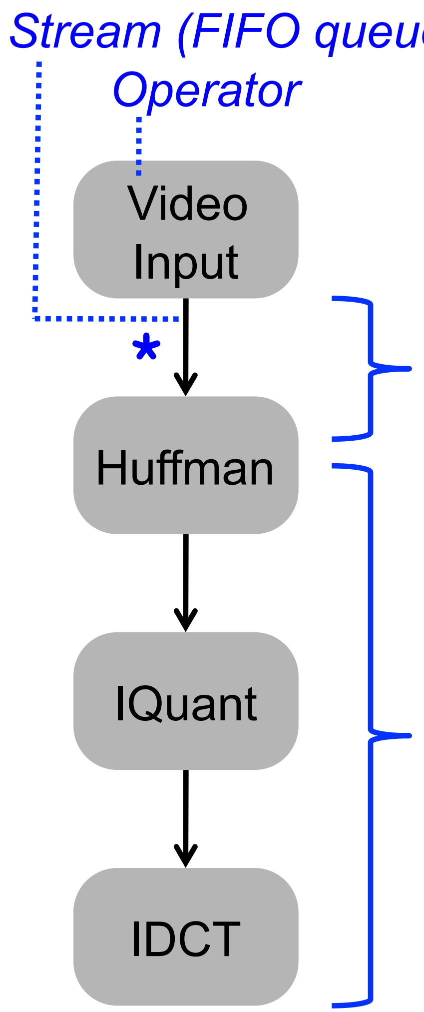
Dynamic Expressivity with Static Optimization for Streaming Languages

Robert Soulé	Michael I. Gordon	Saman Amarasinghe	Robert Grimm	Martin Hirzel
Cornell	MIT	MIT	NYU	IBM

DEBS 2013

Problem



"Rate" = number of queue pushes/pops per operator firing

Dynamic rate (varies at runtime)

➔ *Requires dynamic expressivity*

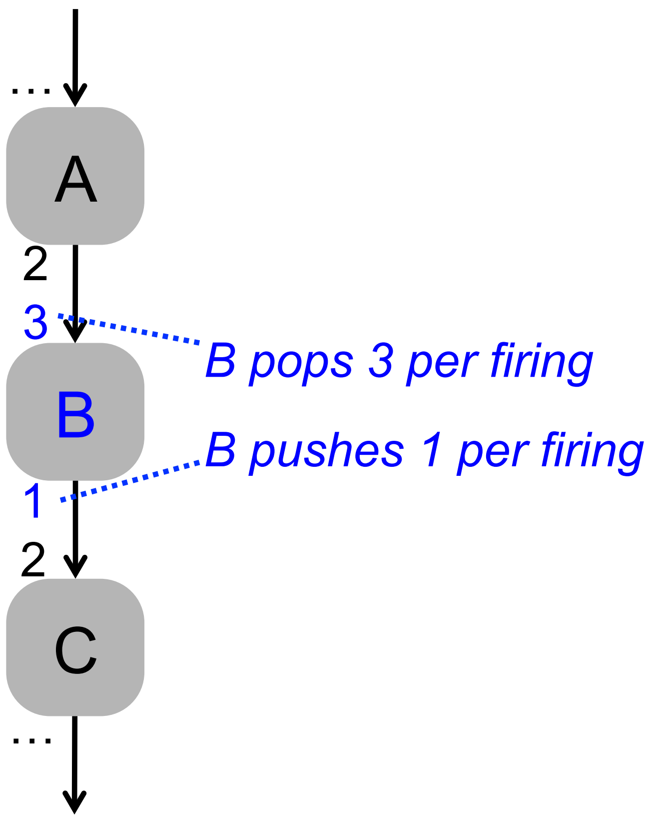
Static rate (known at compile time)

➔ *Enables static optimization*

How to get both?

Observation: applications are "mostly static"
(Thies, Amarasinghe [PACT 2010])

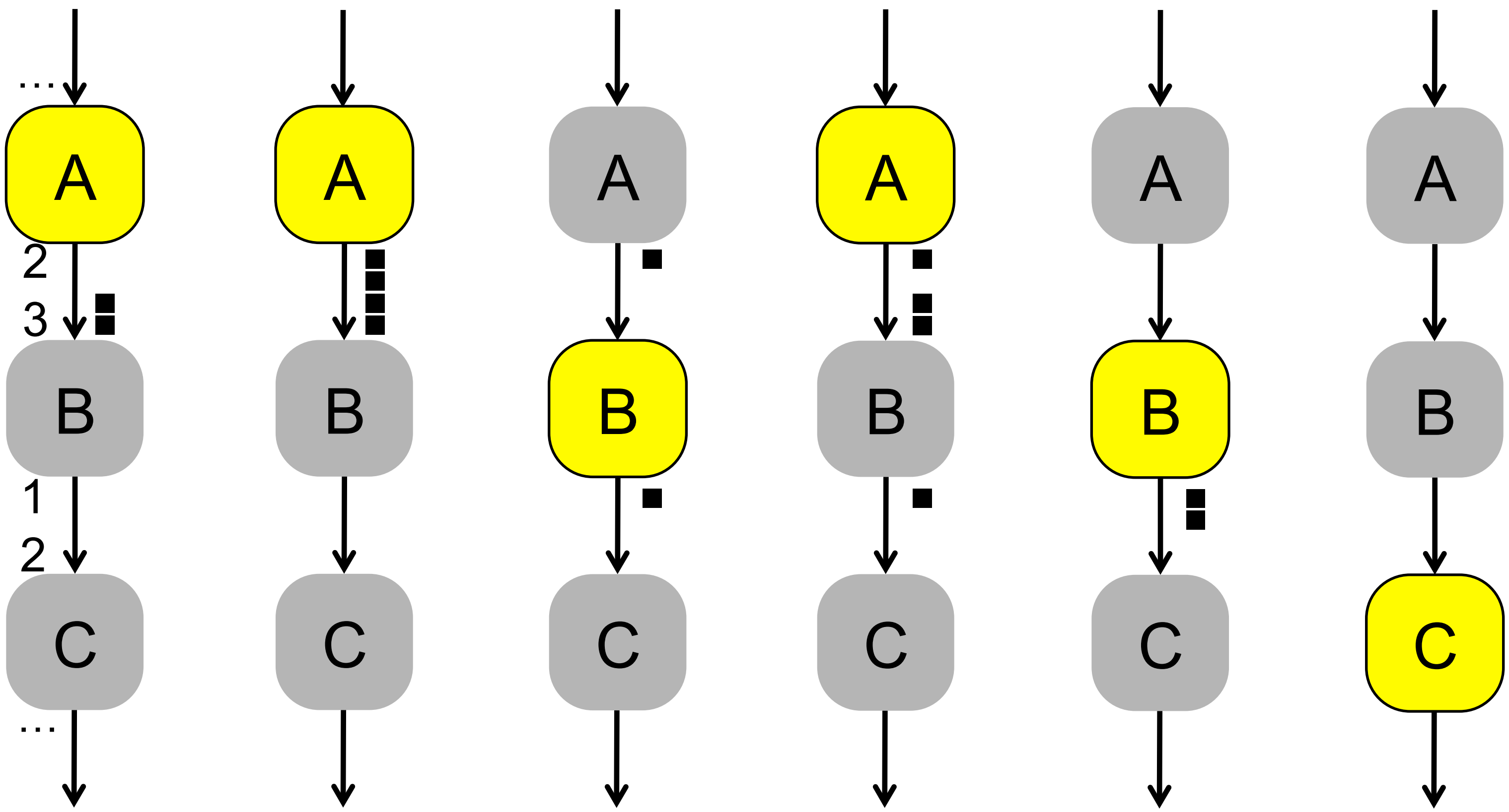
StreamIt, a Streaming Language Designed for Static Optimization




```
float->float pipeline ABC {
  add float->float filter A() {
    work pop … push 2
      { … }
  }
  add float->float filter B() {
    work pop 3 push 1
      { … }
  }
  add float->float filter C() {
    work pop 2 push …
      { … }
  }
}
```

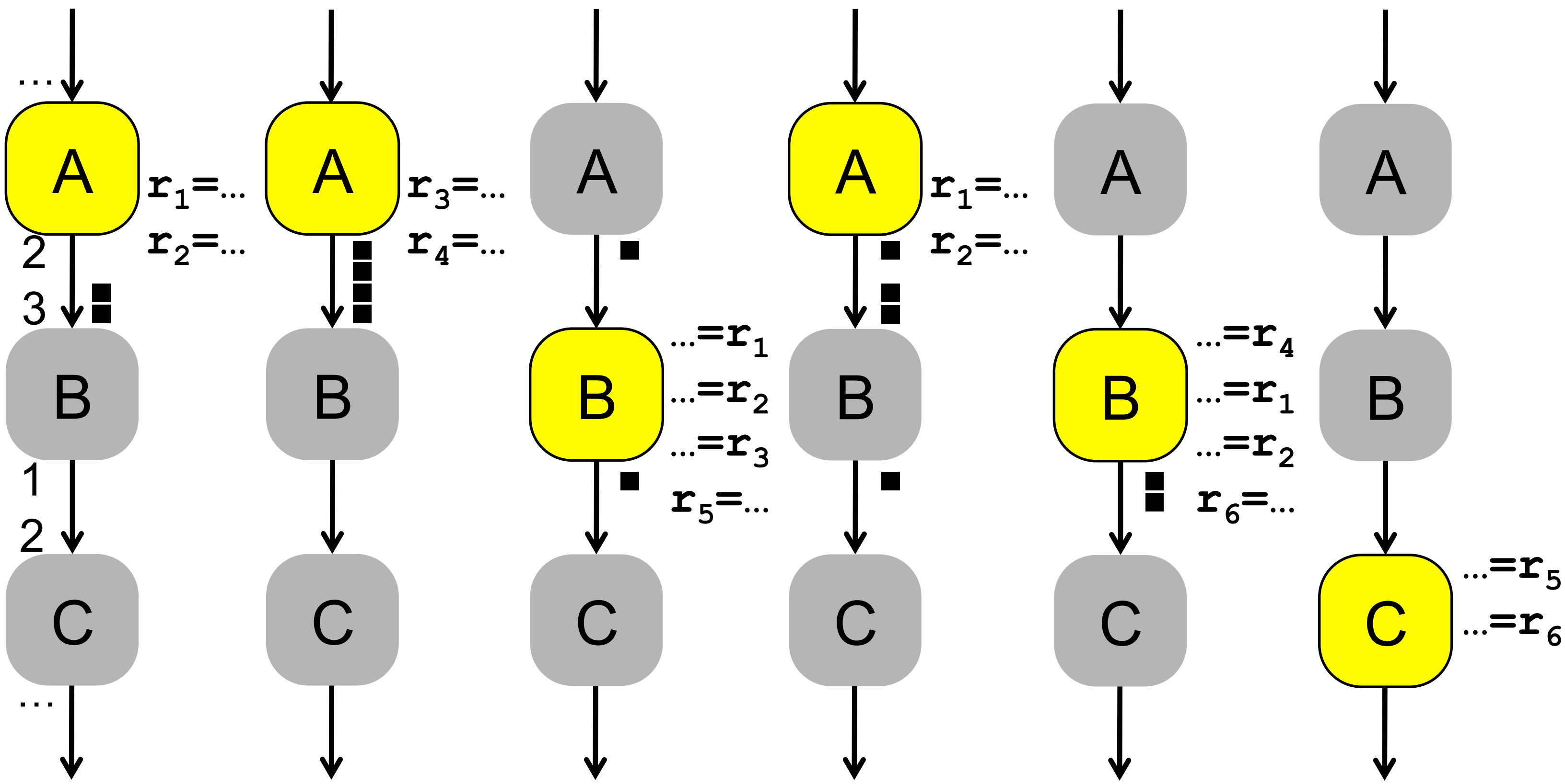
➔ ***Statically known push/pop rates (SDF = Synchronous Dataflow)***

SDF Steady-State Schedule



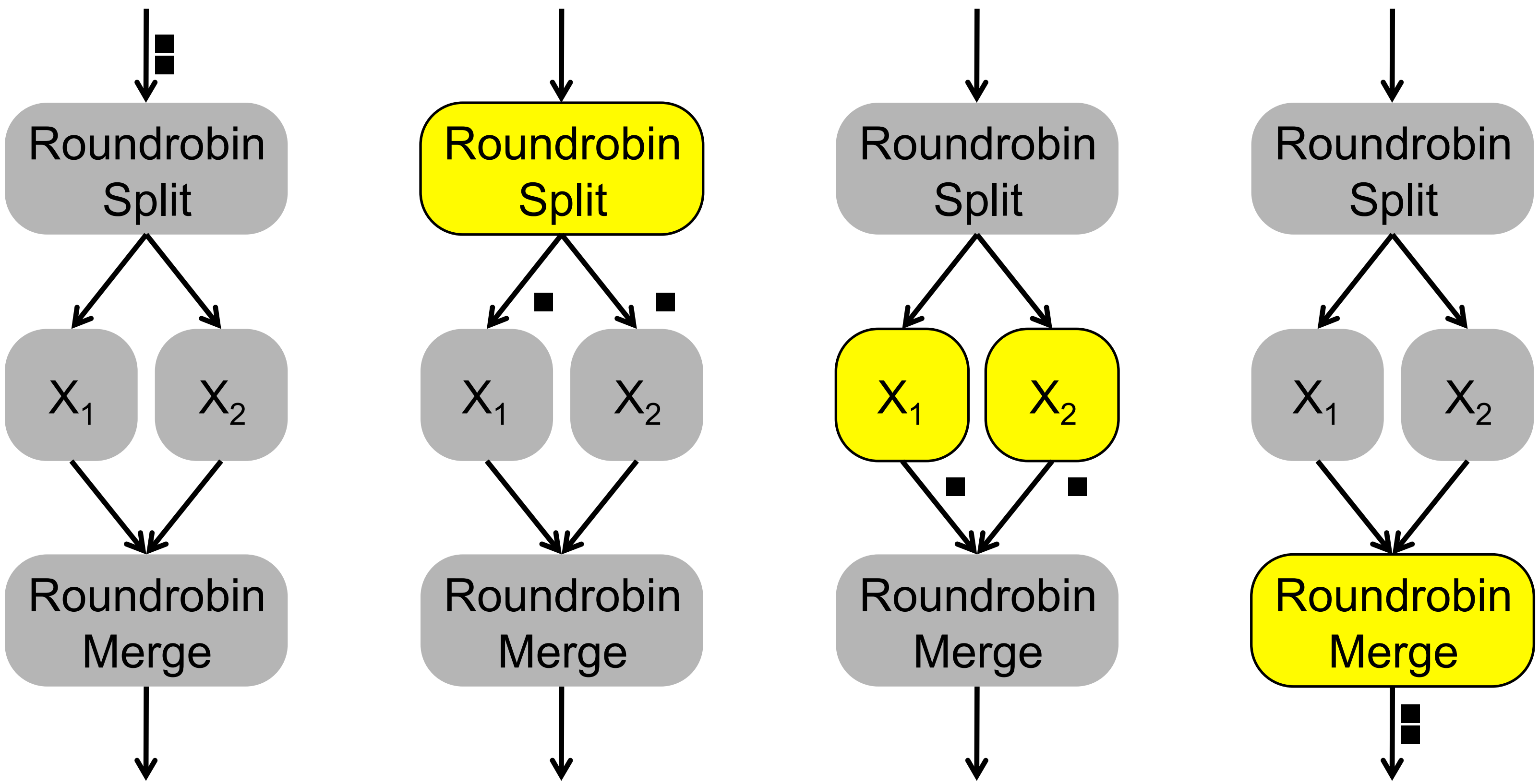
➔ ***Statically known firing order and FIFO queue sizes***

Scalarization



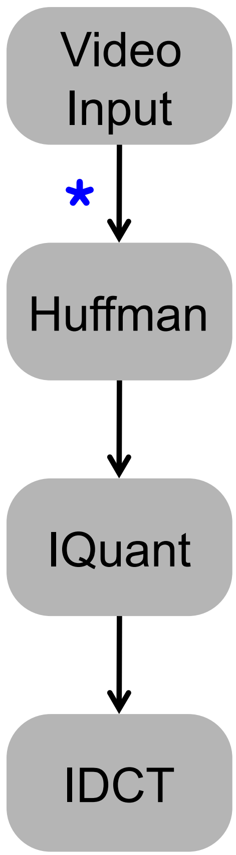
➔ Implement FIFO queue via local variables, or even registers
(more intricate with "peek", not shown in this talk)

Fission (Data Parallelism)



➔ ***Round-robin split and merge rely on static rates***

Dynamic Rates




```
float->float pipeline Decoder {
  add float->float filter VideoInput() {
    work pop 1 push 1
      { … }
  }
  add float->float filter Huffman() {
    work pop * push 1
      { … }
  }
  add float->float filter IQuant() {
    work pop 64 push 64
      { … }
  }
  add float->float filter IDCT() {
    work pop 8 push 8
      { … }
  }
}
```

➔ ***No more static optimization?***

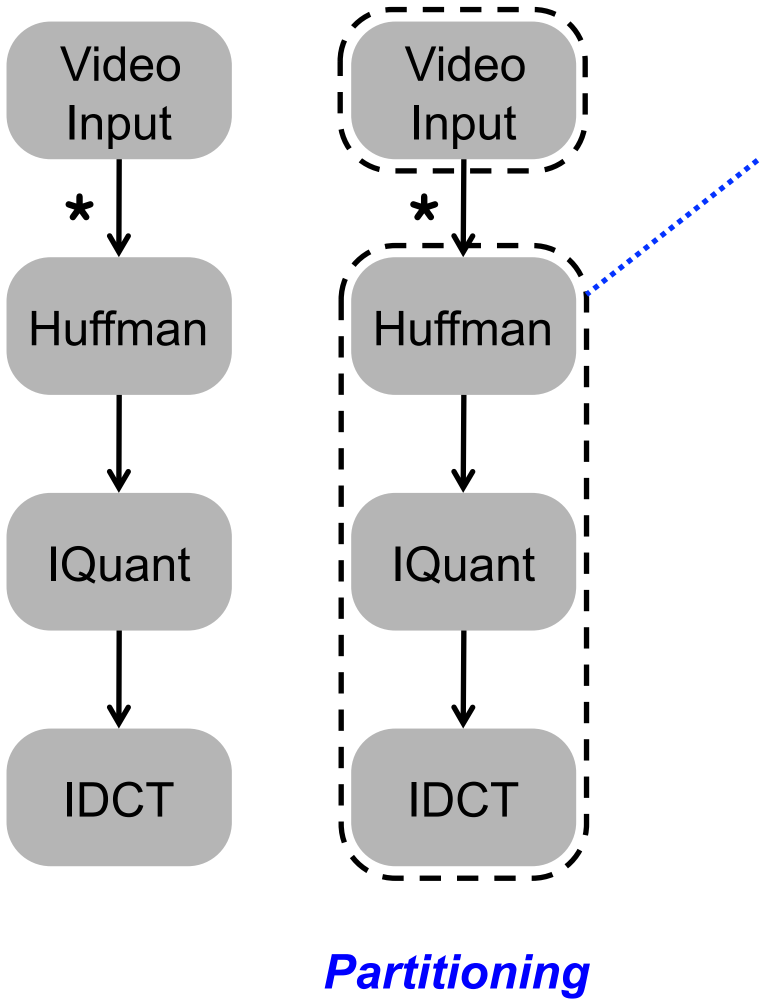
Dynamic Scheduling Approaches

Scheduling approach	Description	Representative citation
OS Thread	Each operator has its own thread	SPC, Amini et al. [DMSSP 2006]
Demand	Recruit from thread pool	Aurora, Abadi et al. [VLDBJ 2003]
No-op	Static rate, send nonce when no data	CQL, Arasu et al. [VLDBJ 2006]

Our Approach: Locally Static + Globally Dynamic

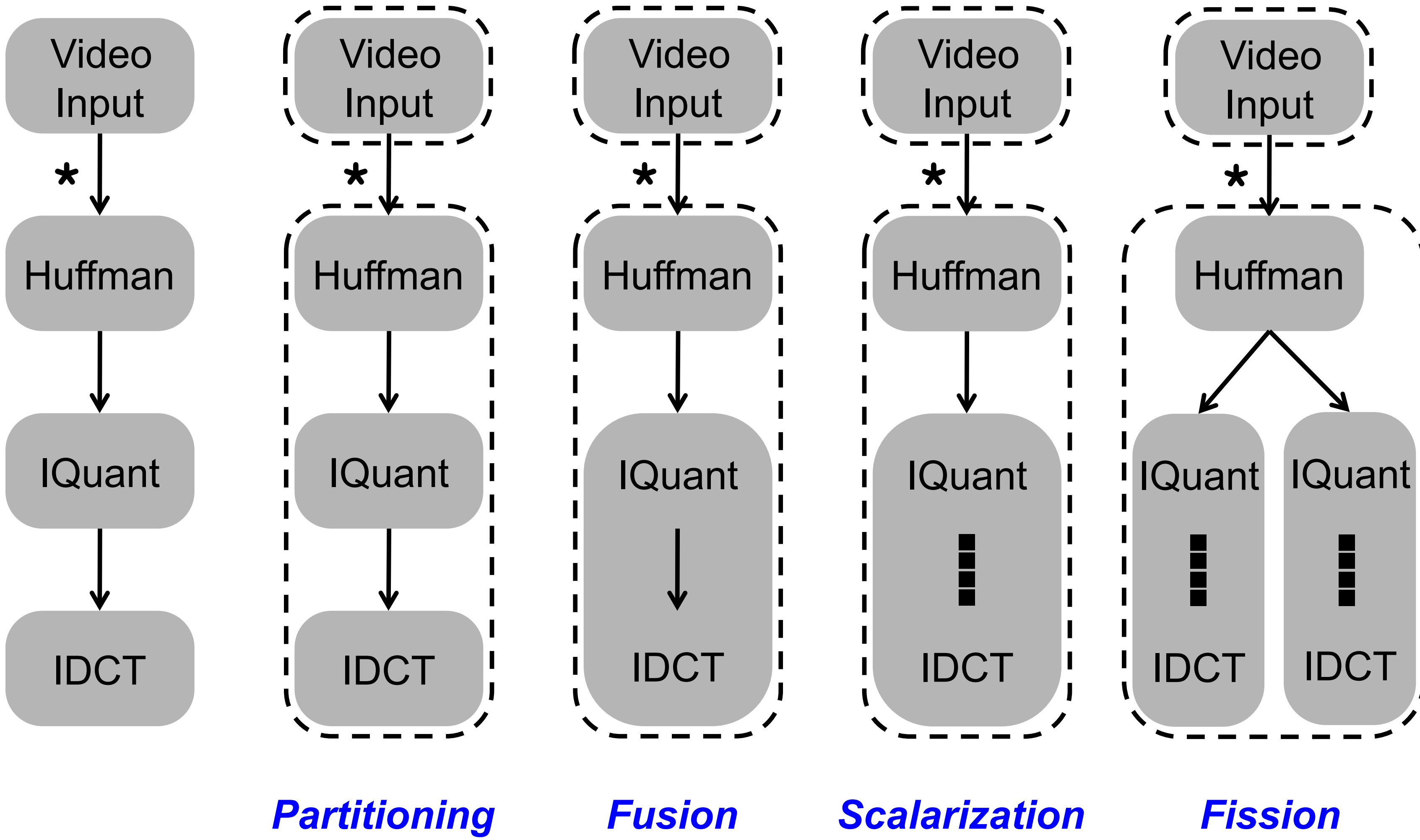
1. Partitioning into static subgraphs
2. Locally optimize static subgraphs
   - 2a. Fusion
   - 2b. Scalarization
   - 2c. Fission
3. Placement
   - 3a. Core placement
   - 3b. Thread placement
4. Globally dynamic scheduling

Partition into Static Subgraphs

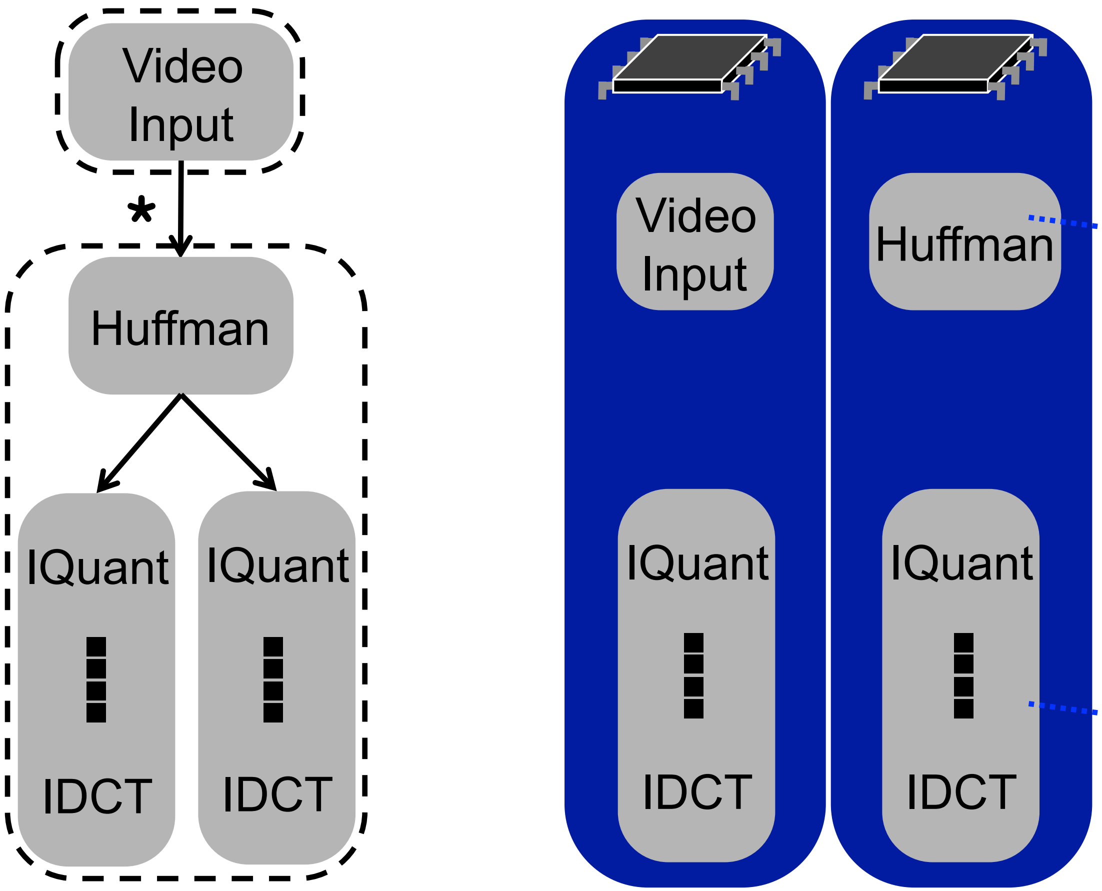


Static subgraph:
Weakly connected component after deleting dynamic edges.

Locally Optimize Static Subgraphs



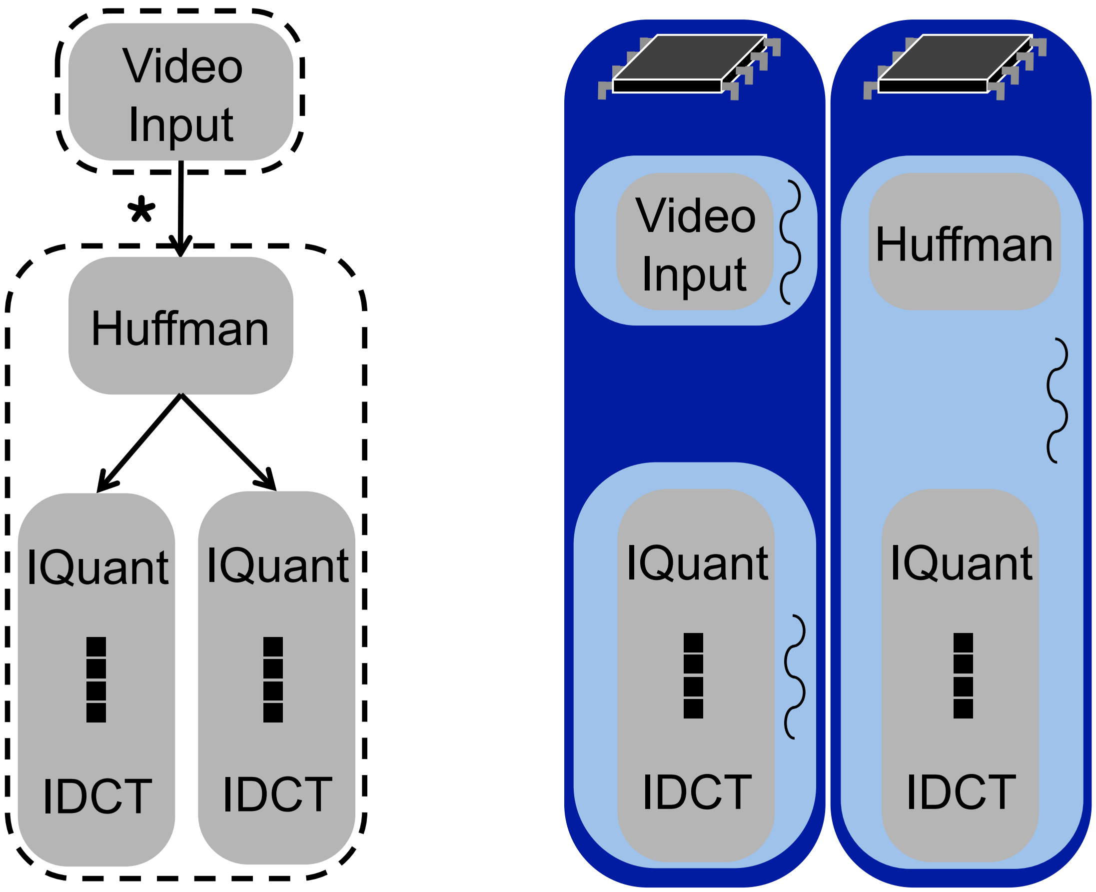
Core Placement



Static weight estimate and greedy bin-packing

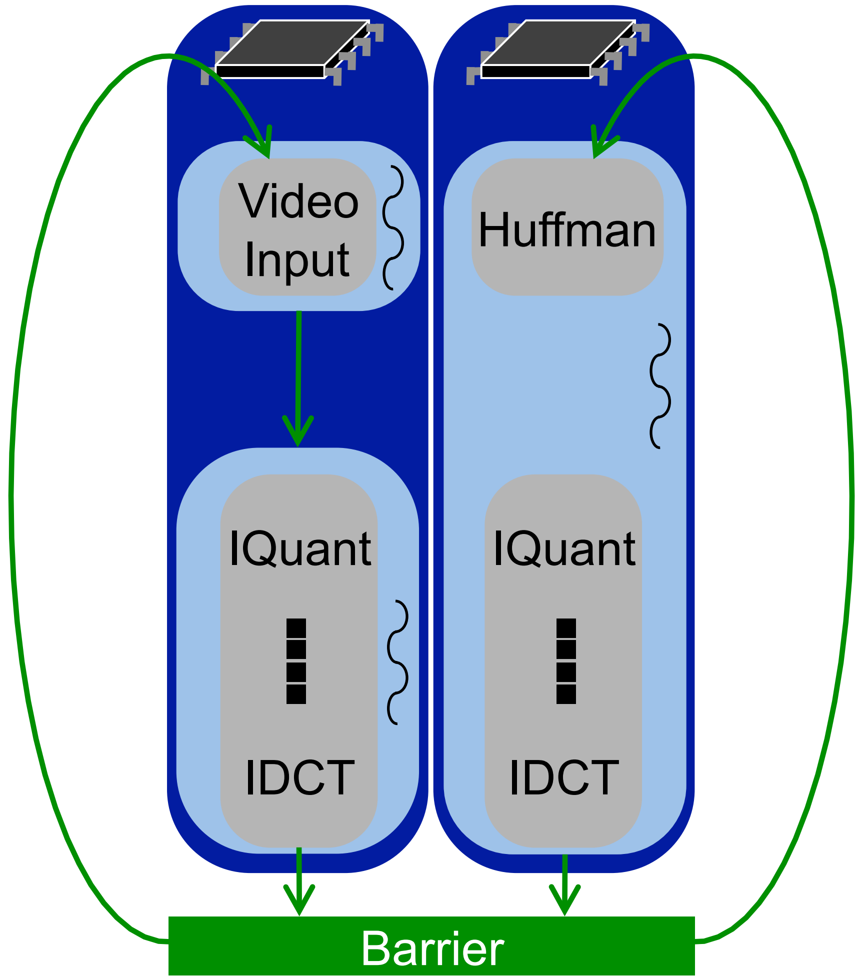
Place fission replicas on all cores

Thread Placement

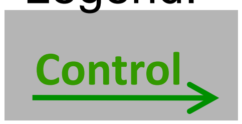


One pinned thread per static subgraph per core (must be able to suspend dynamic reader when no input)

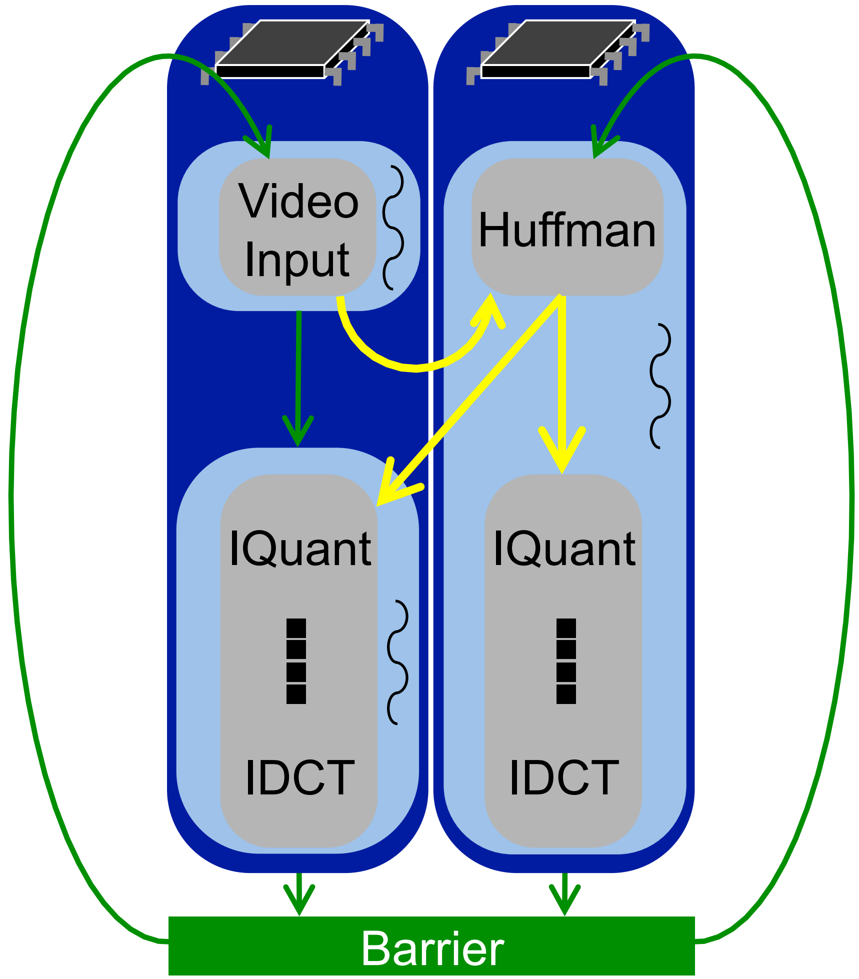
Dynamic Scheduling



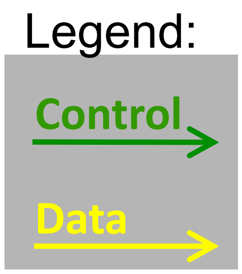
Use condition variables for hand-off to successor



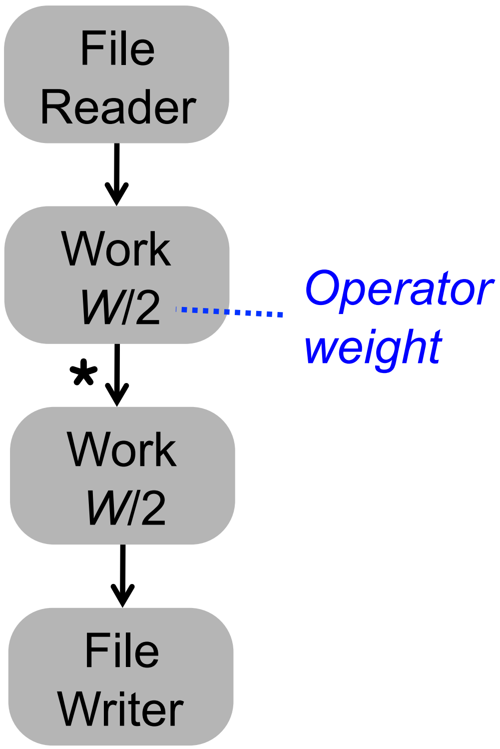
Data Pipelining

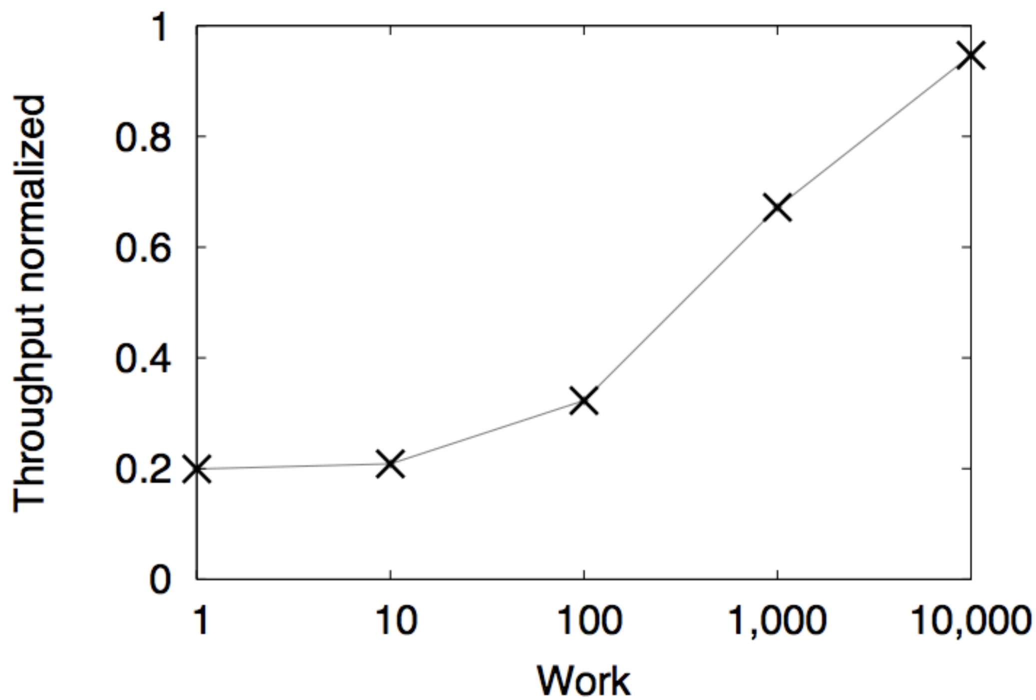


Use buffer for pipeline parallelism



Dynamic vs. Static Performance



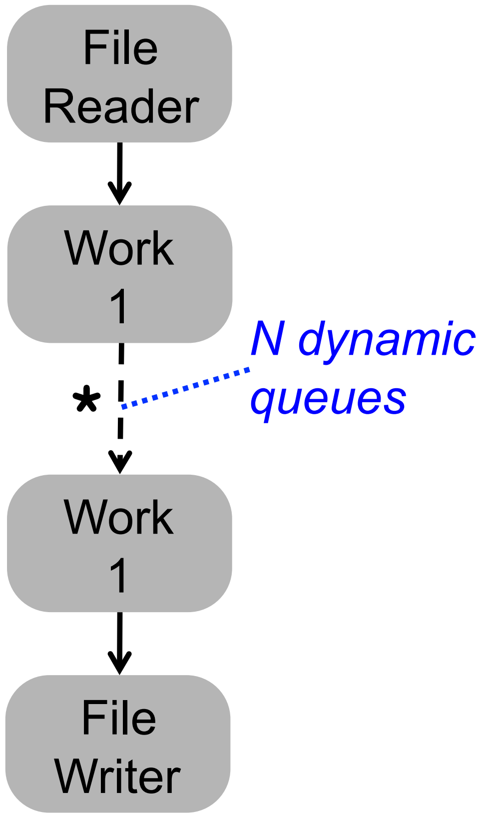


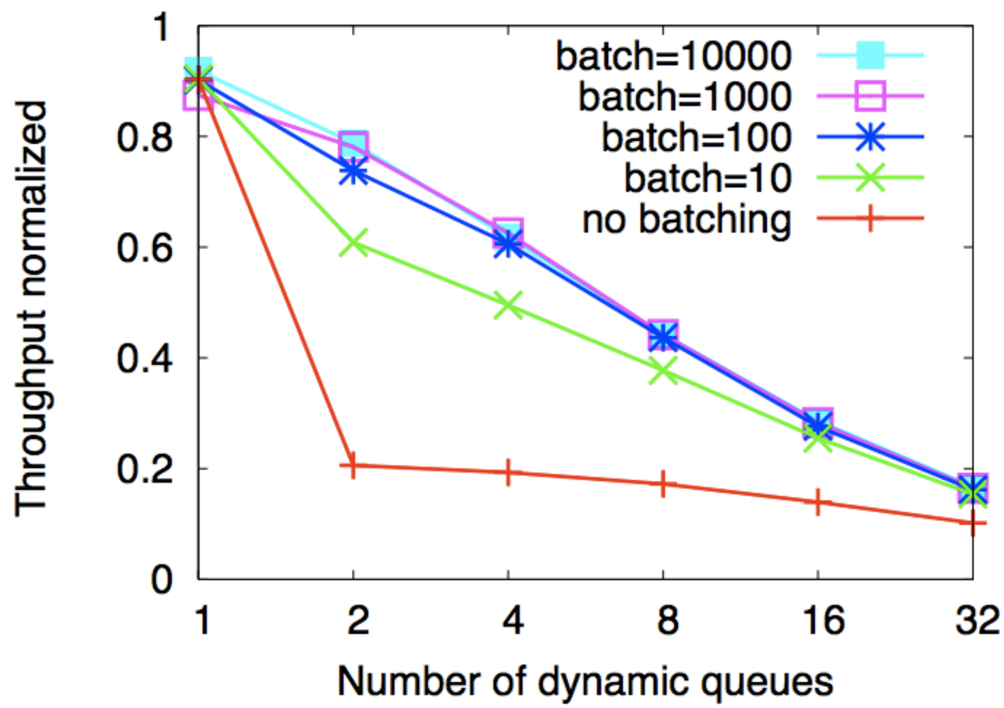
➔ Close enough for heavy operators, but what about light operators?

Amortizing the Thread Switching Overhead

1. Partitioning into static subgraphs
2. Locally optimize static subgraphs
    - 2a. Fusion
    - 2b. Scalarization
    - 2c. Fission
    - 2d. **Batching**
3. Placement
    - 3a. Core placement
    - 3b. Thread placement
4. Globally dynamic scheduling

Benefit of Batching





➔ Amortize thread switching overhead without heavy operators

Our vs. Other Dynamic Schedulers Performance

Scheduling approach	Experiment	Result
OS Thread	32 threads, 1 core, work 31 per operator	Our scheduler is 10x faster
Demand	Huffman encoder and decoder	Our scheduler is 1.2x faster
No-op	2 programs: VWAP and predicate filter	Our scheduler is 5.1x and 4.9x faster

➔ ***Our scheduler was faster in all cases (see paper for details)***

Conclusions

- Static streaming languages such as StreamIt enable powerful optimizations
- But many real-world applications require dynamic rates
- We extend the StreamIt optimizing compiler to handle dynamic rates