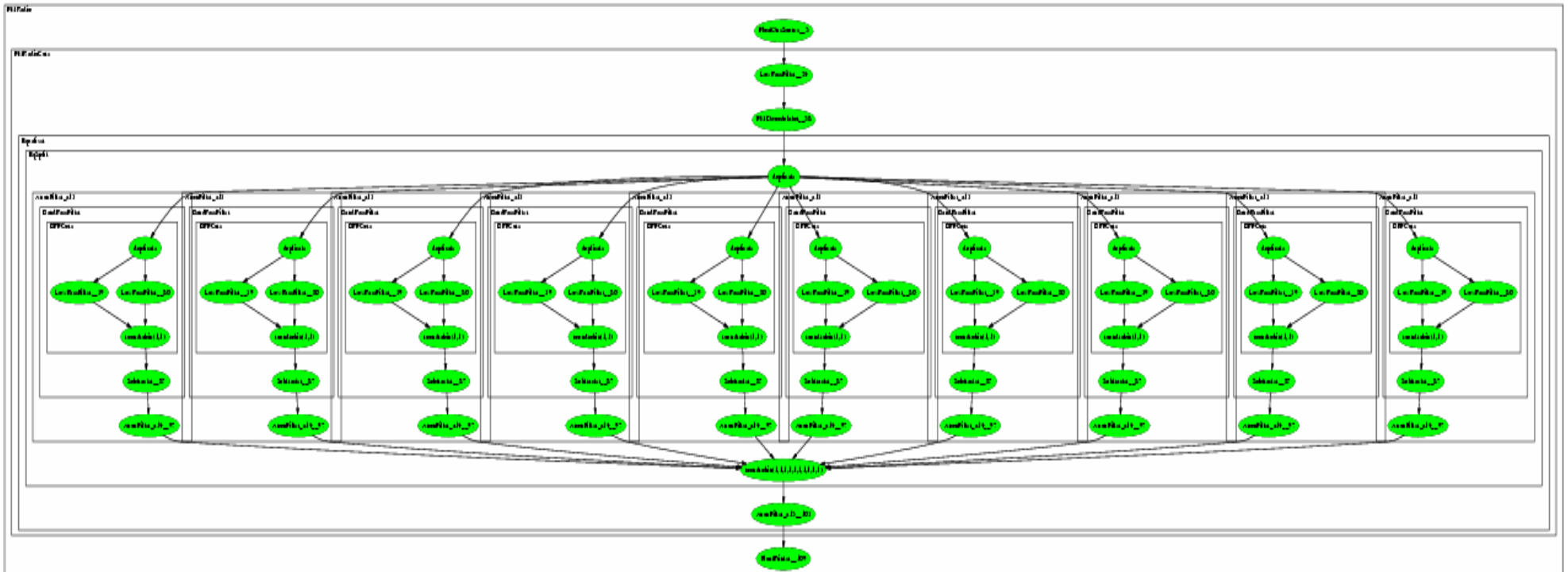
MP3 Decoder



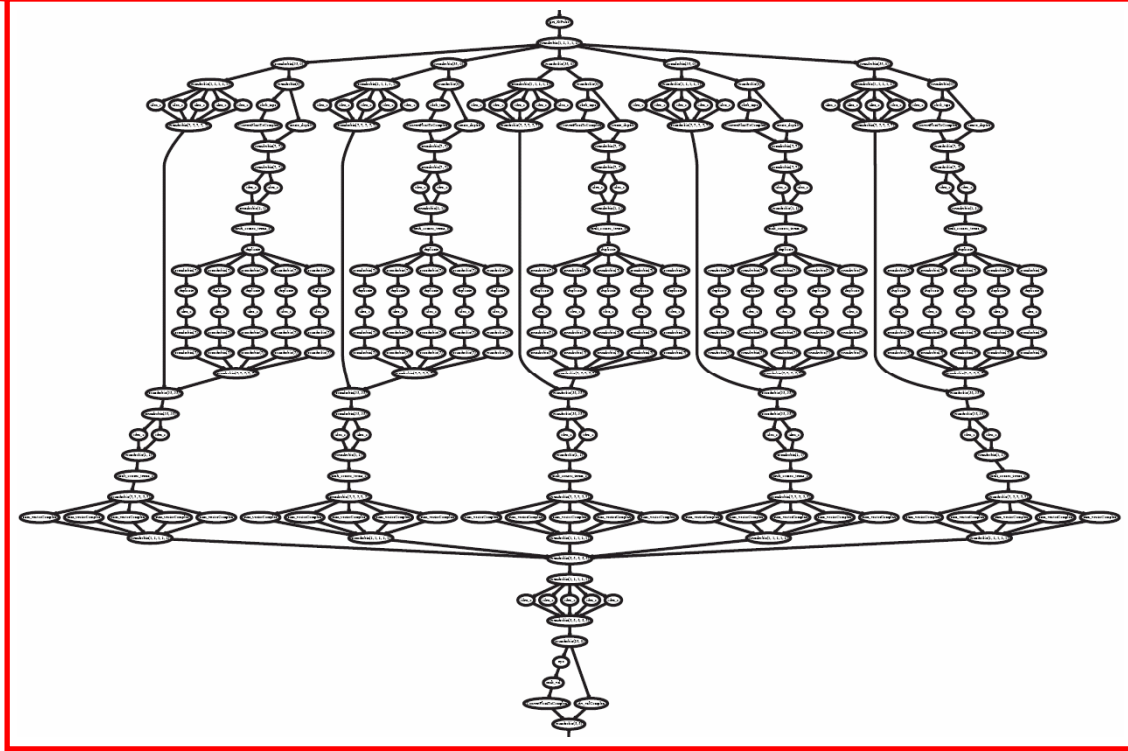
Bitonic Sort



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 {
```

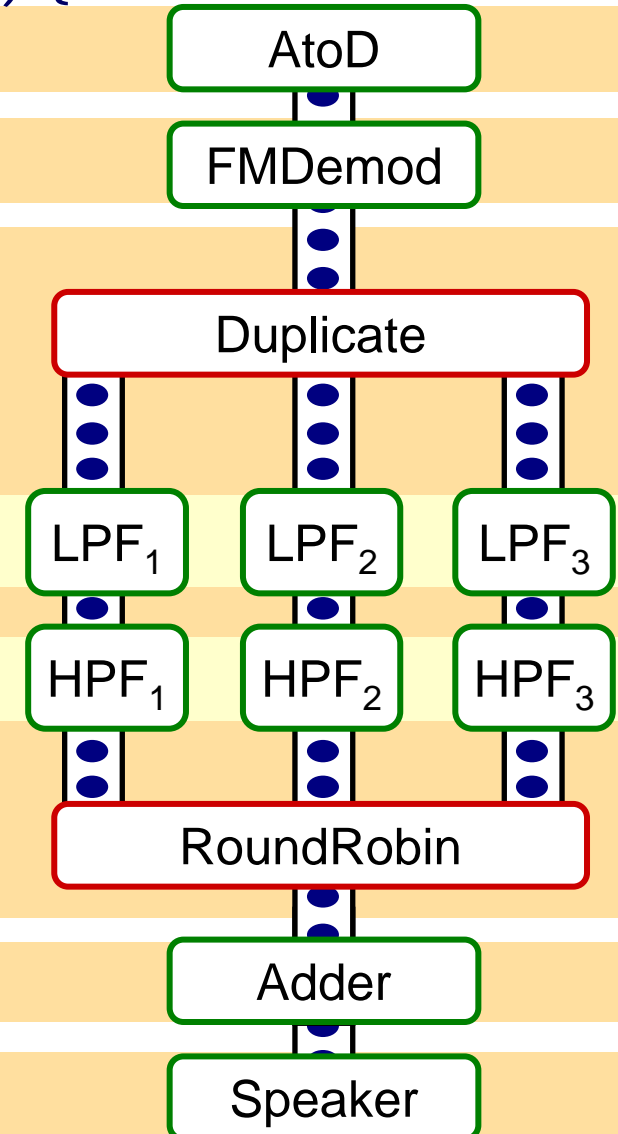
```
        add LowPassFilter(lo + i*(hi - lo)/N);
```

```
        add HighPassFilter(lo + i*(hi - lo)/N);
```

```
      }  
    }  
  }  
  join roundrobin();
```

```
  add Adder();
```

```
  add Speaker();  
}
```



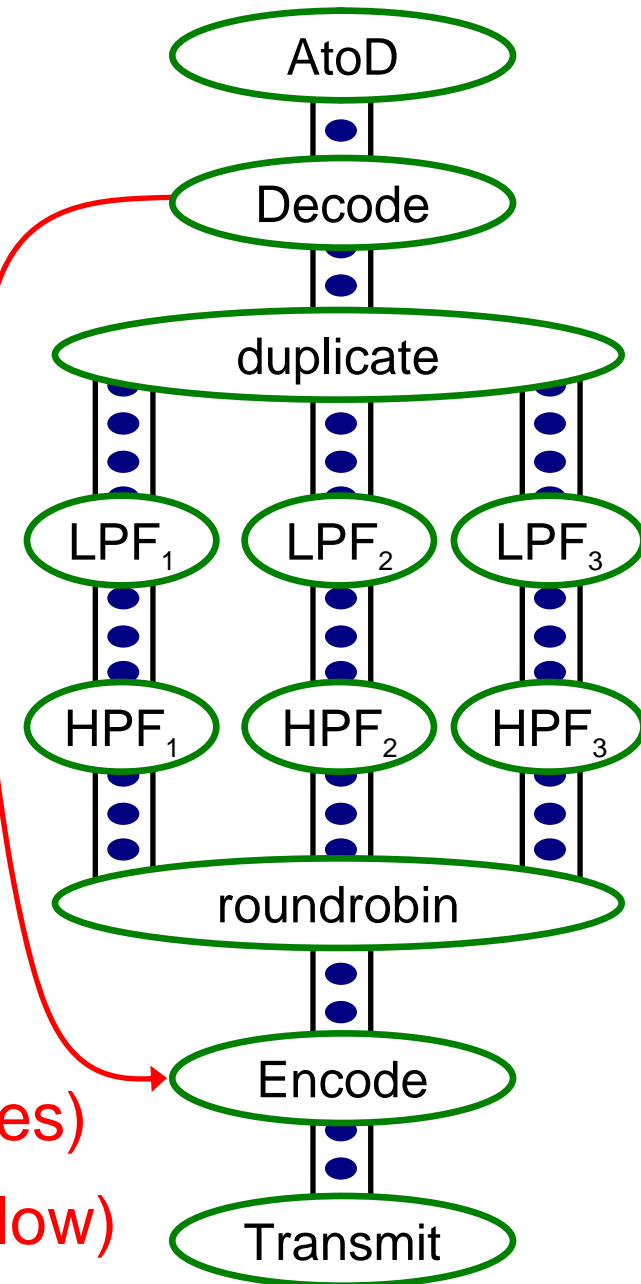
StreamIt Application Suite

- 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)



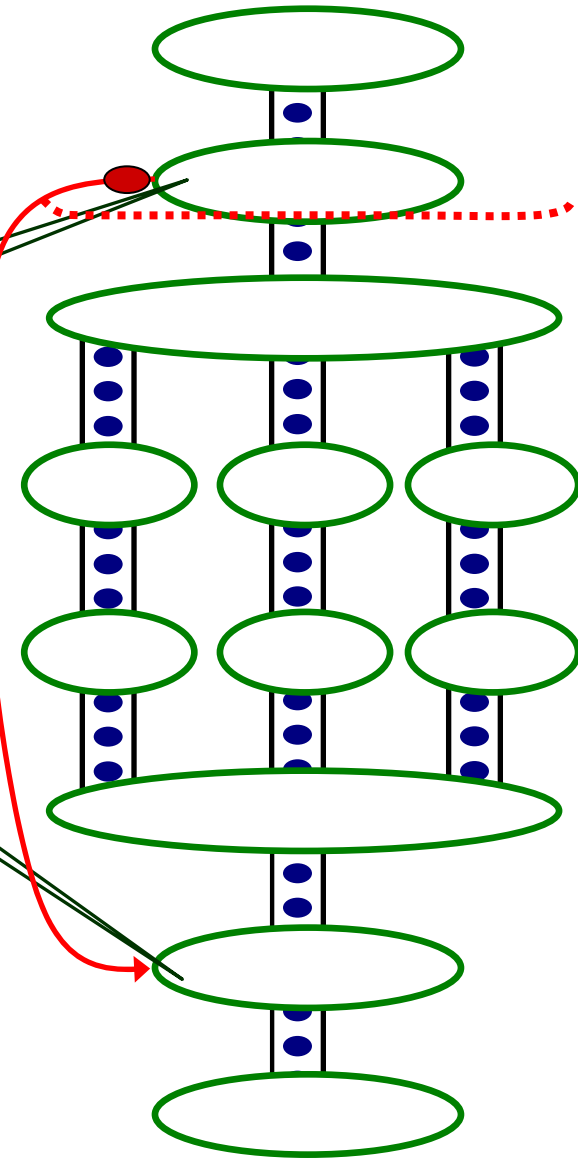
Idea 2: Teleport Messaging

- 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

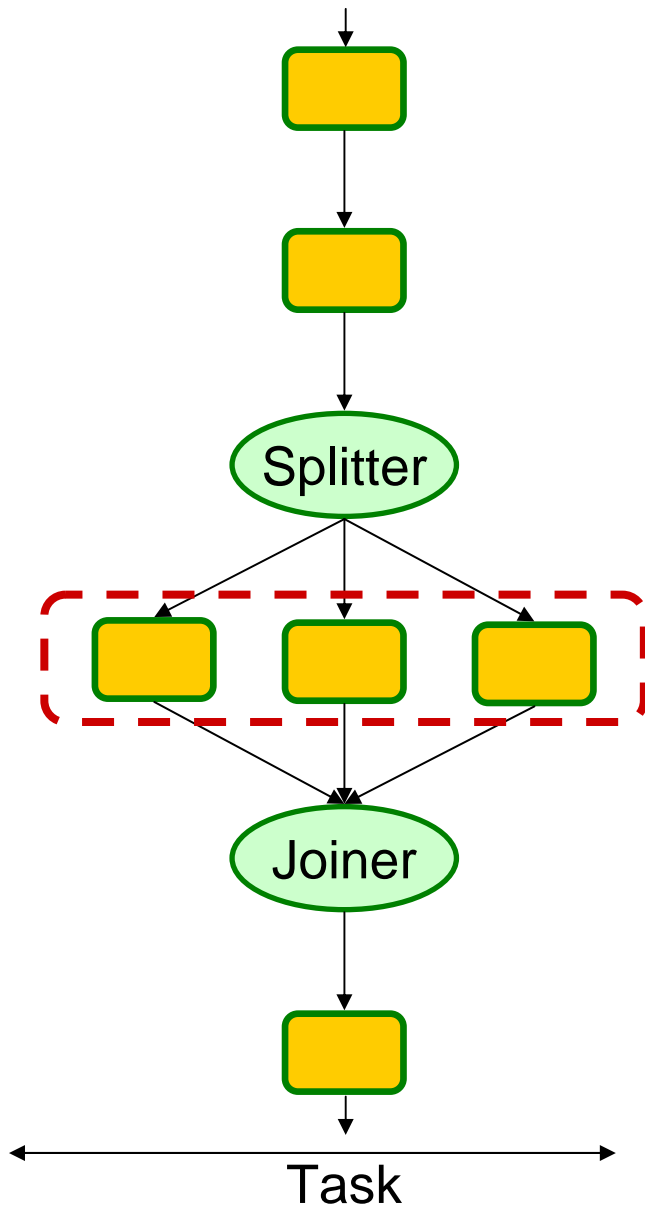
Michael I. Gordon, William Thies, Saman Amarasinghe (ASPLOS'06)

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**

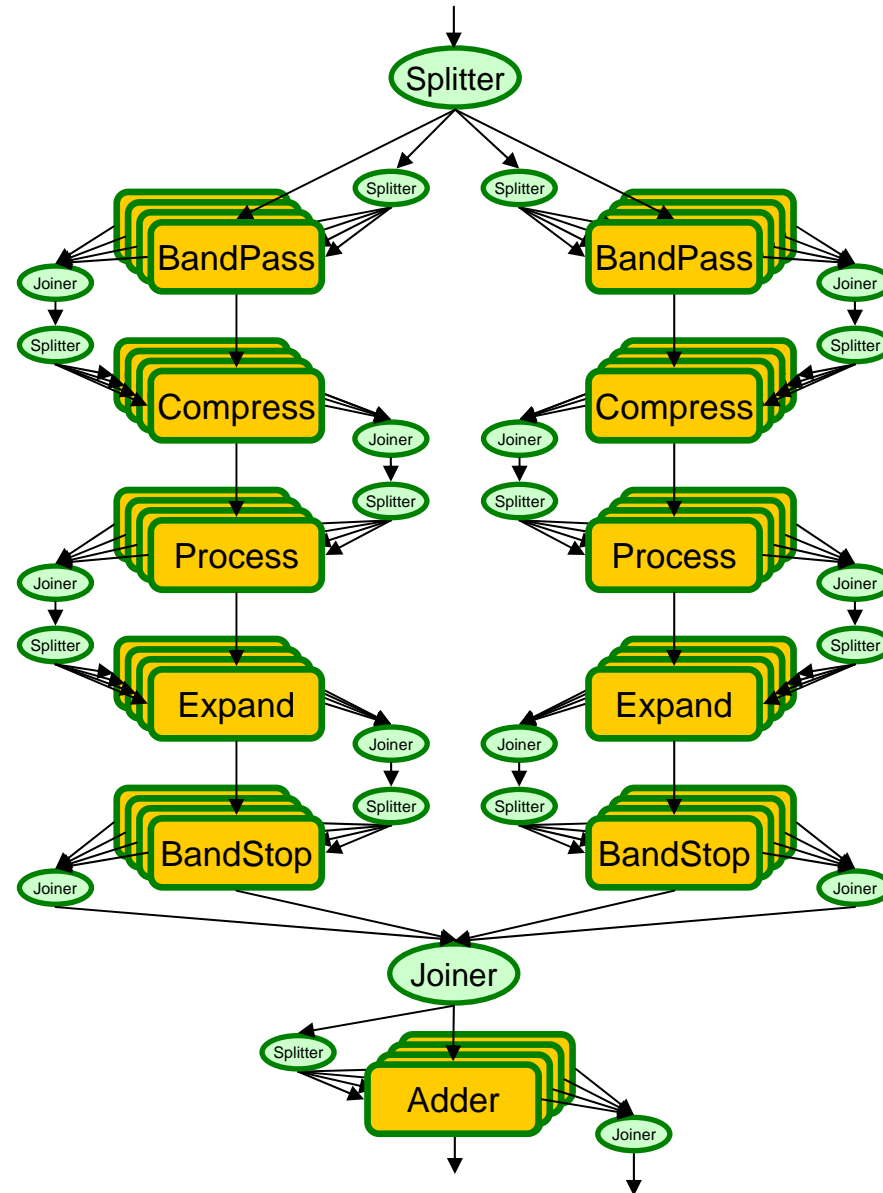
Parallelism in Stream Programs



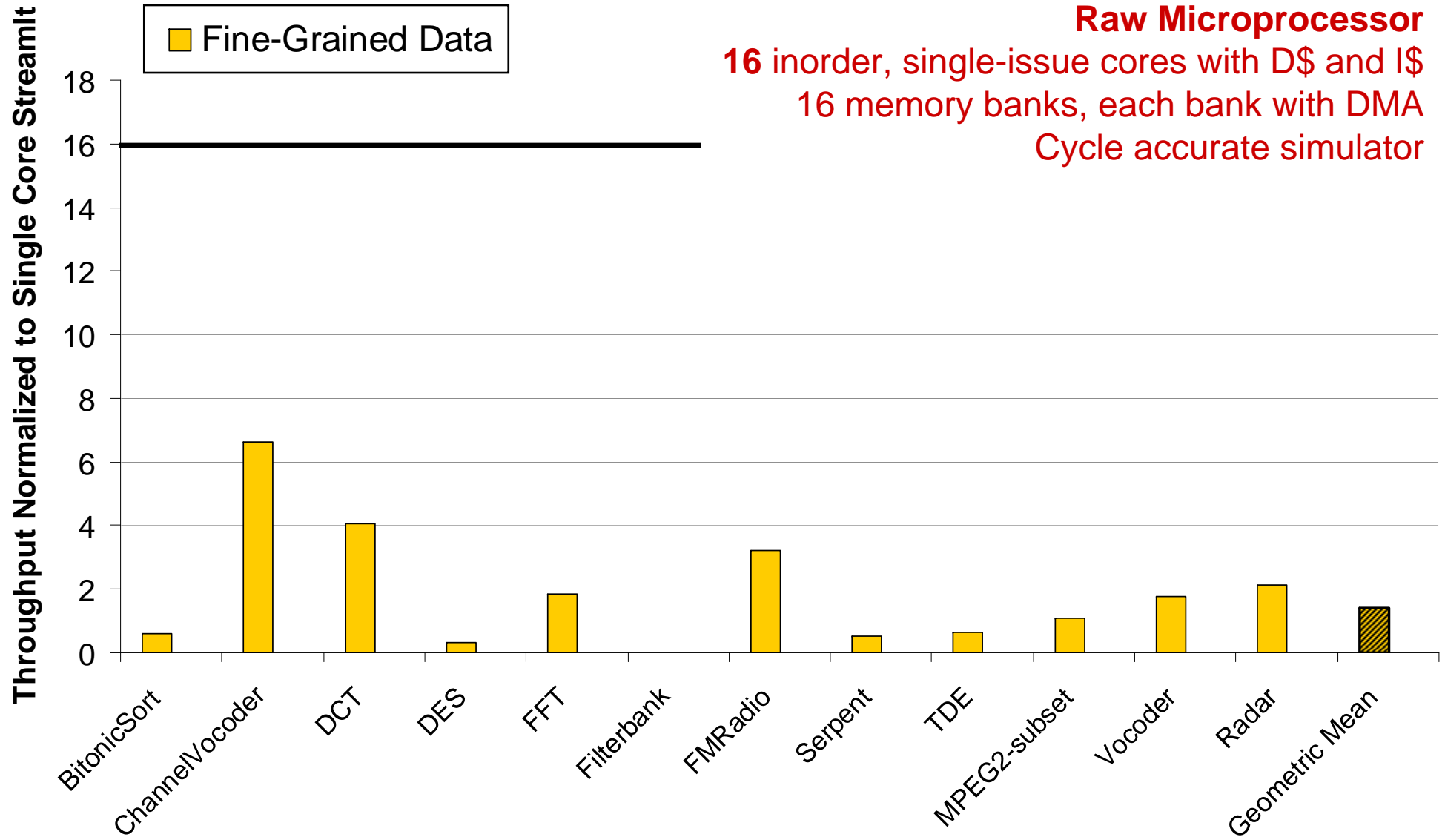
Task parallelism

- Analogous to thread (fork/join) parallelism

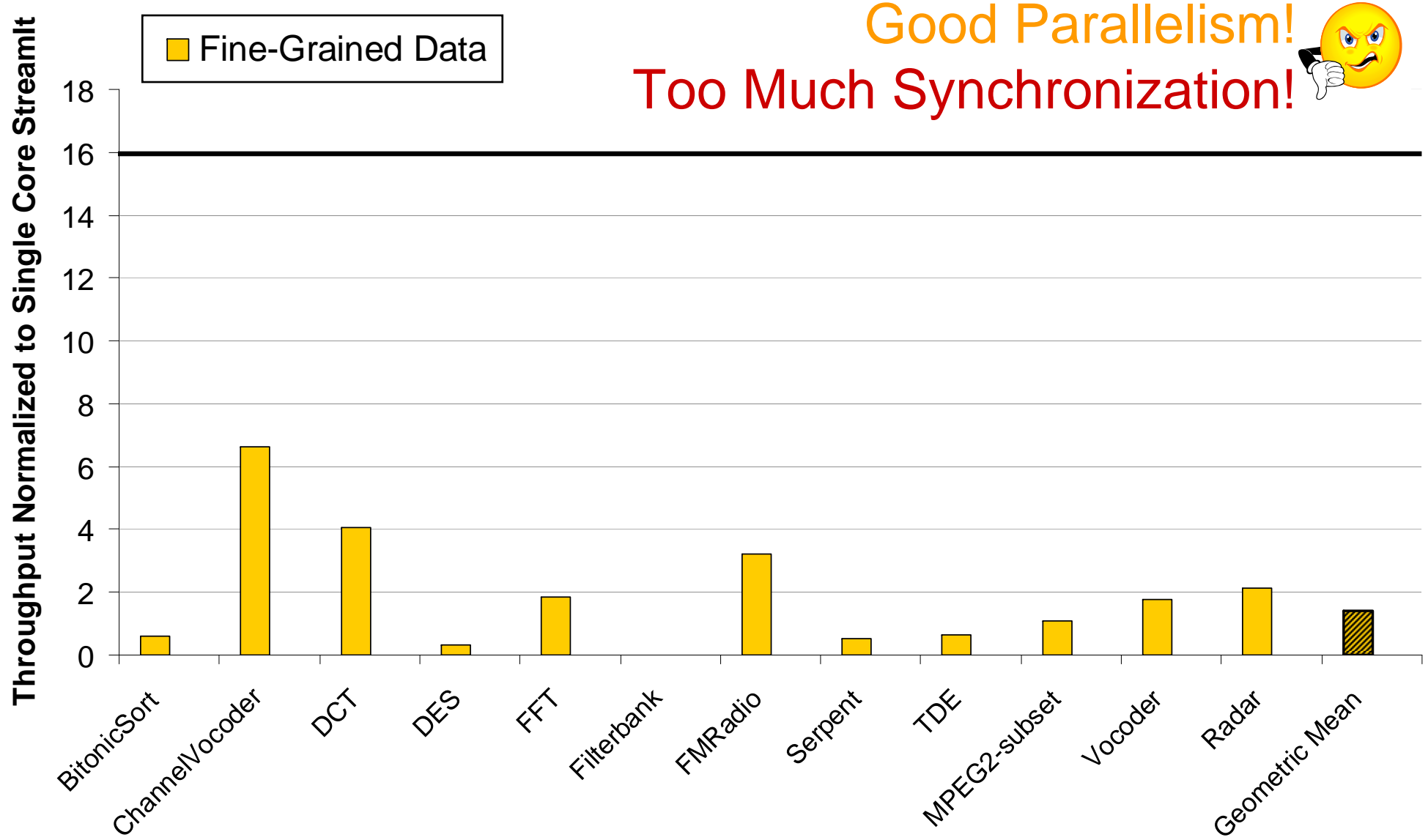
Baseline: Fine-Grained Data Parallelism



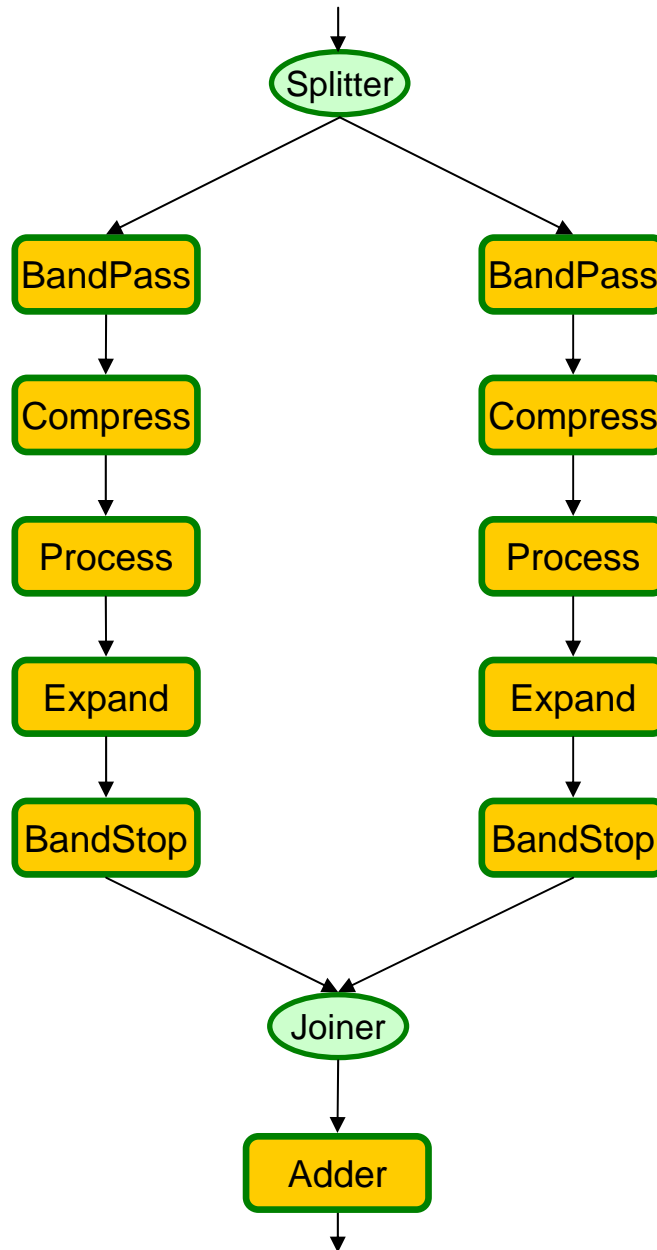
Evaluation: Fine-Grained Data Parallelism



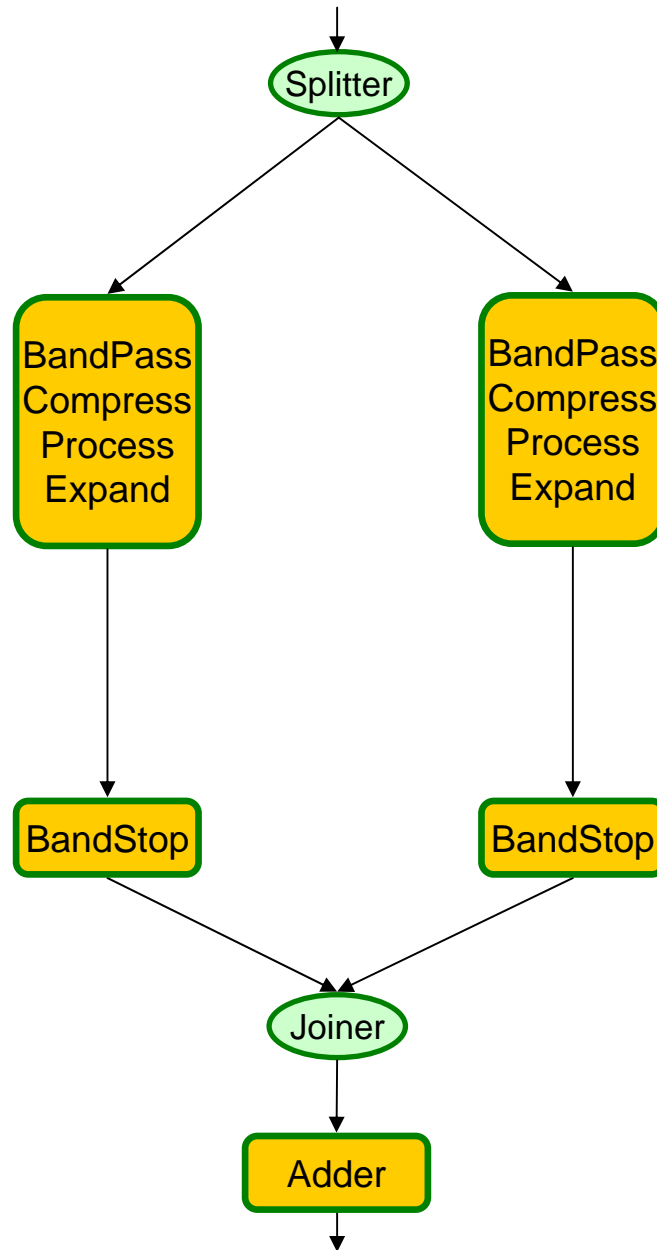
Evaluation: Fine-Grained Data Parallelism



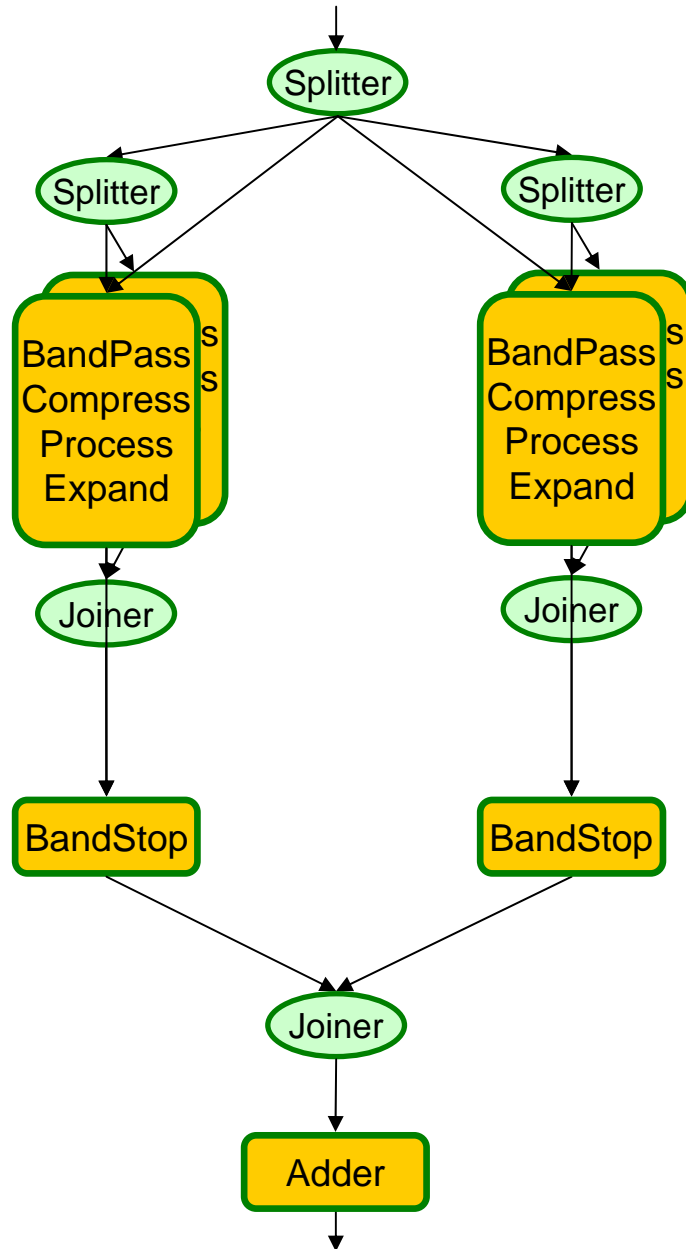
Coarsening the Granularity



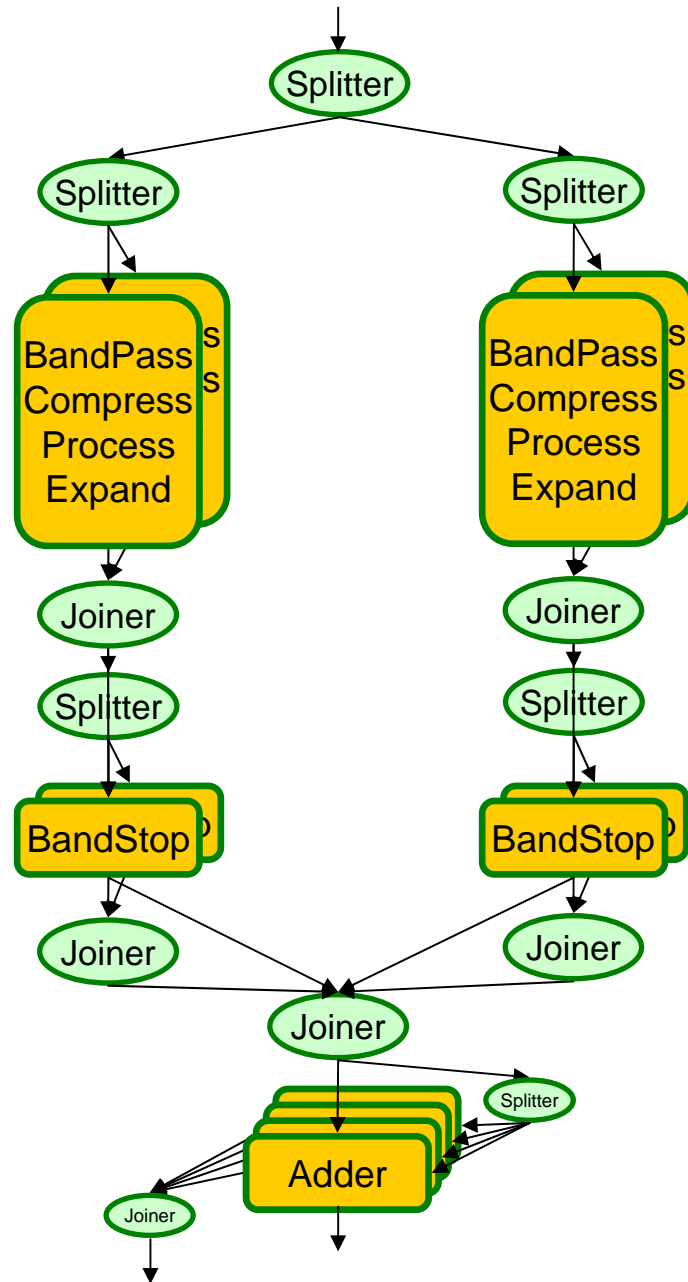
Coarsening the Granularity



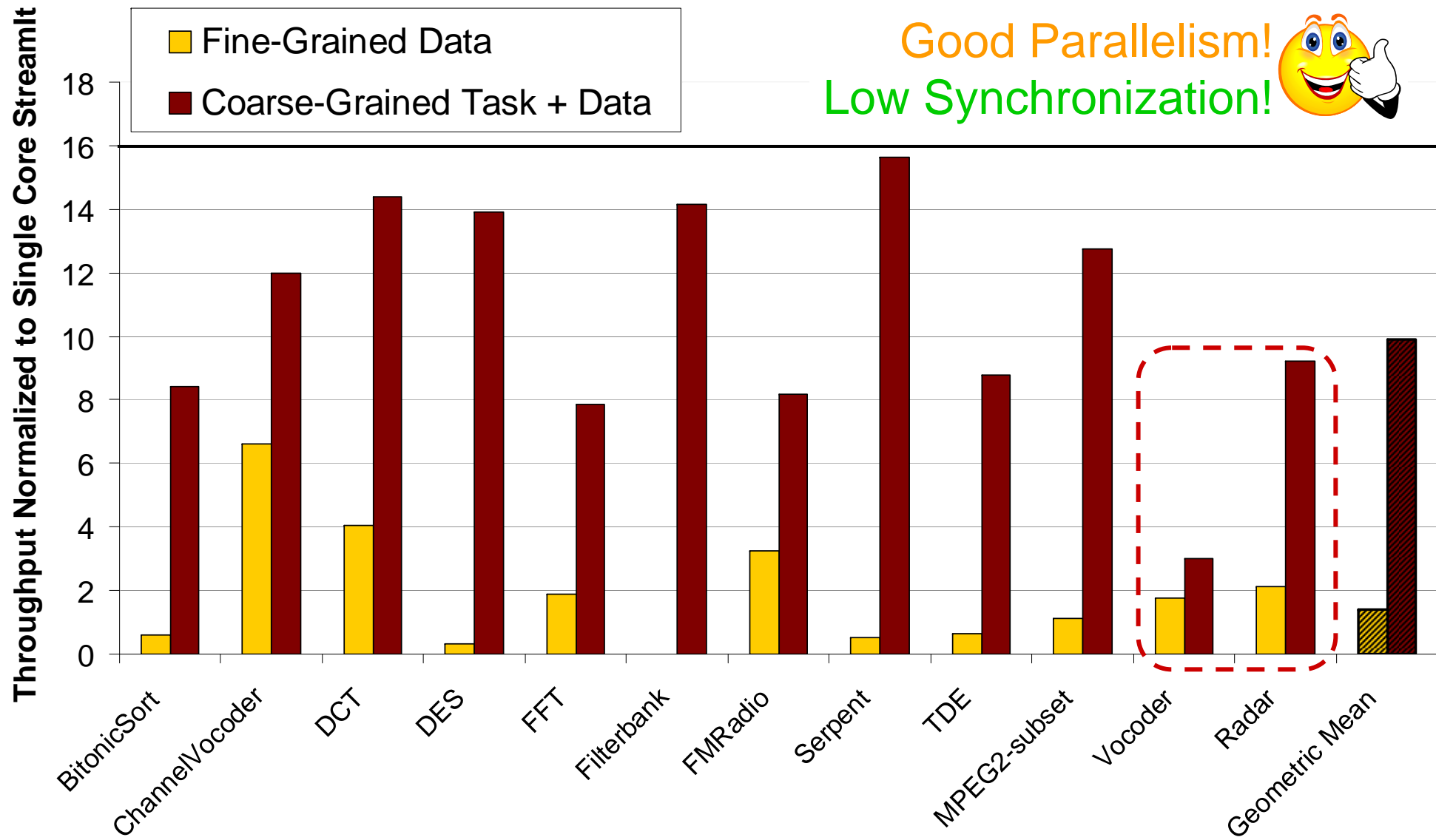
Coarsening the Granularity



Coarsening the Granularity



Evaluation: Coarse-Grained Data Parallelism

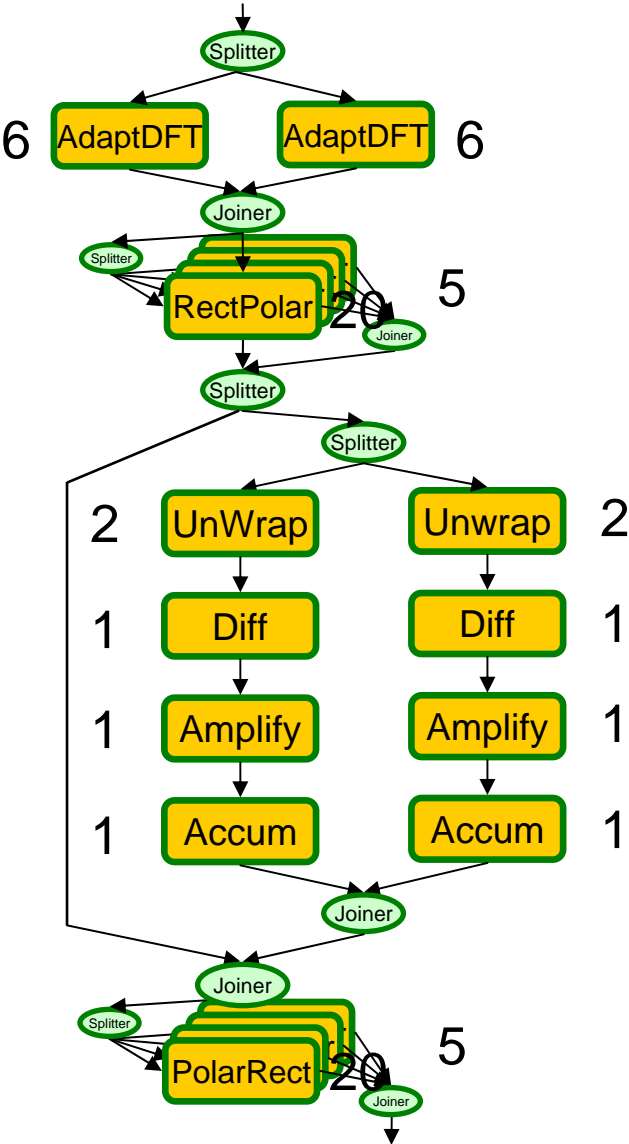


Simplified Vocoder



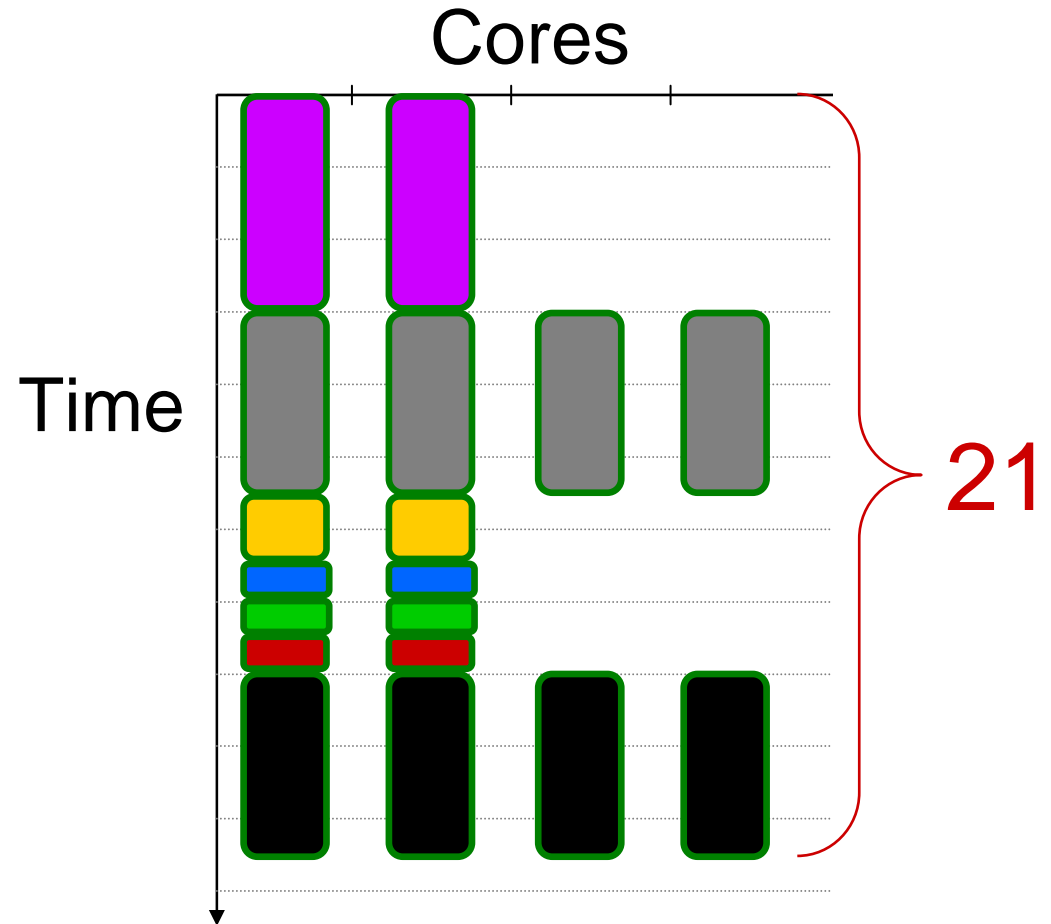
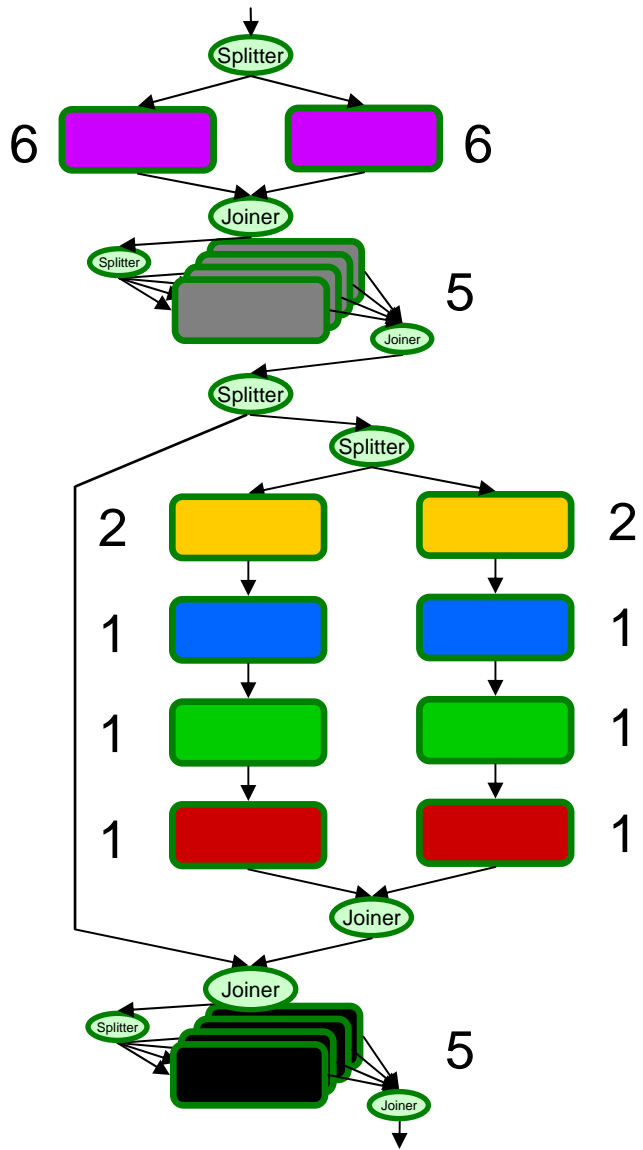
Target a 4-core machine

Data Parallelize



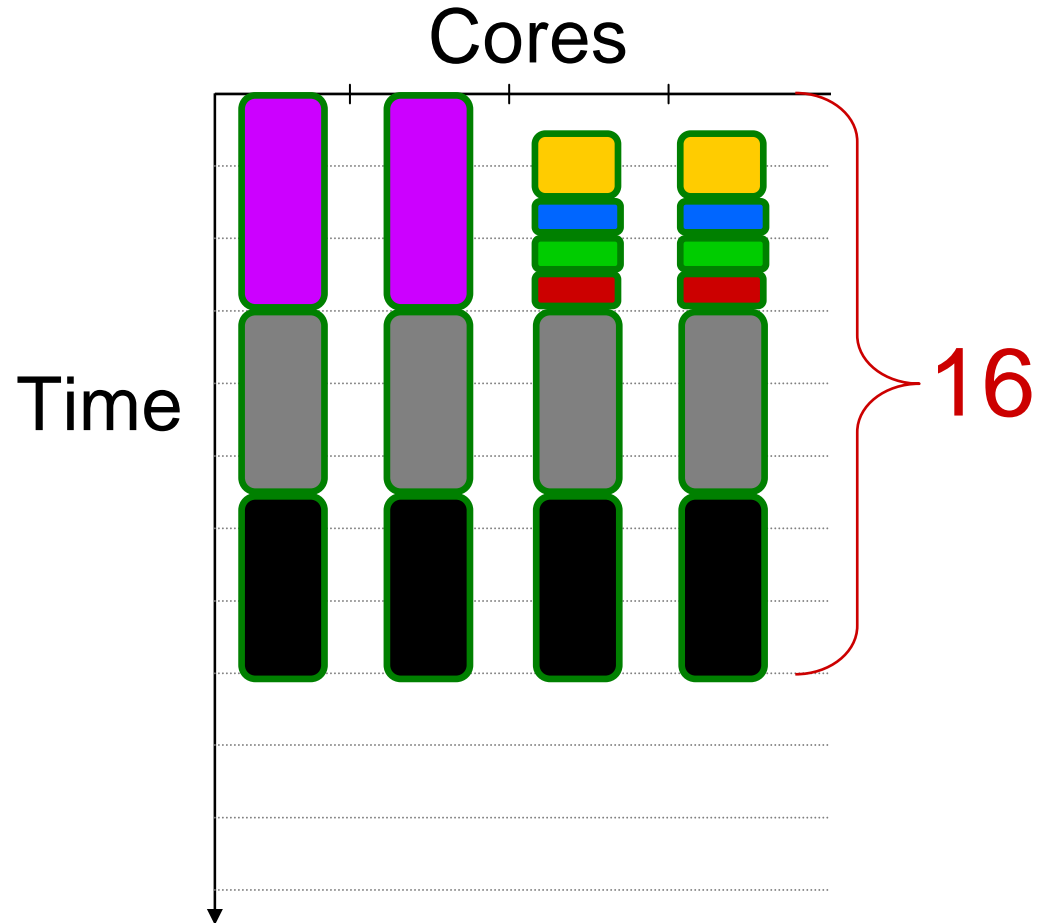
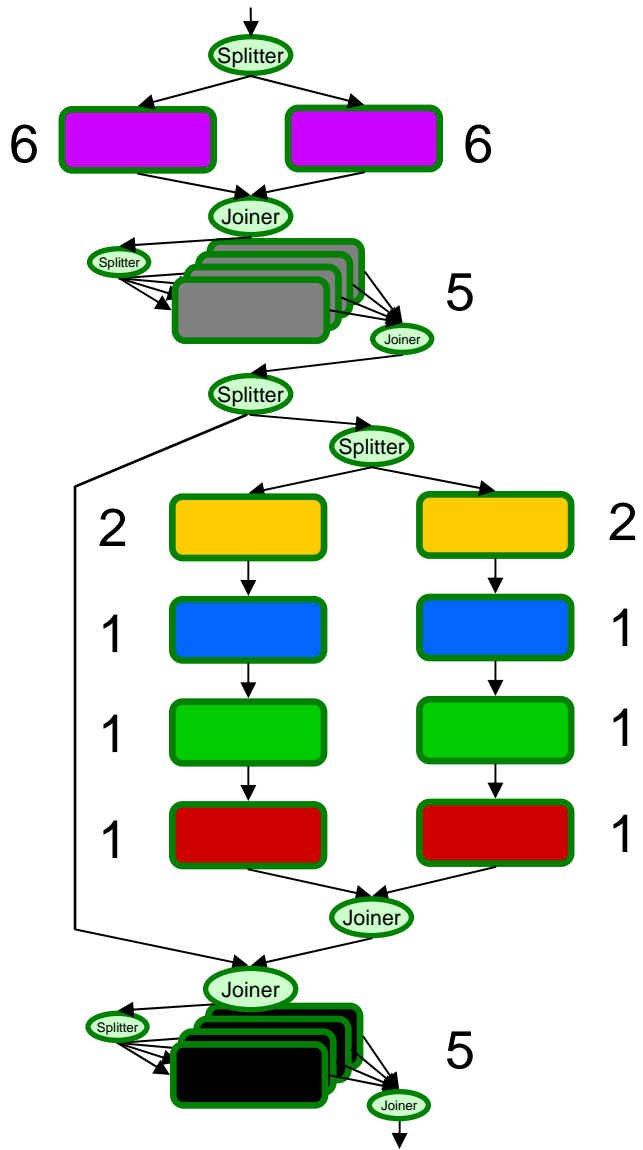
Target a 4-core machine

Data + Task Parallel Execution



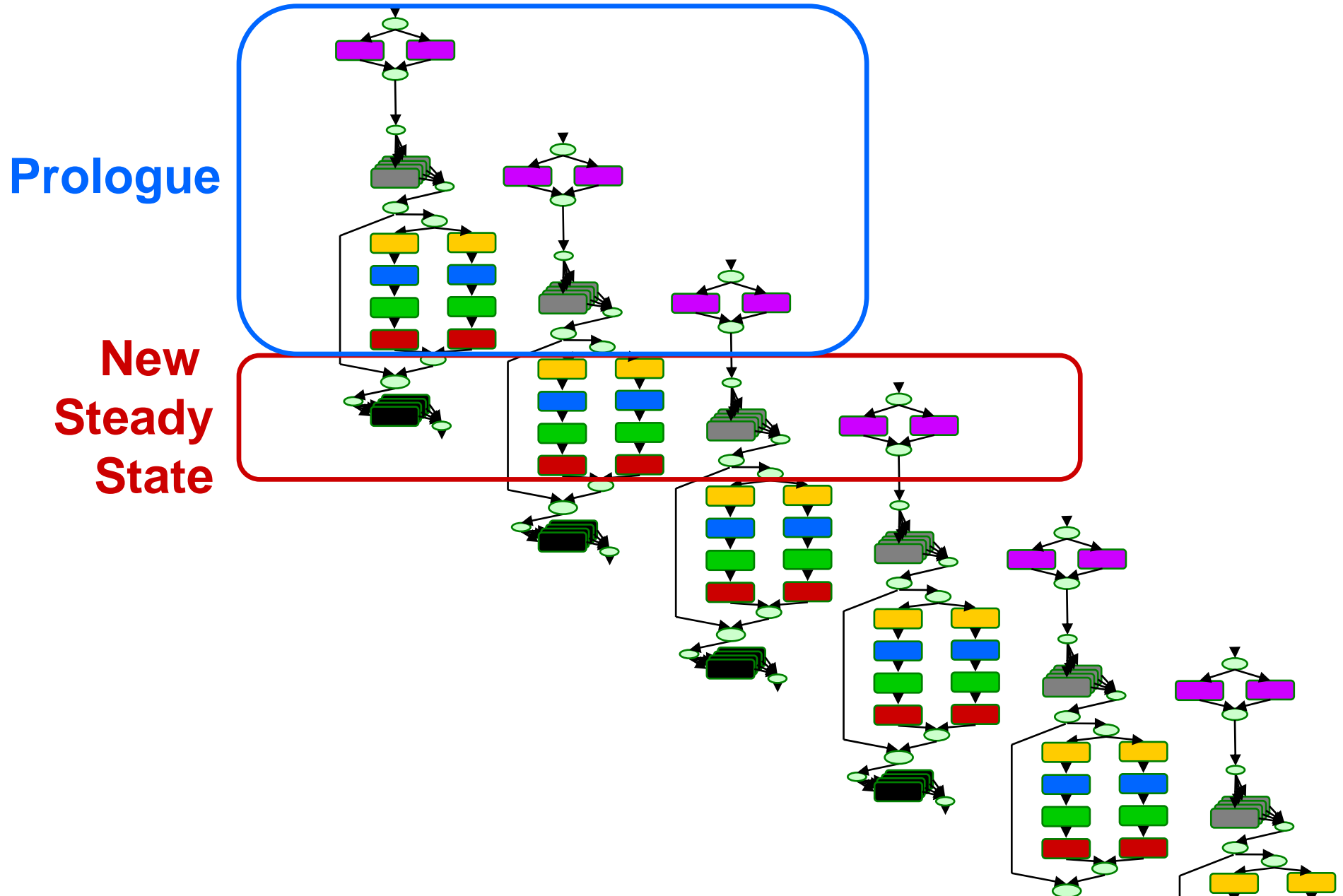
Target a 4-core machine

We Can Do Better

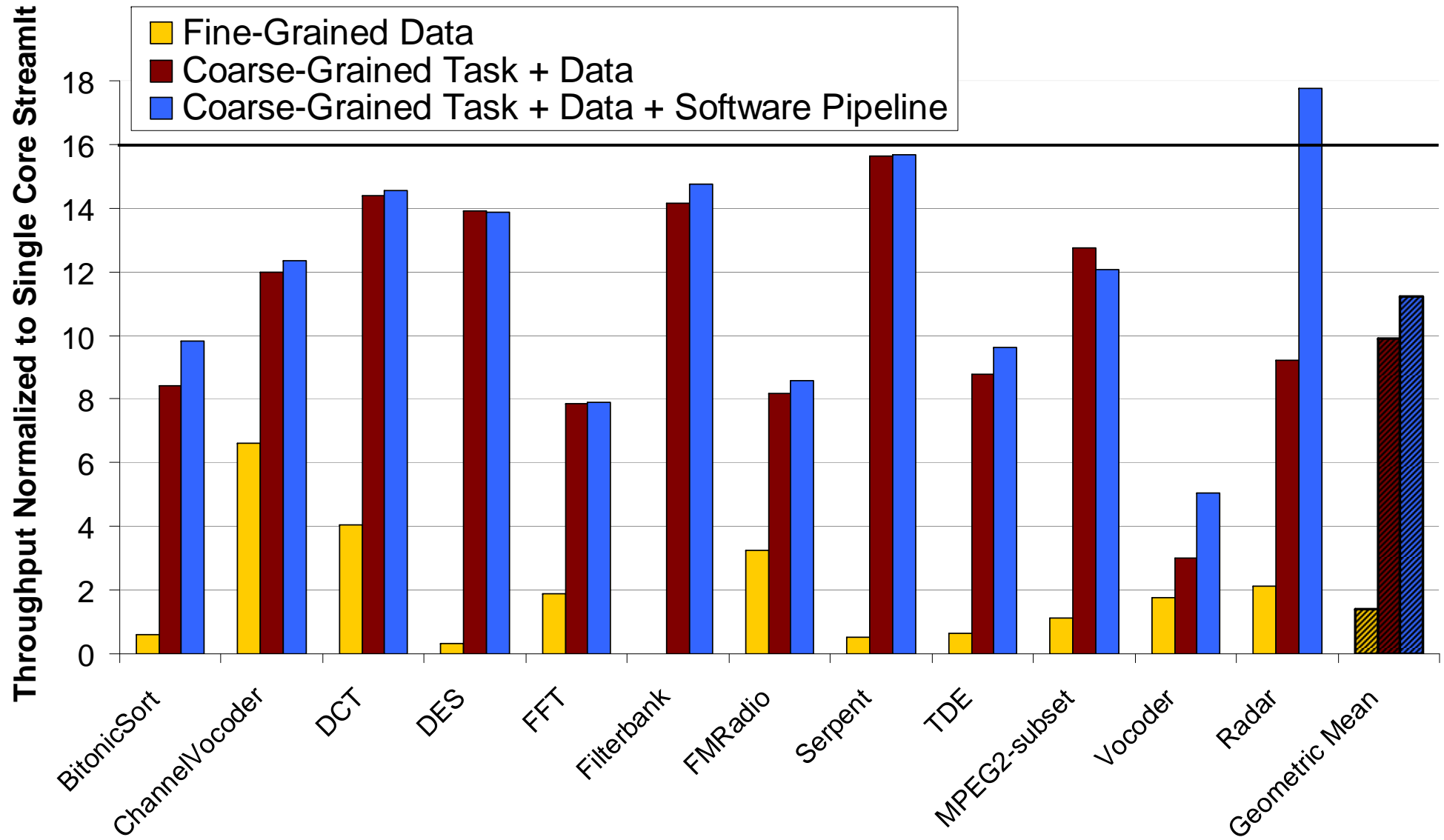


Target a 4-core machine

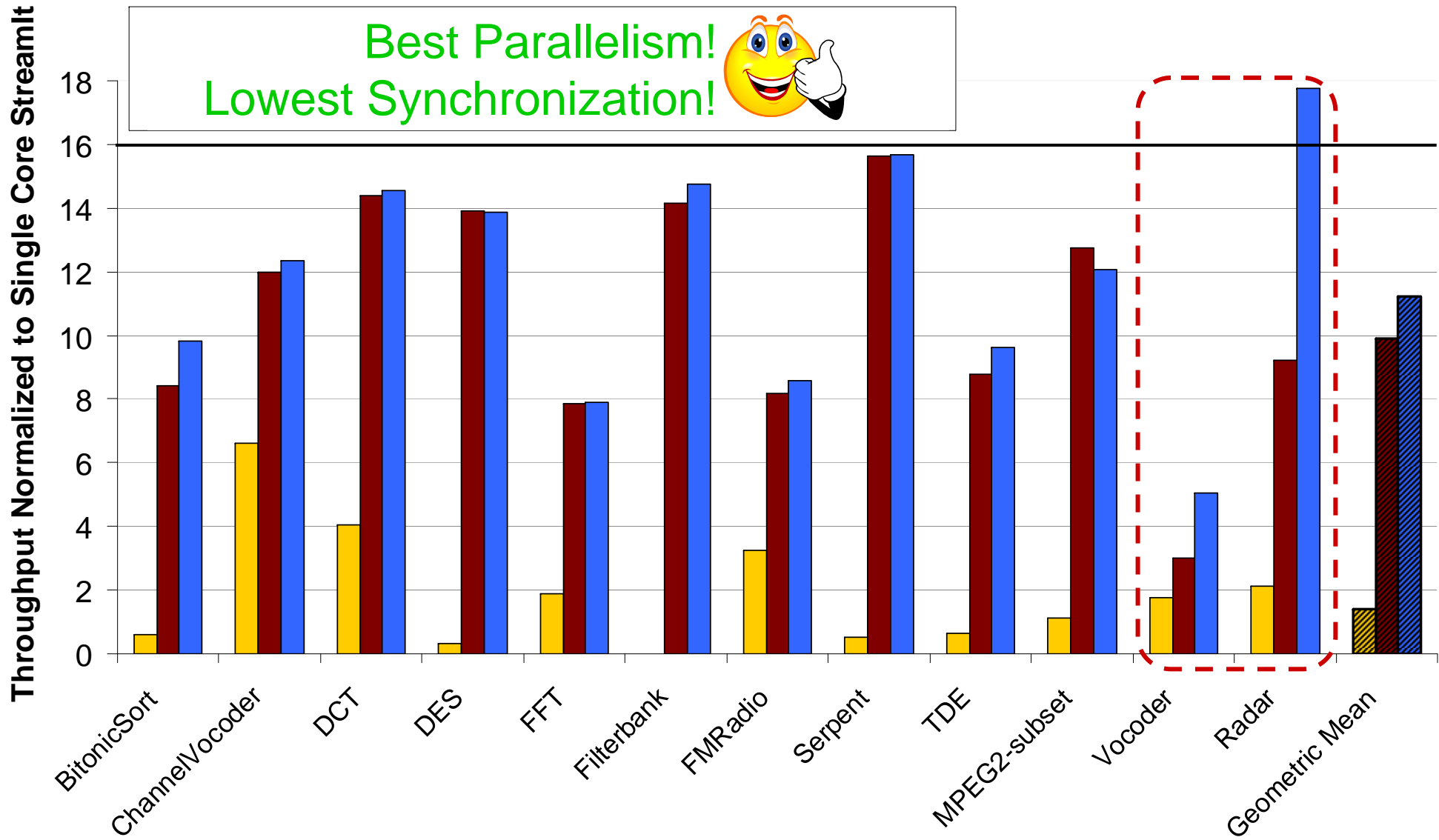
Coarse-Grained Software Pipelining



Evaluation: Coarse-Grained Task + Data + Software Pipelining

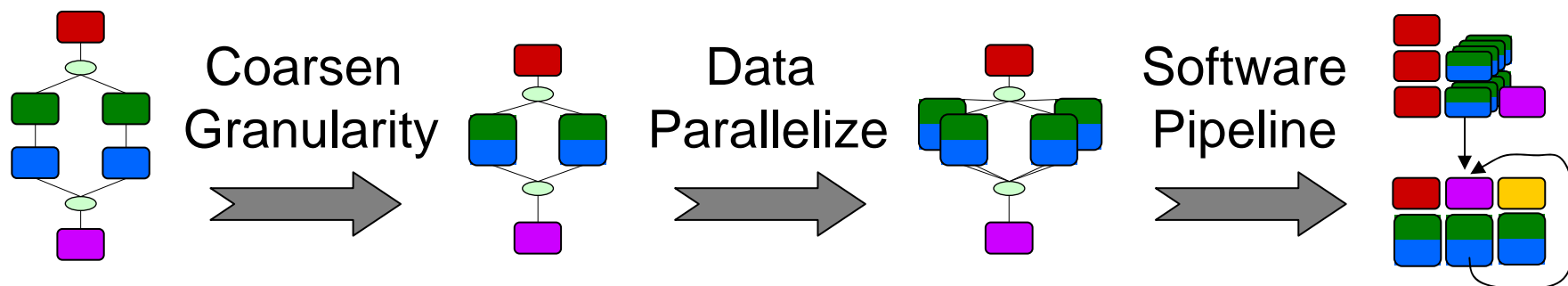


Evaluation: Coarse-Grained Task + Data + Software Pipelining



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 [\[Zhang, dasCMP'07\]](#)

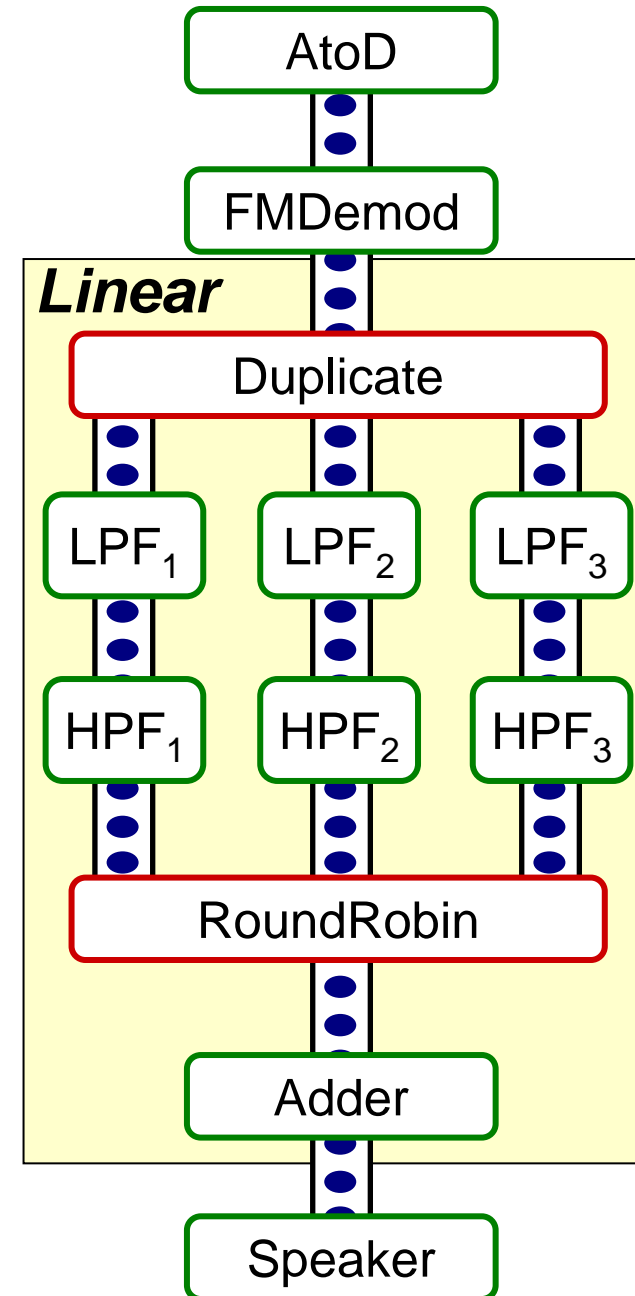
Part 3: Domain-Specific Optimizations

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

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

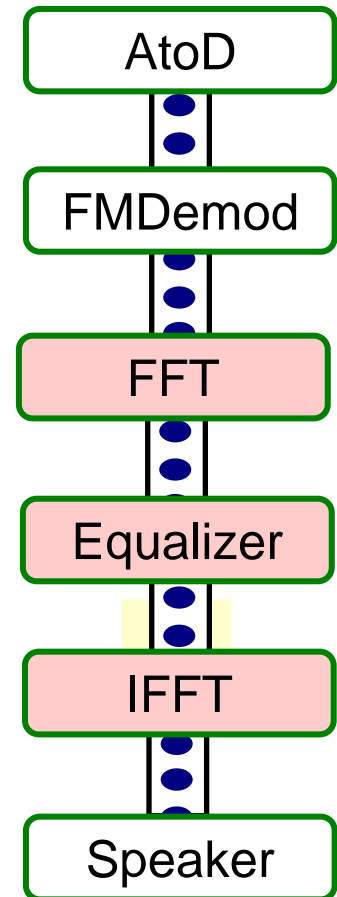
DSP Optimization Process

- Given specification of algorithm, minimize the computation cost



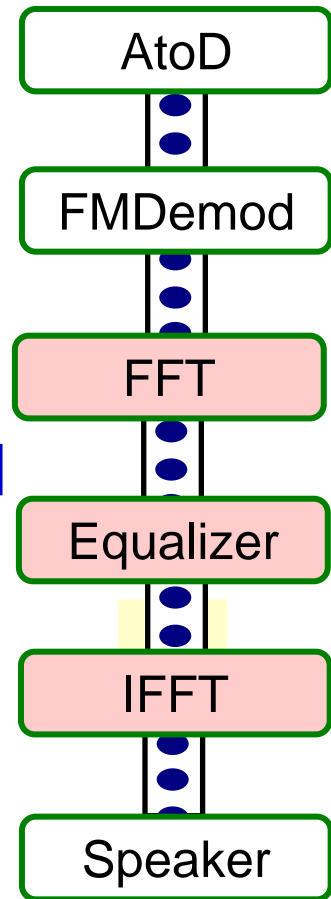
DSP Optimization Process

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



DSP Optimization Process

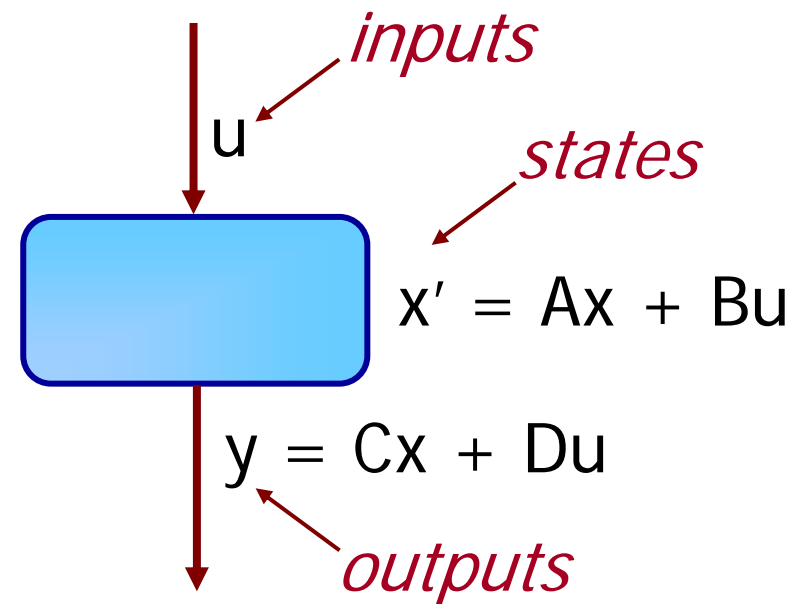
- **Given specification of algorithm, minimize the computation cost**
 - Currently done by hand (MATLAB)
- **Can compiler replace DSP expert?**
 - Library generators limited [[Spiral](#)] [[FFTW](#)] [[ATLAS](#)]
 - Enable unified development environment



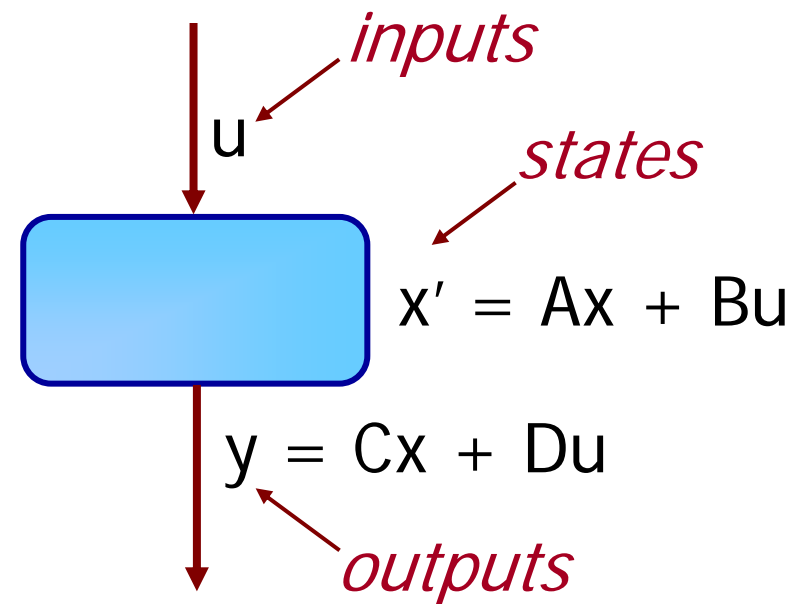
Focus: Linear State Space Filters

- **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



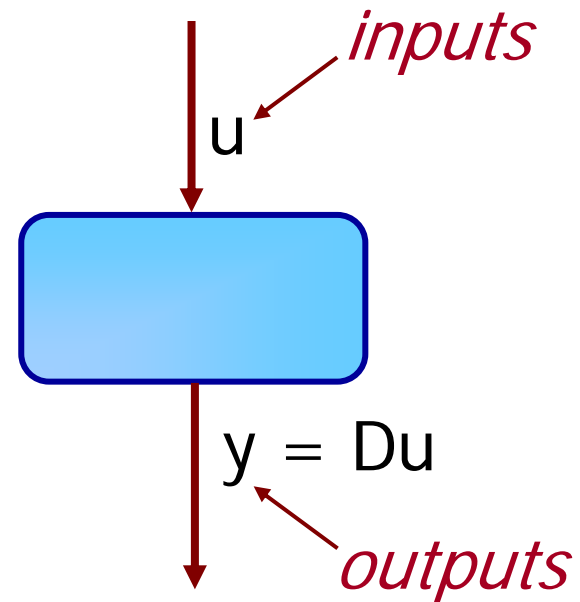
Focus: Linear State Space Filters



Focus: Linear Filters

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

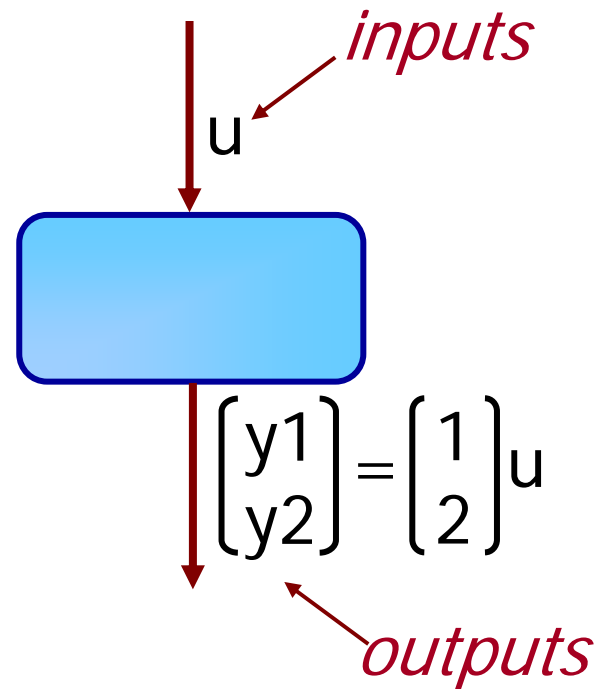
Linear
dataflow
analysis



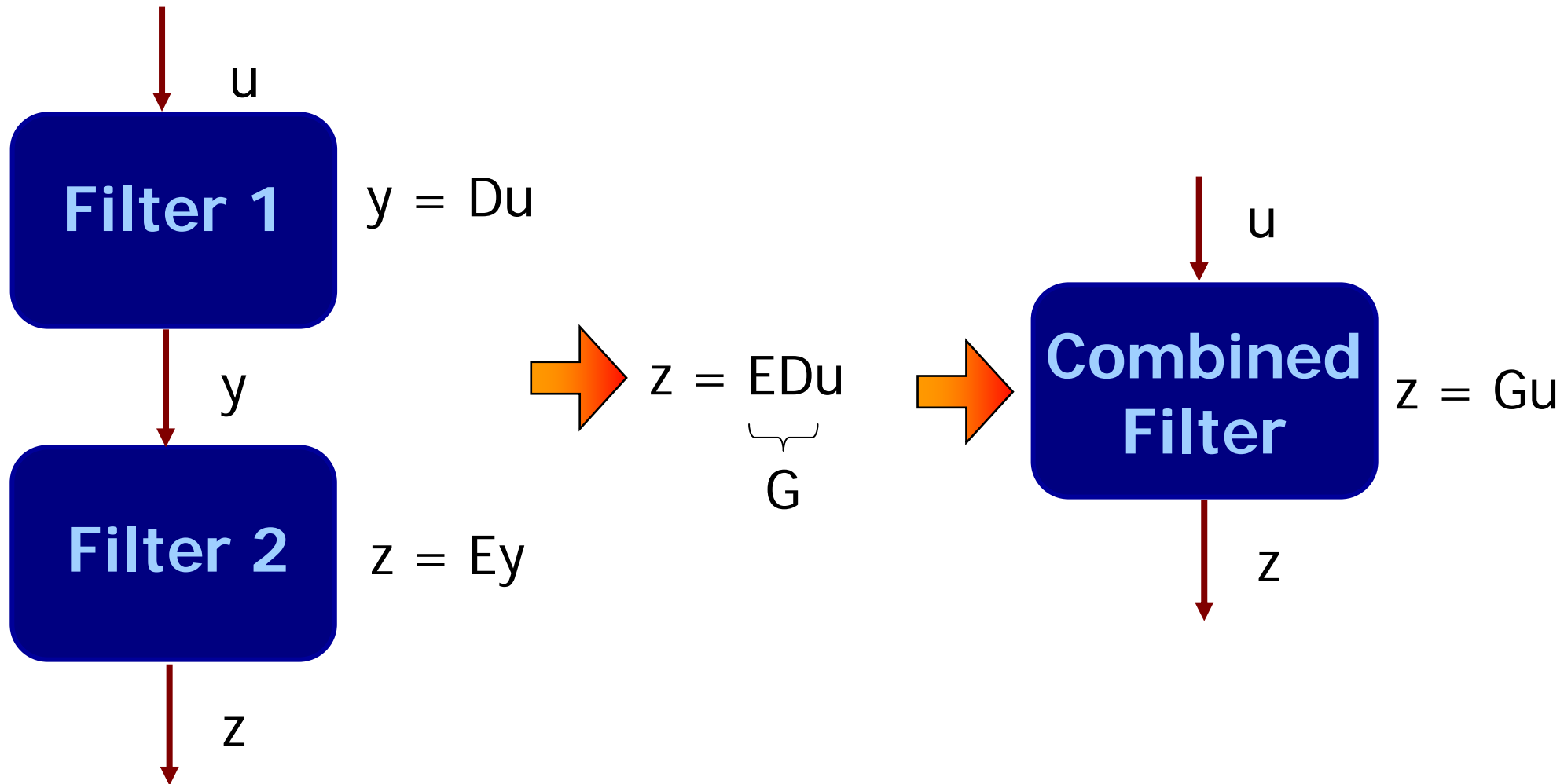
Focus: Linear Filters

```
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

$\frac{6 \text{ mults}}{\text{output}}$



u



$$D = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

y



$$E = [4 \quad 5 \quad 6]$$

z

$\frac{1 \text{ mults}}{\text{output}}$



u



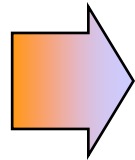
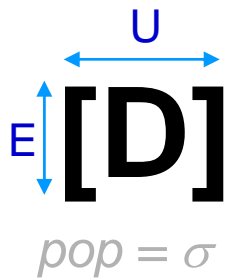
$$G = [32]$$

z

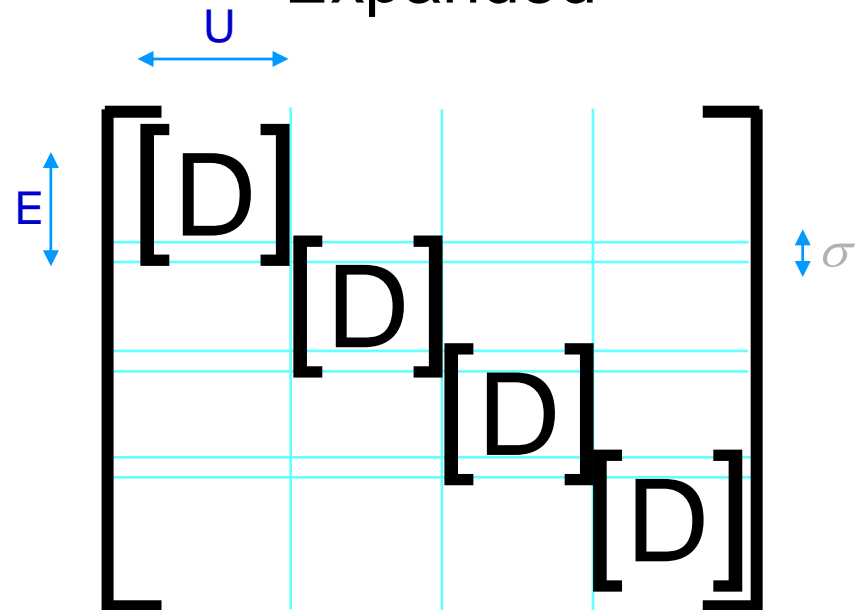
The General Case

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

Original



Expanded



The General Case

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

$$A^e = A^n A_{\text{pre}}$$

$$B^e = \begin{bmatrix} A^n B_{\text{pre}} & A^{n-1} B & A^{n-2} B & \dots & B \end{bmatrix}$$

$$C^e = \begin{bmatrix} CA_{\text{pre}} \\ CAA_{\text{pre}} \\ \dots \\ CA^{n-1} A_{\text{pre}} \end{bmatrix}$$

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

The General Case

Pipelines

$$\begin{aligned} A &= \begin{bmatrix} A_1 & 0 \\ B_2 C_1 & A_2 \end{bmatrix} & A_{\text{pre}} &= \begin{bmatrix} A_1^e & 0 \\ B_{\text{pre}2} C_1^e & A_{\text{pre}2} \end{bmatrix} \\ B &= \begin{bmatrix} B_1 \\ B_2 D_1 \end{bmatrix} & B_{\text{pre}} &= \begin{bmatrix} B_1^e \\ B_{\text{pre}2} D_1^e \end{bmatrix} \\ C &= \begin{bmatrix} D_2 C_1 & C_2 \end{bmatrix} & \overrightarrow{\text{initVec}} &= \begin{bmatrix} \overrightarrow{\text{initVec}_1} \\ \overrightarrow{\text{initVec}_2} \end{bmatrix} \\ D &= D_2 D_1 \end{aligned}$$

Feedback Loops

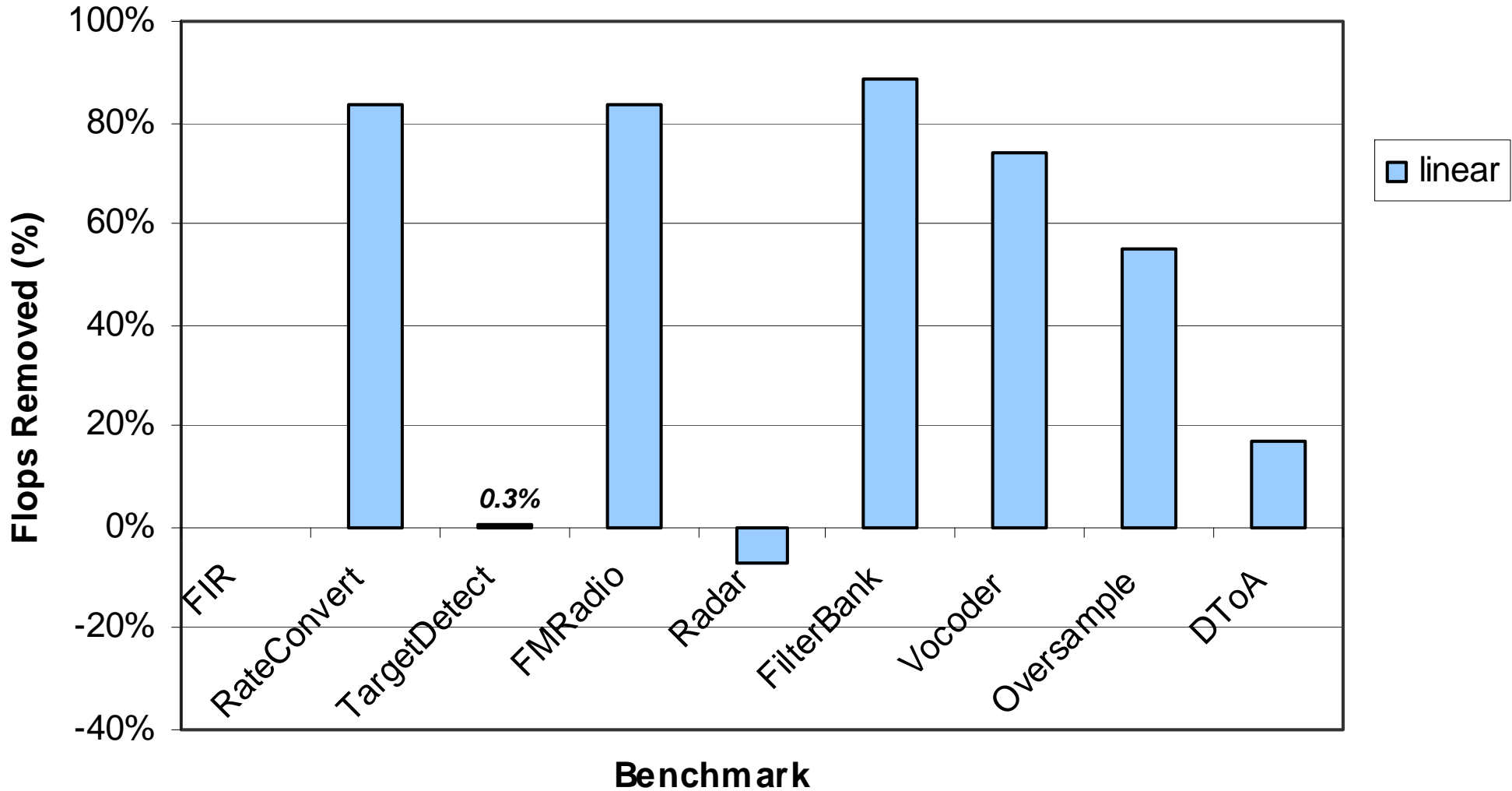
$$\begin{aligned} \vec{x}_1 &= A_1 \vec{x}_1 + B_1 \vec{u}_1 = A_1 \vec{x}_1 + B_1 \vec{y} = A_1 \vec{x}_1 + B_1 (C_2 \vec{x}_2 + D_{2,1} \vec{u} + D_{2,2} C_3 \vec{x}_3) \\ &= A_1 \vec{x}_1 + B_1 C_2 \vec{x}_2 + B_1 D_{2,1} \vec{u} + B_1 D_{2,2} C_3 \vec{x}_3 \\ \vec{x}_2 &= A_2 \vec{x}_2 + B_2 \vec{u}_2 = A_2 \vec{x}_2 + B_{2,1} \vec{u} + B_{2,2} \vec{y}_3 = A_2 \vec{x}_2 + B_{2,1} \vec{u} + B_{2,2} C_3 \vec{x}_3 \\ \vec{y}_2 &= C_2 \vec{x}_2 + D_2 \vec{u}_2 = C_2 \vec{x}_2 + D_{2,1} \vec{u} + D_{2,2} \vec{y}_3 = C_2 \vec{x}_2 + D_{2,1} \vec{u} + D_{2,2} C_3 \vec{x}_3 \\ \vec{x}_3 &= A_3 \vec{x}_3 + B_3 \vec{u}_3 = A_3 \vec{x}_3 + B_3 \vec{y}_1 = A_3 \vec{x}_3 + B_3 (C_1 \vec{x}_1 + D_1 \vec{u}_1) \\ &= A_3 \vec{x}_3 + B_3 (C_1 \vec{x}_1 + D_1 \vec{y}) = A_3 \vec{x}_3 + B_3 (C_1 \vec{x}_1 + D_1 (C_2 \vec{x}_2 + D_{2,1} \vec{u} + D_{2,2} C_3 \vec{x}_3)) \\ &= A_3 \vec{x}_3 + B_3 C_1 \vec{x}_1 + B_3 D_1 C_2 \vec{x}_2 + B_3 D_1 D_{2,1} \vec{u} + B_3 D_1 D_{2,2} C_3 \vec{x}_3 \end{aligned}$$

The General Case

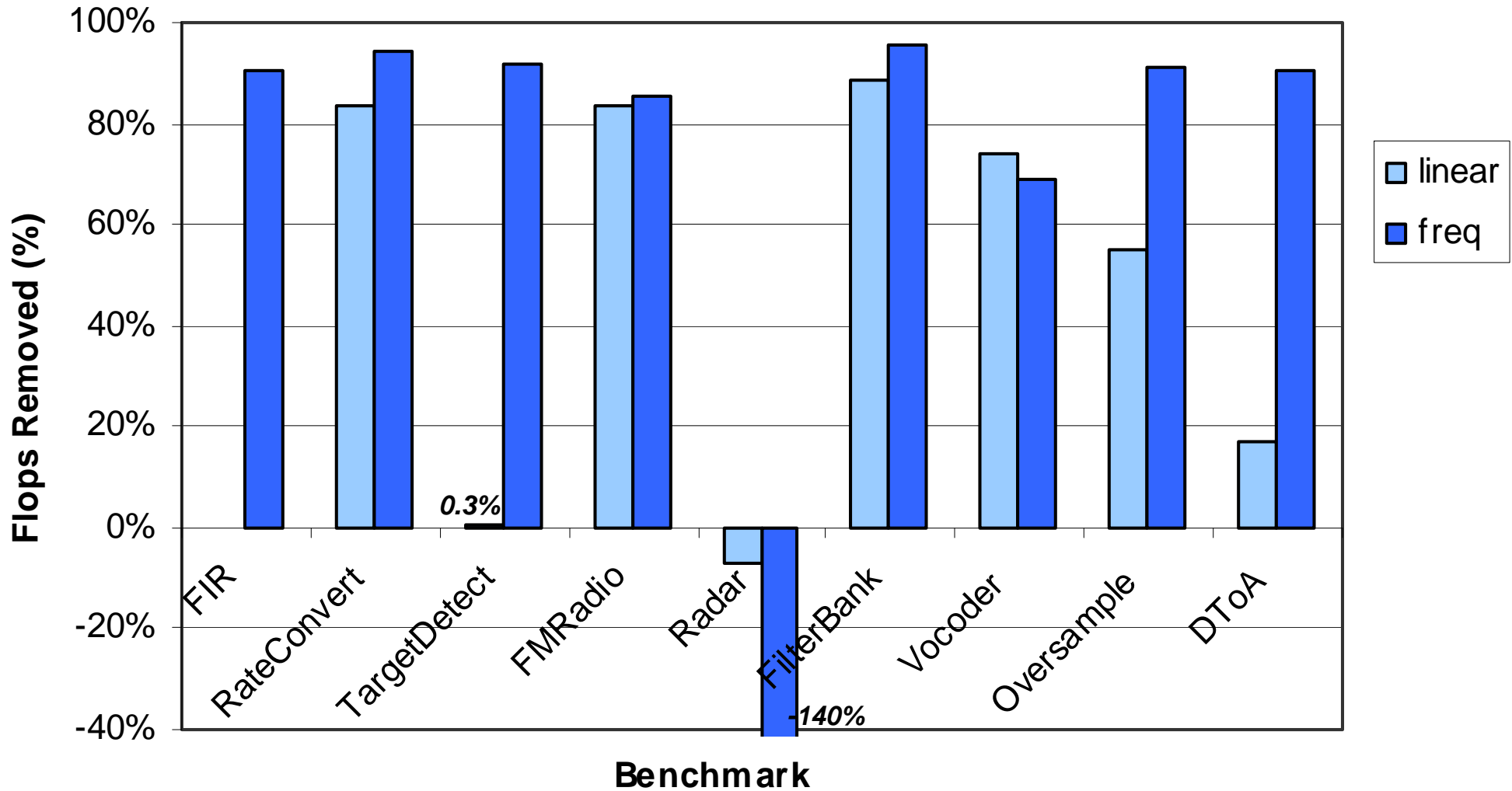
Splitjoins

$$\begin{array}{l}
 \mathbf{A} = \begin{bmatrix} A_s & 0 & 0 & \dots & 0 \\ A_{1rs} & A_{1rr} & 0 & \dots & 0 \\ A_{2rs} & 0 & A_{2rr} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ A_{krs} & 0 & 0 & \dots & A_{krs} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} B_s \\ B_{1r} \\ B_{2r} \\ \dots \\ B_{kr} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} C_{1s1} & C_{1r1} & 0 & \dots & 0 \\ C_{2s1} & C_{2r1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ C_{ks1} & 0 & 0 & \dots & C_{kr1} \\ \dots & \dots & \dots & \dots & \dots \\ C_{1sk} & C_{1rk} & 0 & \dots & 0 \\ C_{2sk} & C_{2rk} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ C_{ksk} & 0 & 0 & \dots & C_{krk} \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} D_{11} \\ D_{21} \\ \dots \\ D_{k1} \\ \dots \\ D_{1k} \\ D_{2k} \\ \dots \\ D_{kk} \end{bmatrix} \\
 \\
 \mathbf{C}_i = \begin{bmatrix} C_{is1} & C_{ir1} \\ C_{is2} & C_{ir2} \\ \dots & \dots \\ C_{isexecutions} & C_{irexecutions} \end{bmatrix} \quad \mathbf{D}_i = \begin{bmatrix} D_{i1} \\ D_{i2} \\ \dots \\ D_{iexecutions} \end{bmatrix} \\
 \\
 \mathbf{A}_{pre} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & A_{pre1rr} & 0 & \dots & 0 \\ 0 & 0 & A_{pre2rr} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & A_{prekrr} \end{bmatrix} \quad \mathbf{B}_{pre} = \begin{bmatrix} B_{pres} \\ B_{pre1r} \\ B_{pre2r} \\ \dots \\ B_{prekr} \end{bmatrix} \quad \vec{\text{initVec}} = \begin{bmatrix} \vec{0} \\ \vec{\text{initVec}}_{1r} \\ \vec{\text{initVec}}_{2r} \\ \dots \\ \vec{\text{initVec}}_{kr} \end{bmatrix}
 \end{array}$$

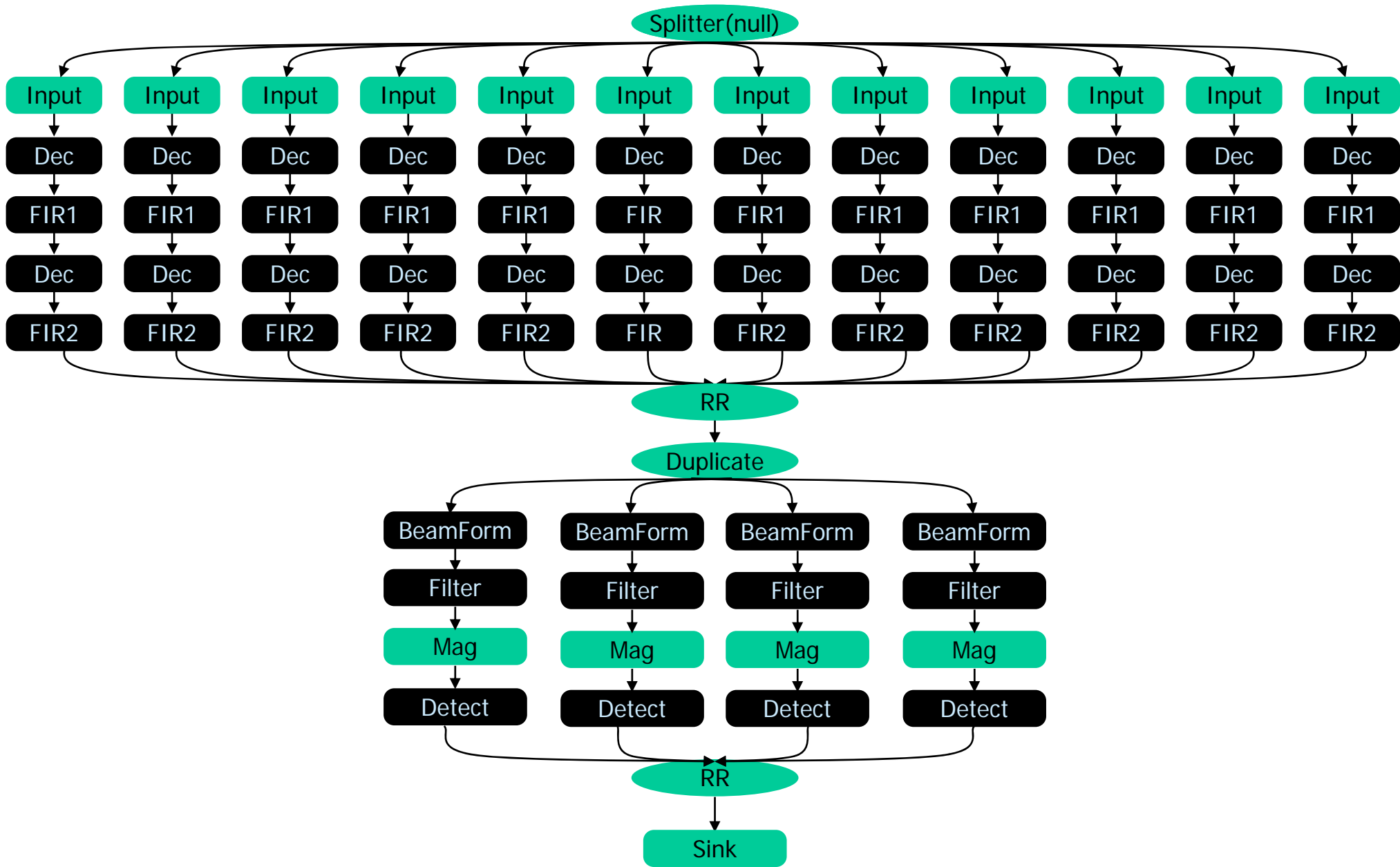
Floating-Point Operations Reduction



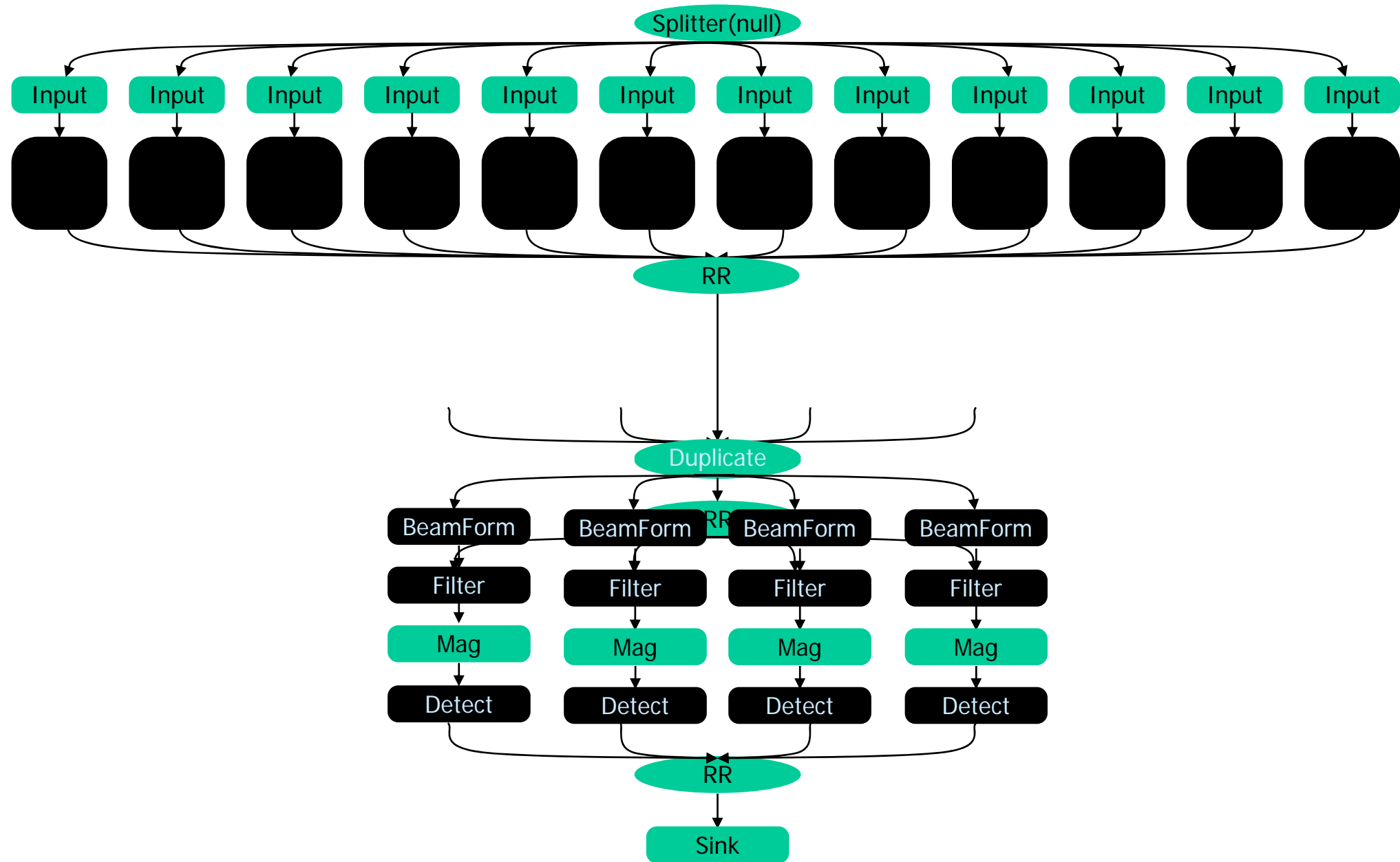
Floating-Point Operations Reduction



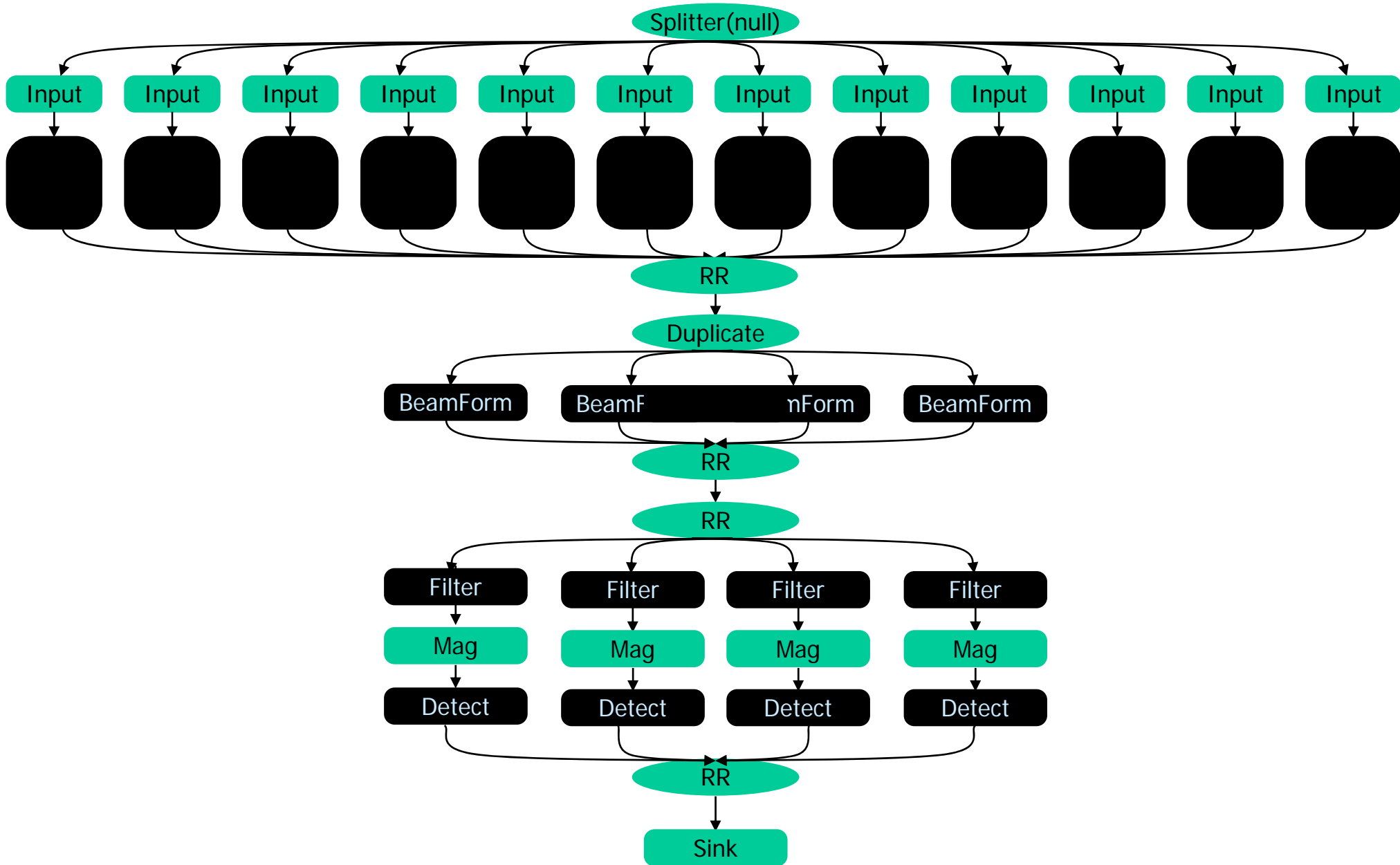
Radar (Transformation Selection)



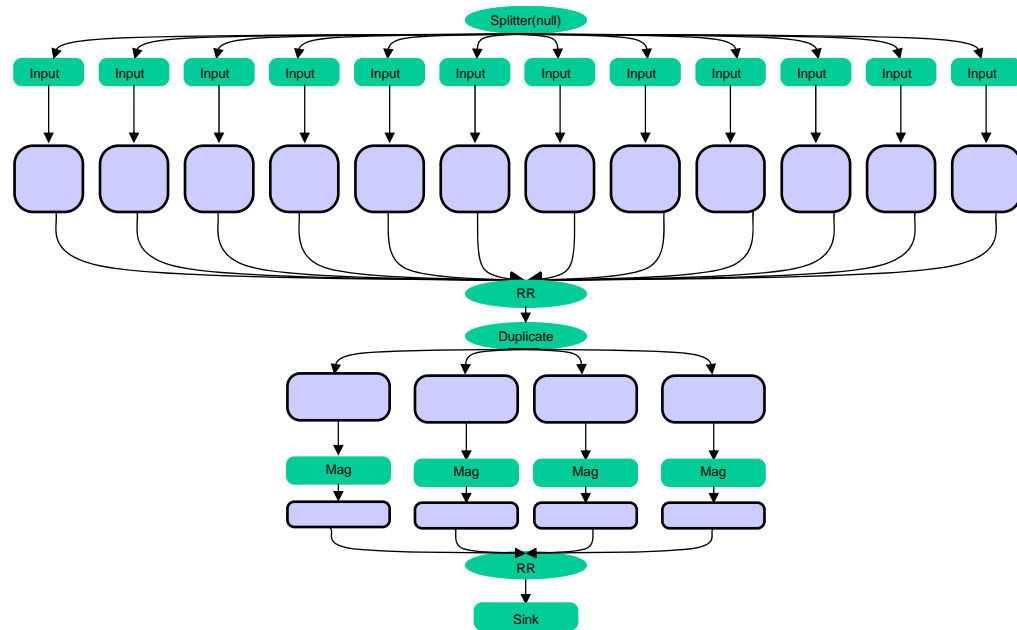
Radar (Transformation Selection)



Radar (Transformation Selection)



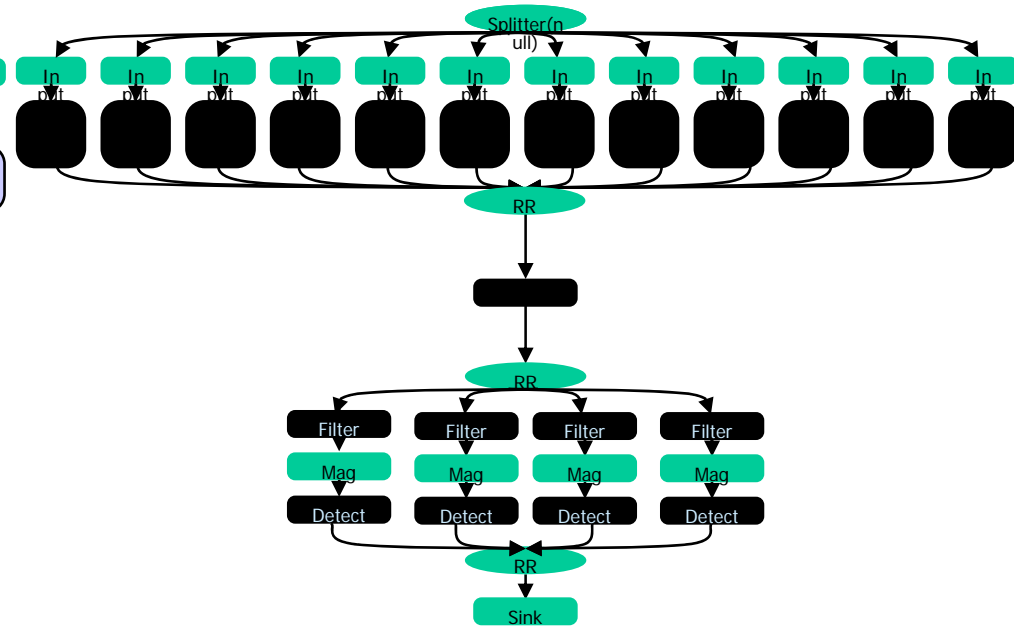
Radar



Maximal Combination and
Shifting to Frequency Domain



2.4 times as
many FLOPS

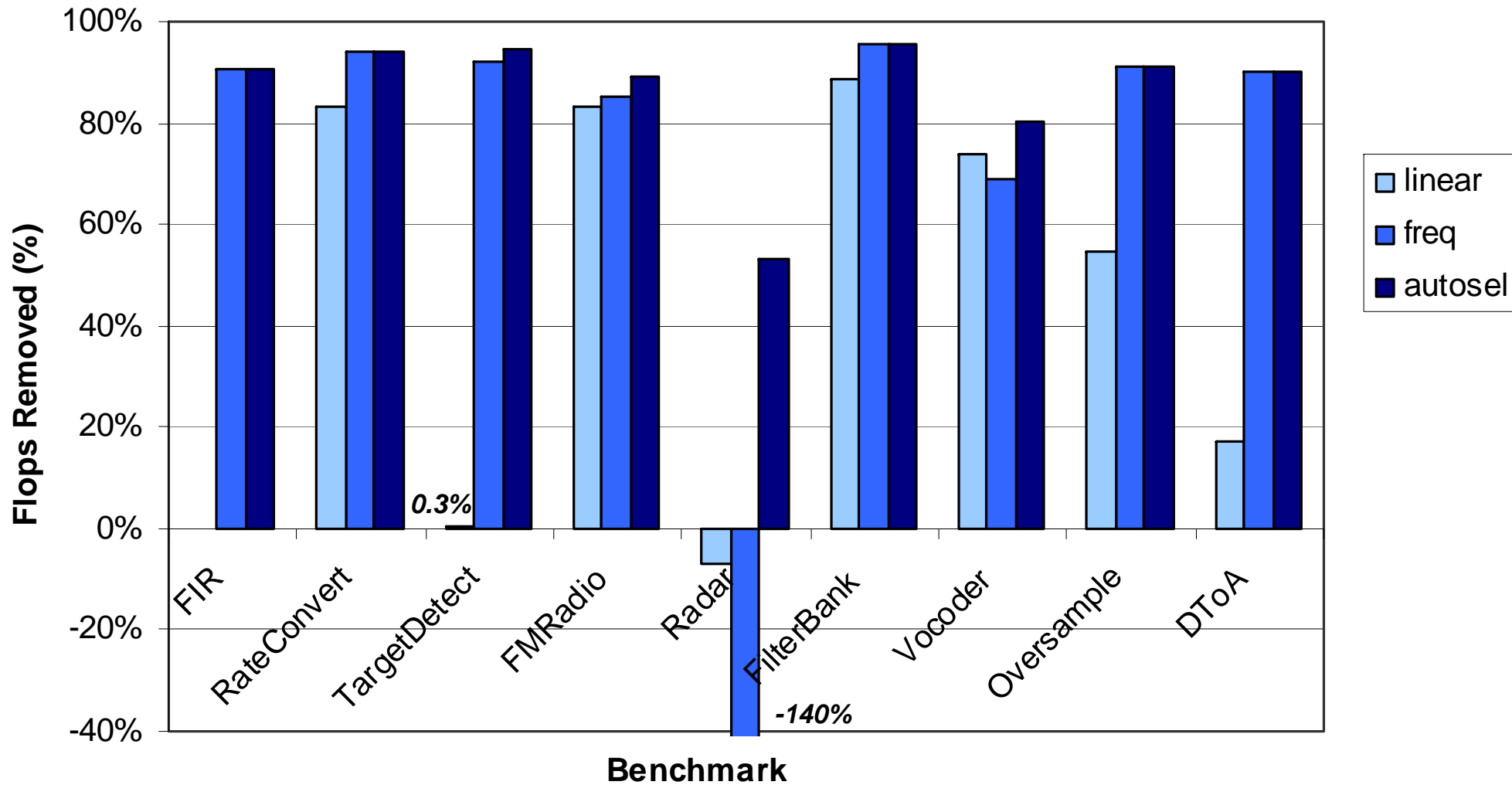


Using Transformation
Selection

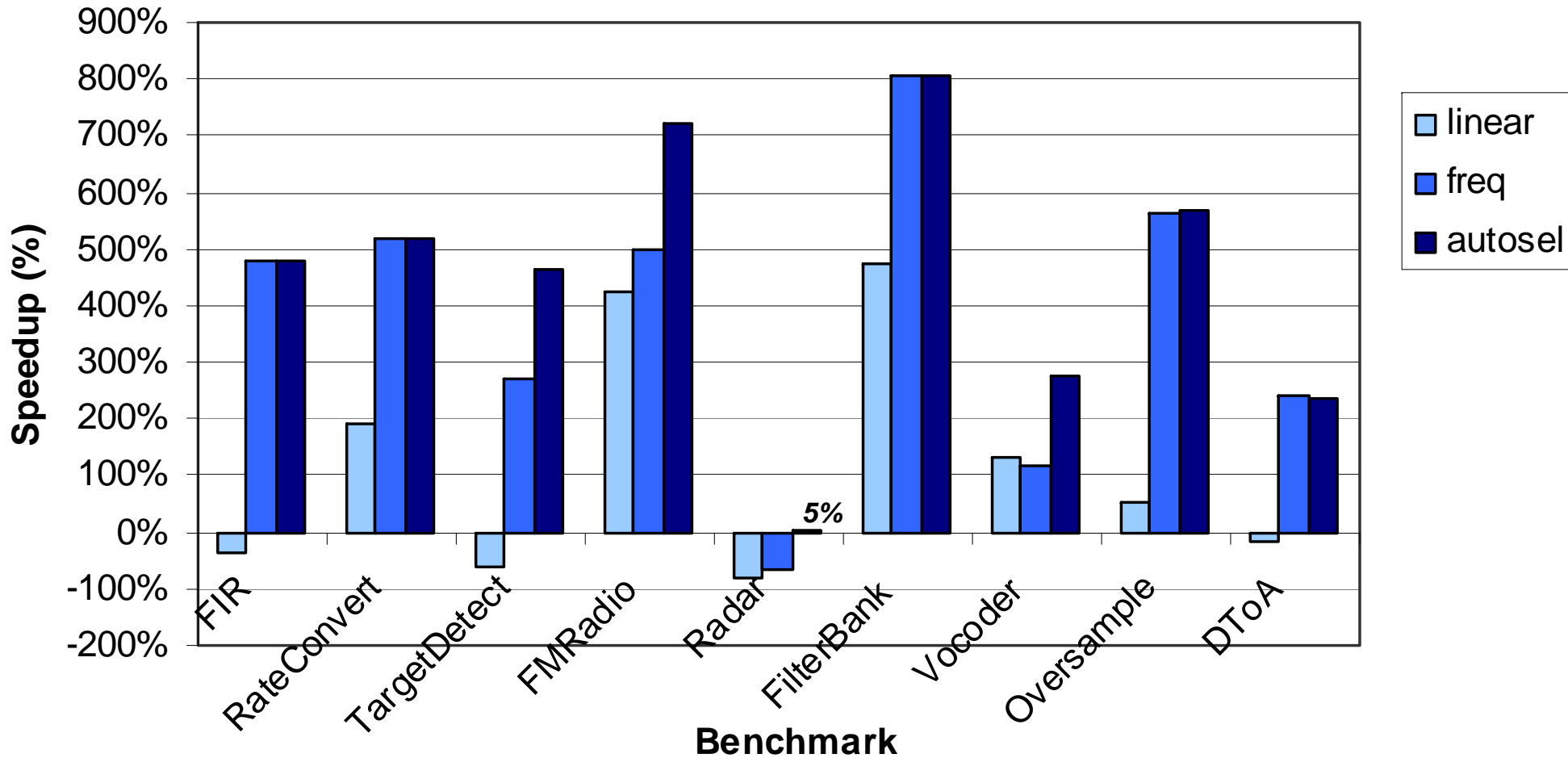


half as many
FLOPS

Floating Point Operations Reduction

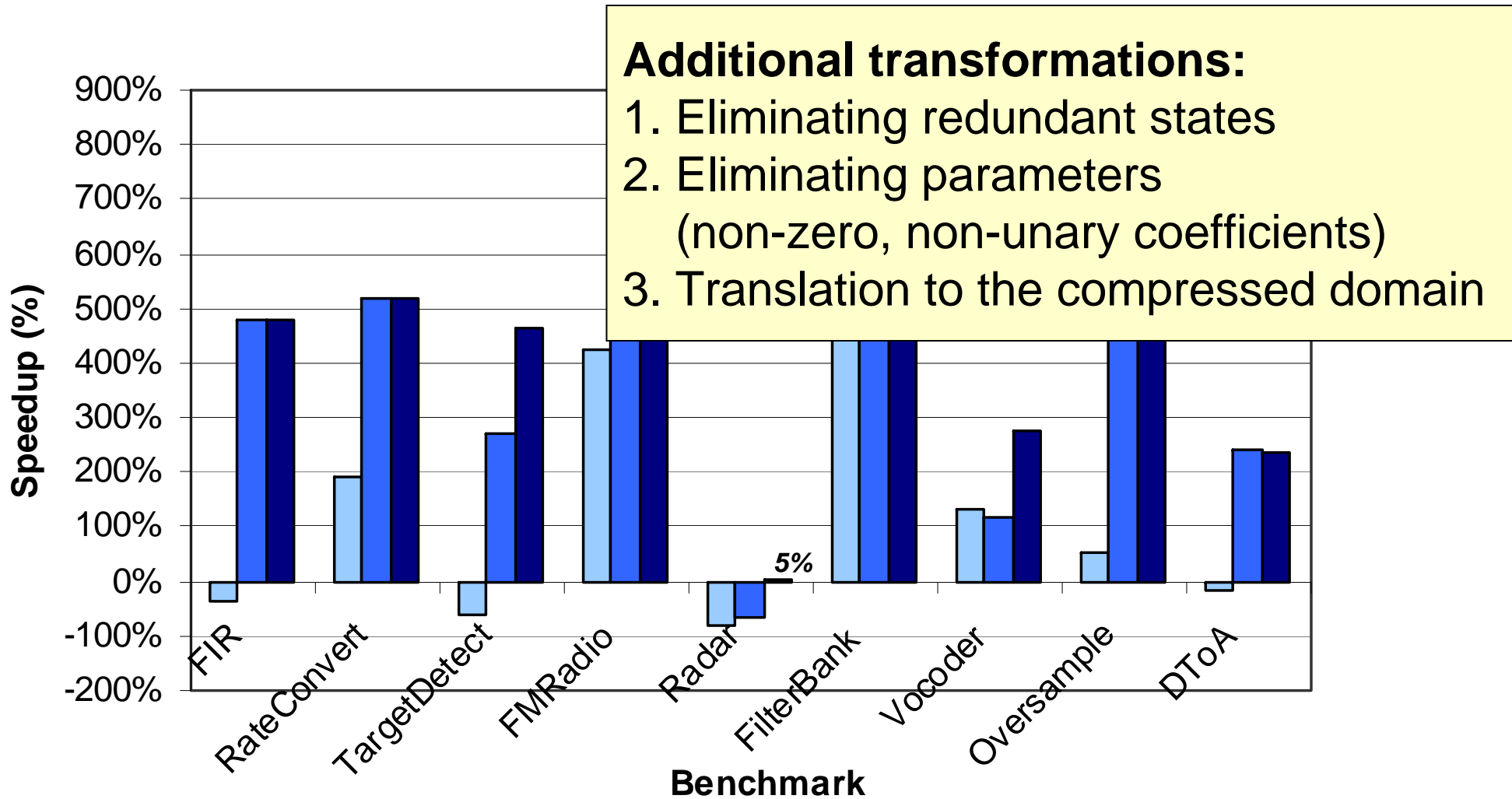


Execution Speedup



On a Pentium IV

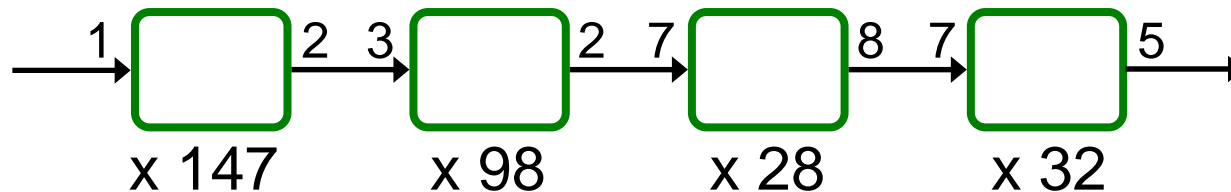
Execution Speedup



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

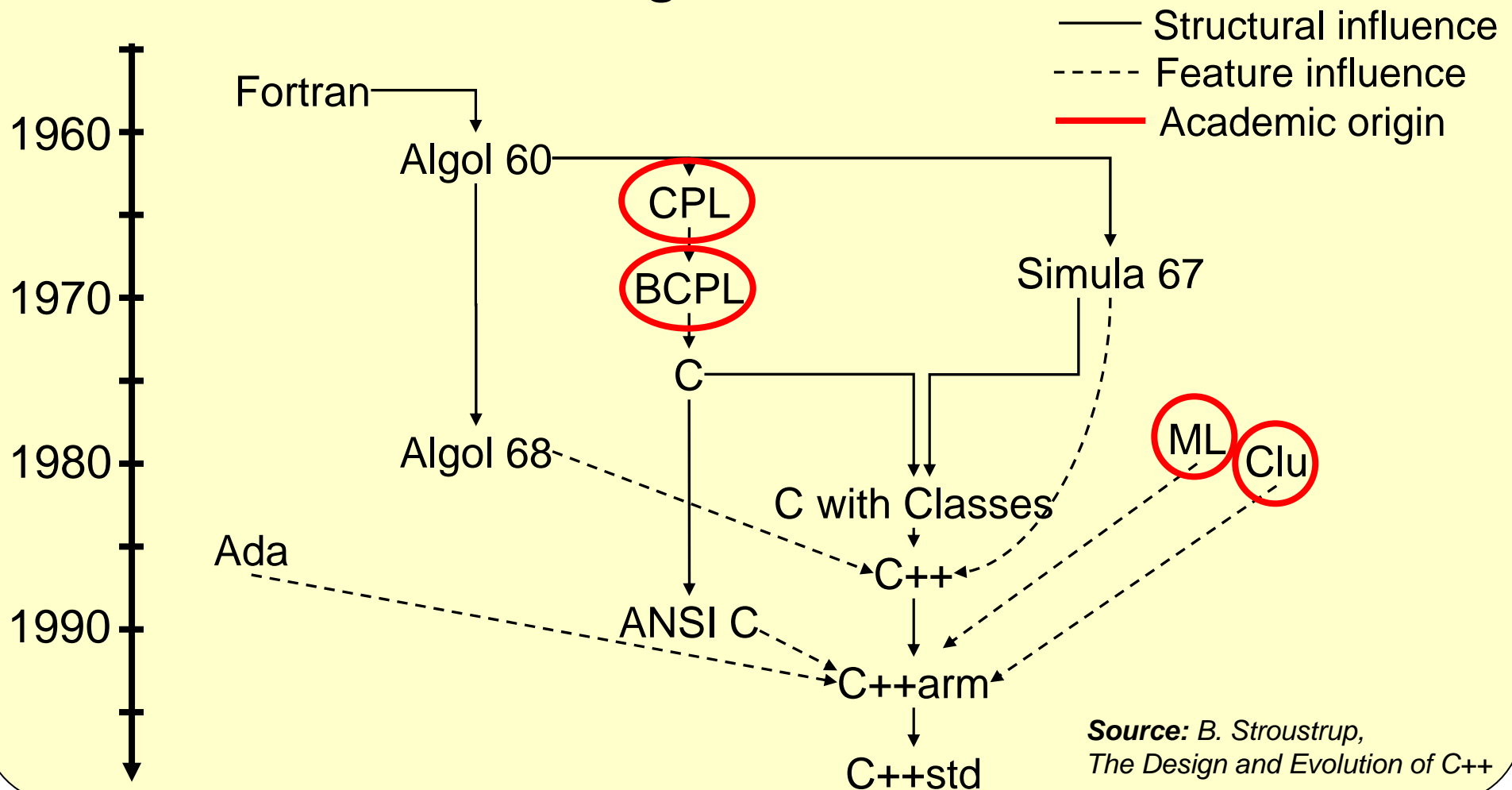
```
void>int filter SquareWave() {  
    int x = 0;  
  
    work push 1 {  
        push(x);  
        x = 1 - x;  
    }  
}
```

stateful

Future of StreamIt

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

Origins of C++

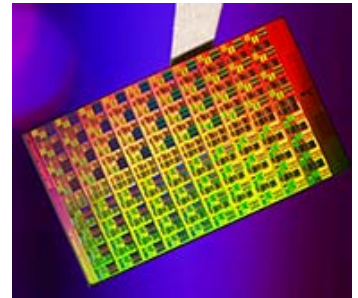


Research Trajectory

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

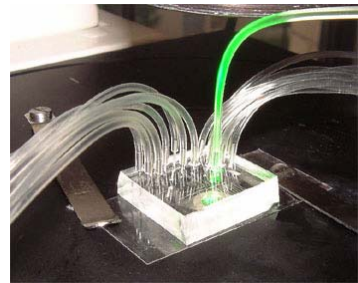
1. Languages, compilers, & tools for multicores

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

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

Acknowledgments

- **Project supervisors**

- Prof. Saman Amarasinghe
- Dr. Rodric Rabbah

- **Contributors to this talk**

- Michael I. Gordon (Ph.D. Candidate) – *leads StreamIt backend efforts*
- Andrew A. Lamb (M.Eng) – *led linear optimizations*
- Sitij Agrawal (M.Eng) – *led statespace optimizations*

- **Compiler developers**

- Kunal Agrawal
- Jasper Lin
- Janis Sermulins
- Allyn Dimock
- Michal Karczmarek
- Phil Sung
- Qiuyuan Jimmy Li
- David Maze
- David Zhang

- **Application developers**

- Basier Aziz
- Shirley Fung
- Ali Meli
- Matthew Brown
- Hank Hoffmann
- Satish Ramaswamy
- Matthew Drake
- Chris Leger
- Jeremy Wong

- **User interface developers**

- Kimberly Kuo
- Juan Reyes