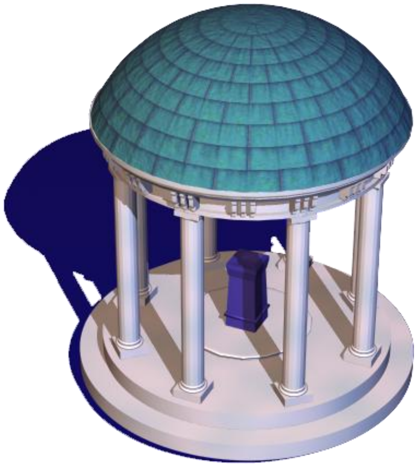
Multi-Token Resource Sharing for Pipelined Asynchronous Systems

John Hansen and Montek Singh

Dept. of Computer Science
University of North Carolina
Chapel Hill, NC, USA

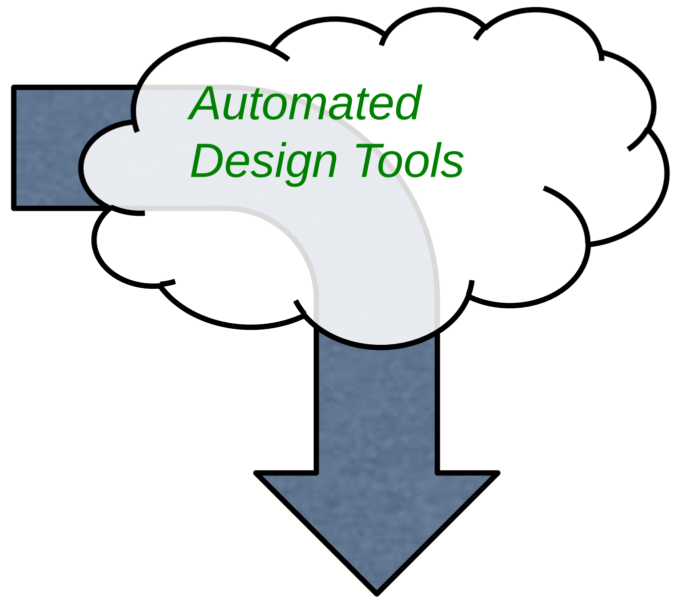


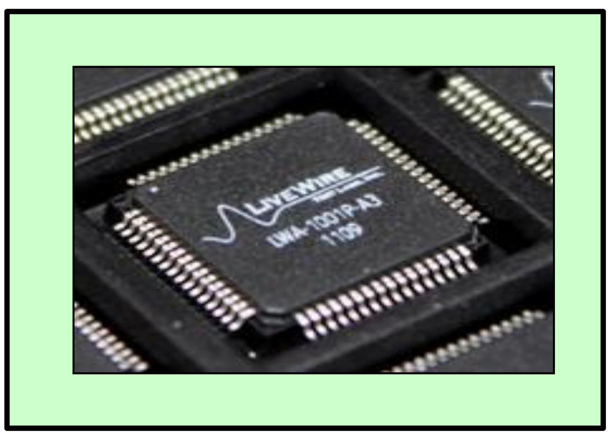
"Grand" Vision

Asynchronous high-level synthesis:

```
&MAIN :  main  proc (IN? chan <<byte, byte, … ).
  begin
    a, b, c, d, e, f, g, h, i, j, k :  var byte
  |
    forever do
      IN?<<a, b, c, d, e, f>>;
      g := a * b;
      h := c * d;
      i := e * f;
      j := g + h;
      k := i * j;
      OUT!k
    od
  end
```

Convert high-level specification…

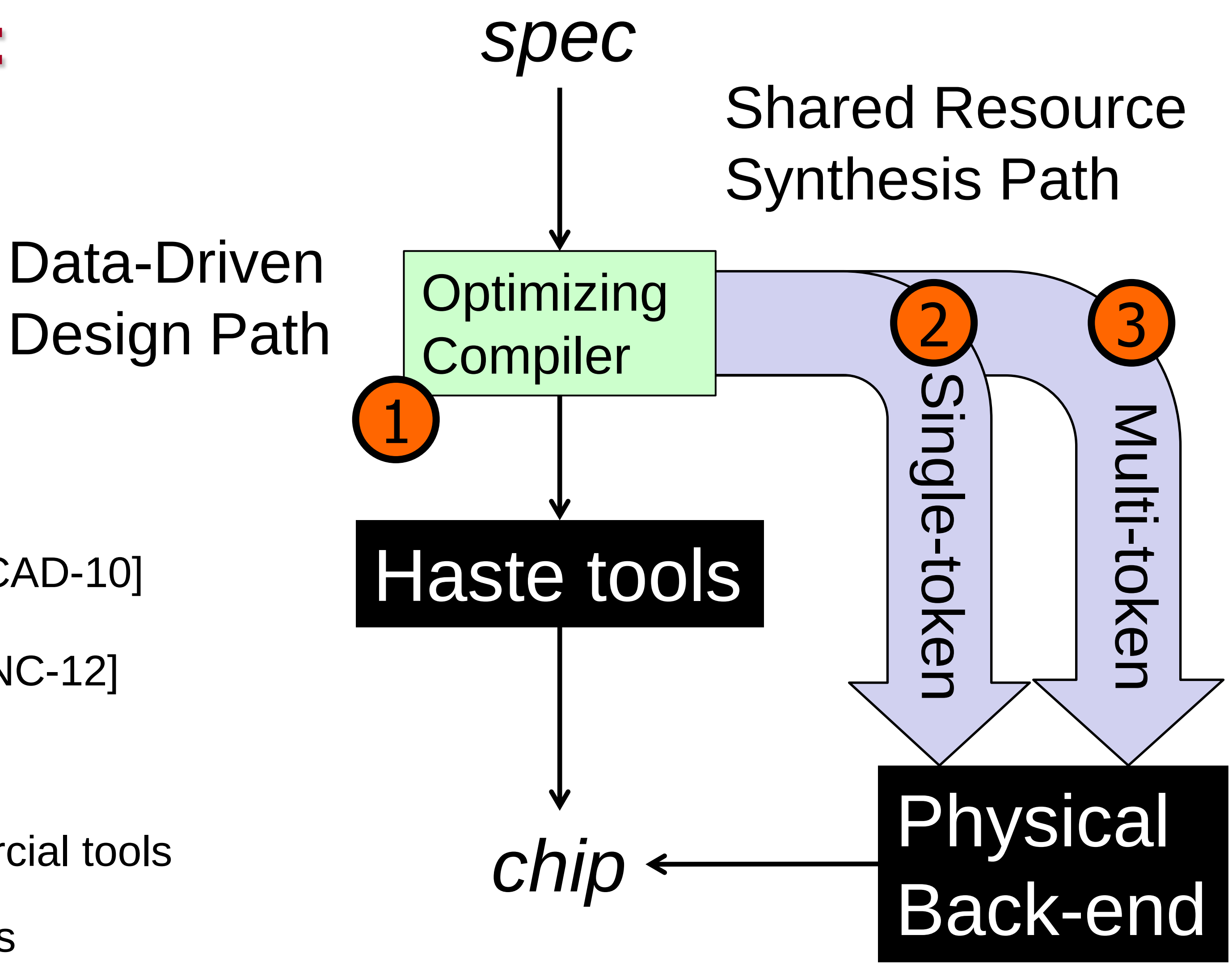




… into custom VLSI chip

Our Overall Design Flow

Several paths:



1 [ASYNC-08]

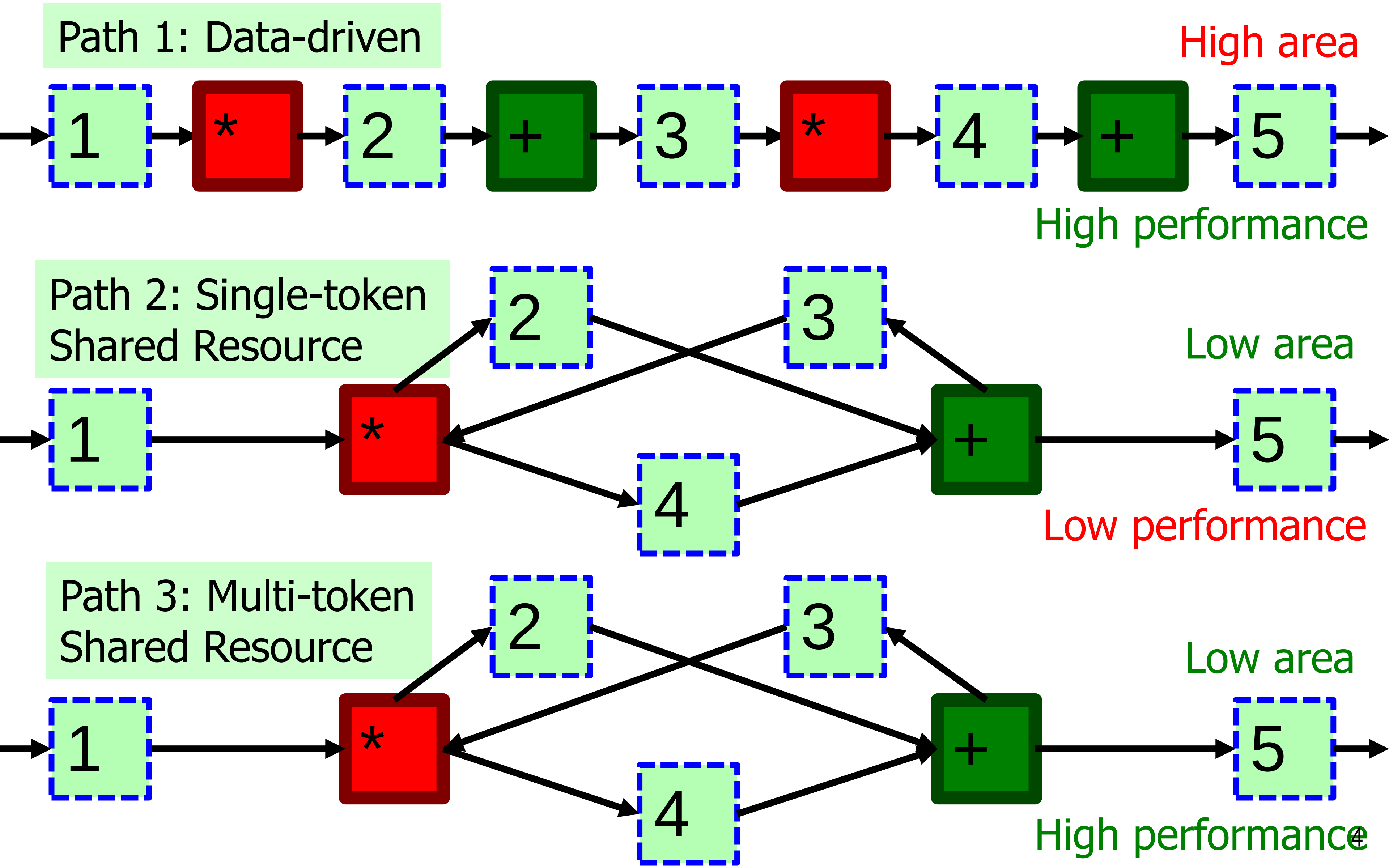
2 [ASYNC-10, ICCAD-10]

3 [DATE-12, ASYNC-12]

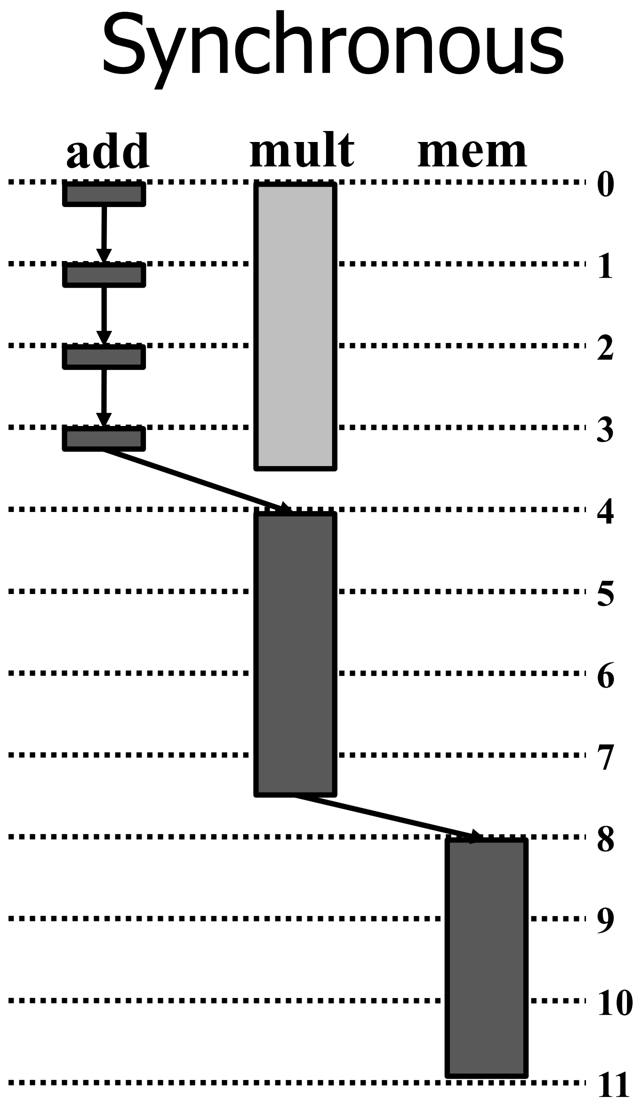
Existing commercial tools

Our contributions

Comparison of Design Paths

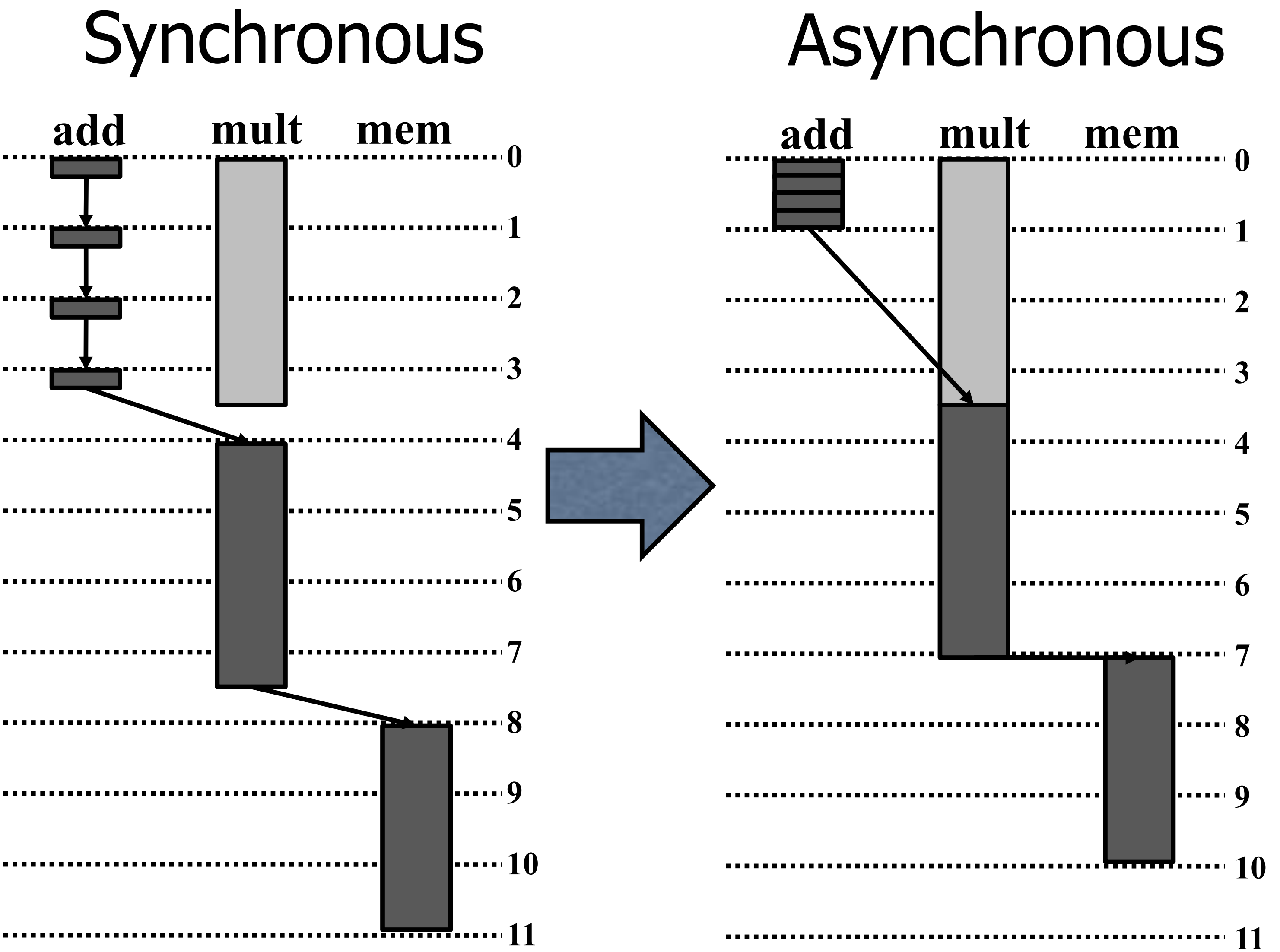


Sync vs. Async Scheduling



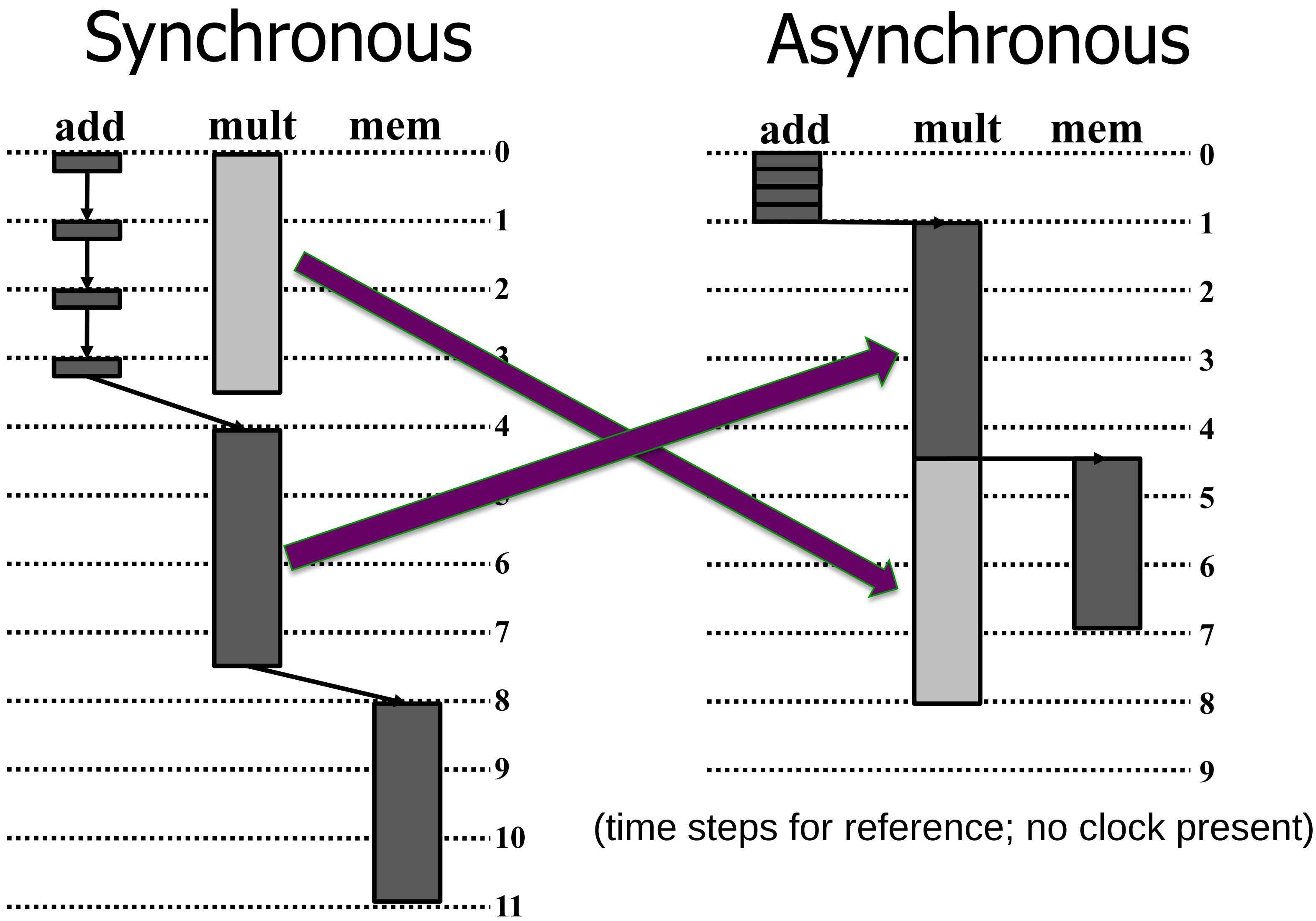
* Basic synchronous scheduling approach
  - Operations can only be scheduled on clock edges
  - Critical path in dark grey
  - Best solution shown

Sync vs. Async Scheduling

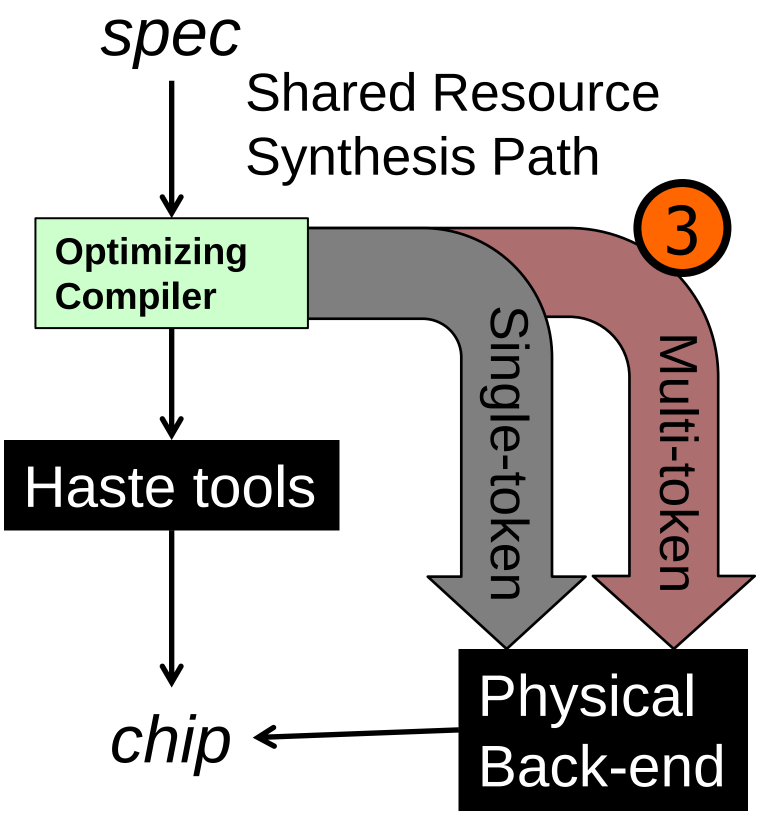


(time steps for reference; no clock present)

Sync vs. Async Scheduling



Multi-Token Synthesis



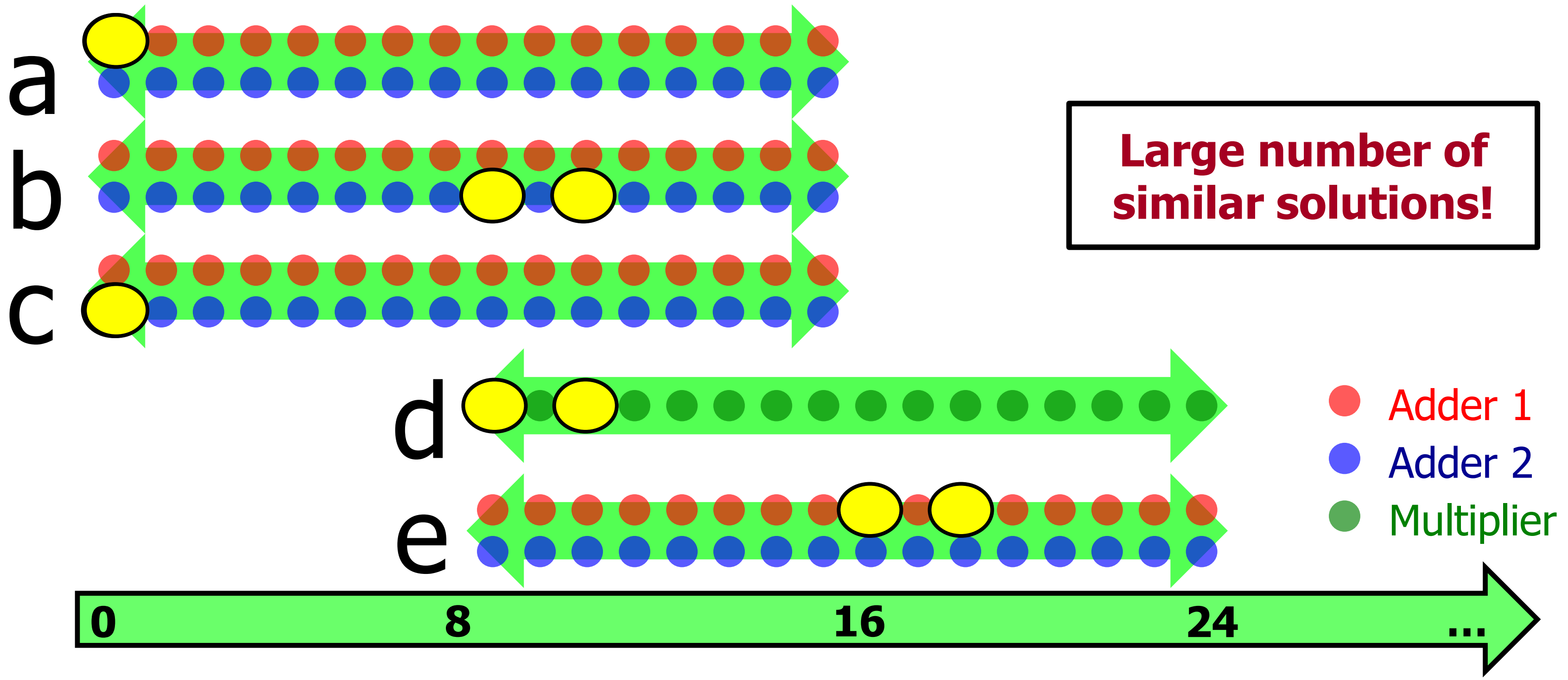
- Multi-Token
  - multiple concurrent computations
  - pipelined
- Unsolved problem
  - even for synchronous systems
- Best of both worlds
  - pipelined and shared-resource
  - high performance, low area
  - explore whole spectrum in-between!

Outline

- ➔ Previous Work
- Our Approach
  - Basic approach [DATE 2012]
  - Hierarchical approach [this paper]
- Results & Conclusions

Existing Sync. Solutions: Poor Match

- Synchronous approaches:
  - SPARK, AutoPilot/AutoESL, GAUT, …
  - Large search space for ILP
    - Each time step for each <operation, func unit> → distinct variable
  - Our idea: Solve for <u>relative ordering</u> of events, not timing



Asynchronous Approaches

- Syntax-directed synthesis tools (Haste/Balsa)
  - No automated resource sharing
    - "what you write is what you get"

- Resource sharing: Single-token
  - Many approaches are not purely async
    - adapt discrete time methods to async
    - E.g.: [Nielsen 2005, Saito 2006, …]

  - Hansen/Singh [ASYNC-10] [ICCAD-10]
    - first exact purely asynchronous solution
    - based on relative order, not absolute timing

Multi-Token Synthesis: Challenges

- More challenging than single-token
  - mix-and-match operations from successive tokens
    - much larger search space
    - how many tokens?
  - more memory elements (buffers)

- General problem unsolved
  - Given: dataflow graph, throughput target
  - Compute: resource schedule that minimizes area
    - over all possible resource allocations
    - over all possible buffer insertions
    - over all possible token counts

Multi-Token Synthesis: Prior Work

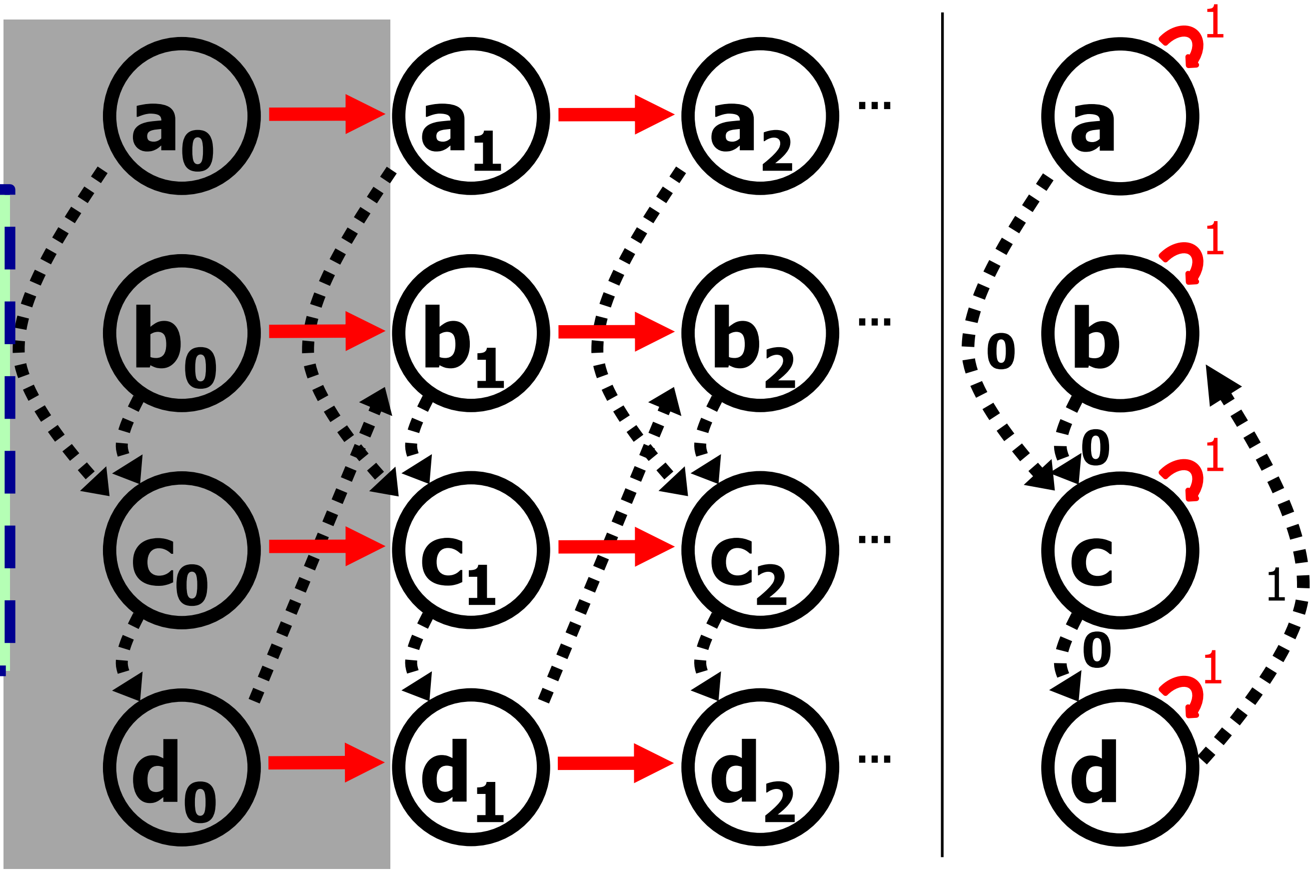
- No prior optimal method for multi-token scheduling
  - existing approaches solve only part of the problem
    - [Beerel 2005] requires token count, discrete time
  - others heuristic, share resources where straightforward
    - not targeting exact area-minimization problem [Spark 2004, Cadence 2011]

Outline

- Previous Work
- Our Approach
  - Part 1: Basic approach [DATE 2012]
  - Part 2: Hierarchical approach [this paper]
- Results & Conclusions

Review: Dependence Graphs

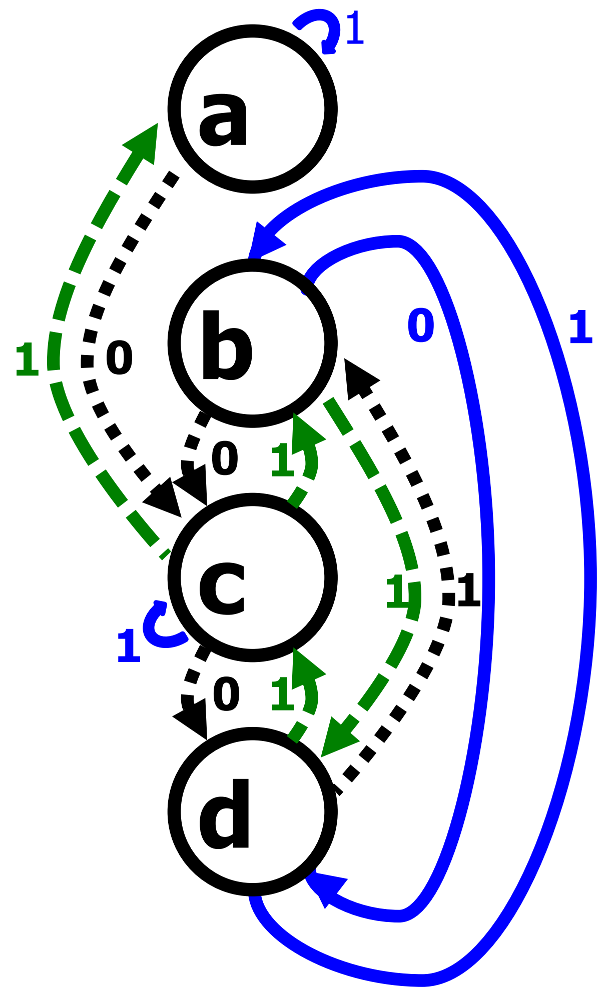
```
while(true){
        a=read();
        b=3*d;
        c=a+b;
        d=c*0.25;
}
```



Folded Dependence Graph:
Encodes dependence constraints across iterations

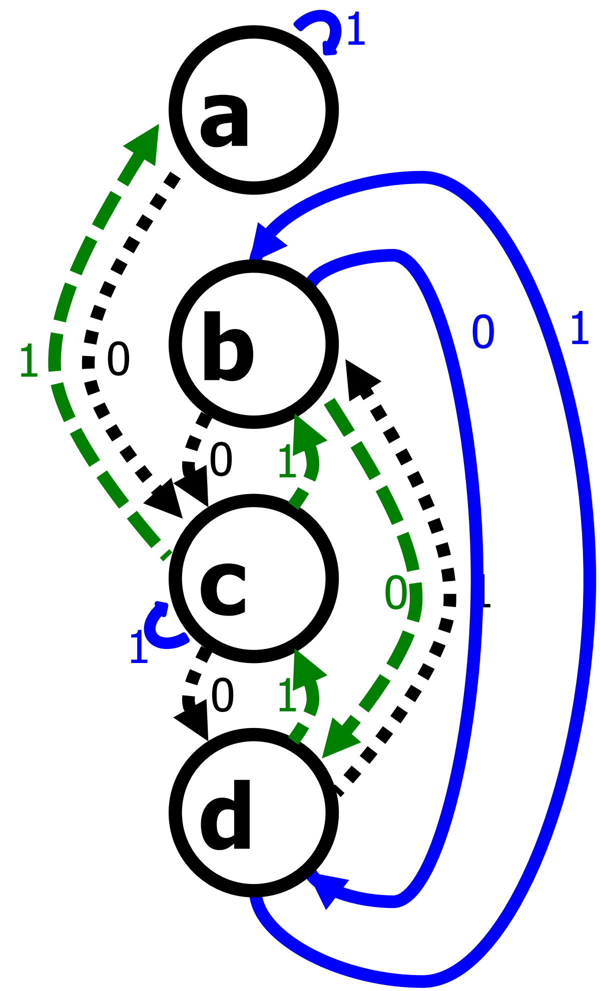
Novel Graphical Model [DATE 2012]

- Three types of arcs
  - data arcs (RAW)
  - reverse arcs (WAR)
  - resource arcs
- Arc properties:
  - weight = difference in iteration count
  - delay = min time elapsed
- Can directly infer the following:
  - resource allocation and schedule
  - number of pipeline buffers
  - performance / cycle time
  - number of tokens



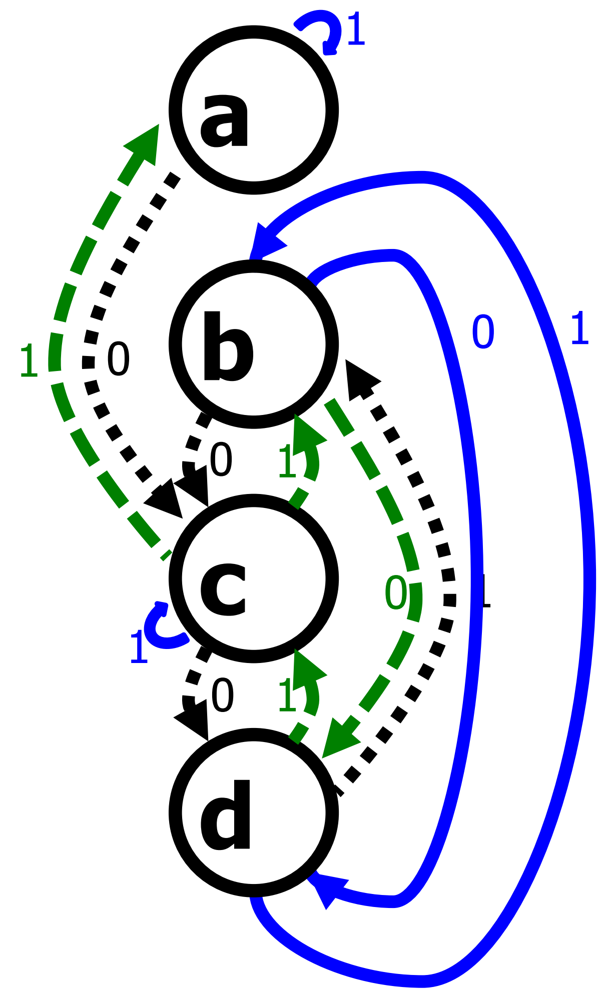
Expressiveness of Graphical Model

- Encodes all of the following:
  - What is the schedule for a resource?
    - Determined by placement and weight of resource arcs
  - How many resources?
    - number of resource cycles
  - How many pipeline stages?
    - Σ weights on data and reverse arcs
  - What is the performance?
    - Cycle metric: Determined by the weight and delay of every cycle in the graph (Σ delays / Σ weights)



Expressiveness of Graphical Model

- Legality constraints:
  - Weight of each cycle > 0
    - avoids deadlock
  - Weight of each resource cycle = 1
    - single-stride schedule
  - Weight of data arcs >= 0
    - dependencies go forward in time

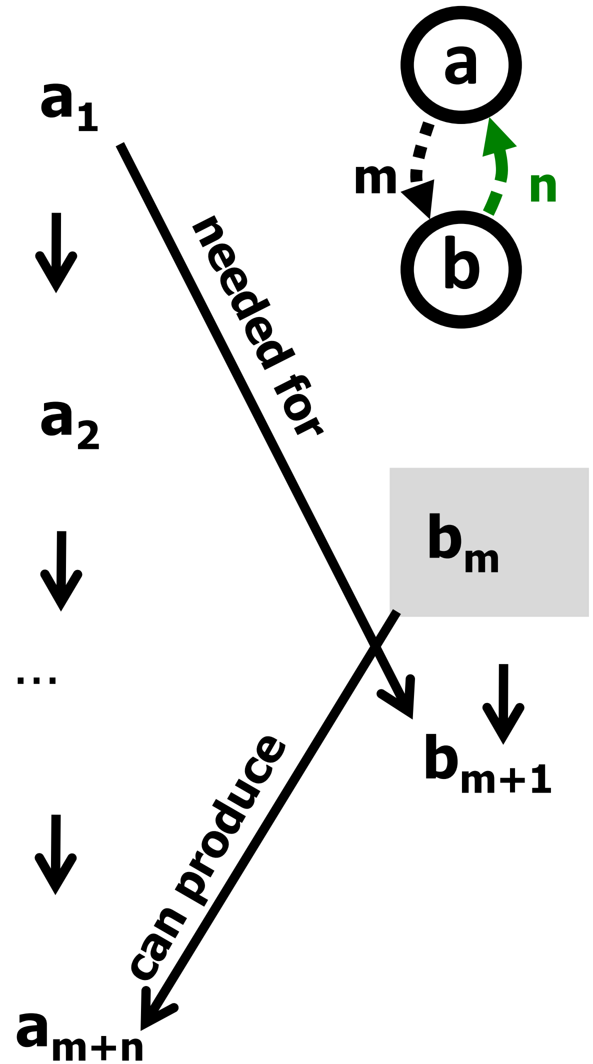


- Goal:
  - Find the lowest-area schedule that meets legality constraints and performance target

Graphical Model: Buffering

Buffers needed for 2 reasons:

1. WAR requirements
   - reverse arcs model WAR
   - multiple values for ***a*** may be live
   - must buffer all waiting to be consumed
   - e.g.: *m+n* buffers needed for ***a***

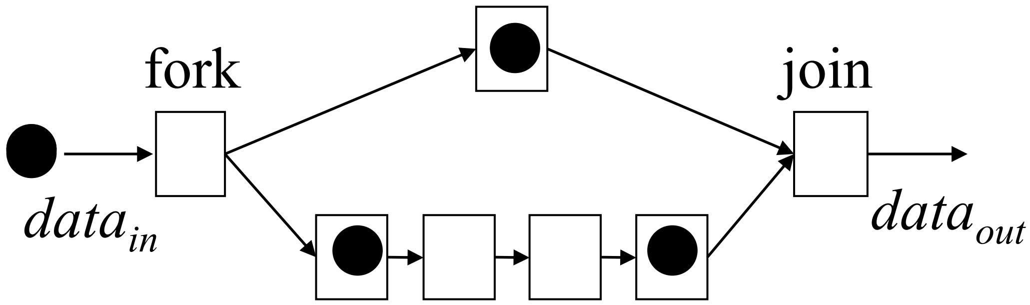


> ***Theorem:***
> **# buffers = weight of data arc +**
> **weight of reverse arc**

Graphical Model: Buffering

Buffers needed for 2 reasons:

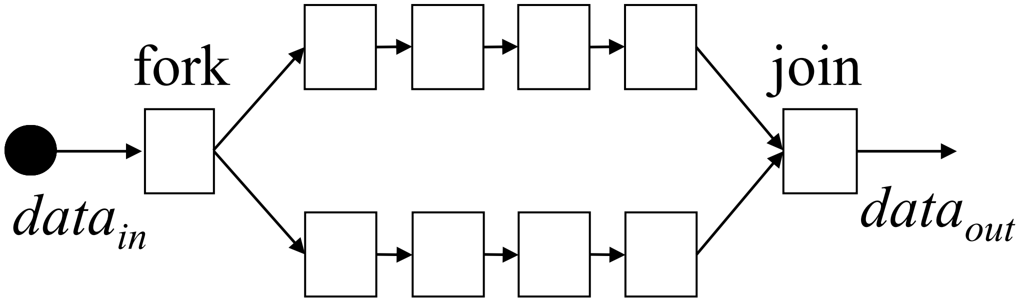
2. Speed requirements
   - too few buffers can cause slowdown
   - "slack mismatch" in reconvergent paths



Graphical Model: Buffering

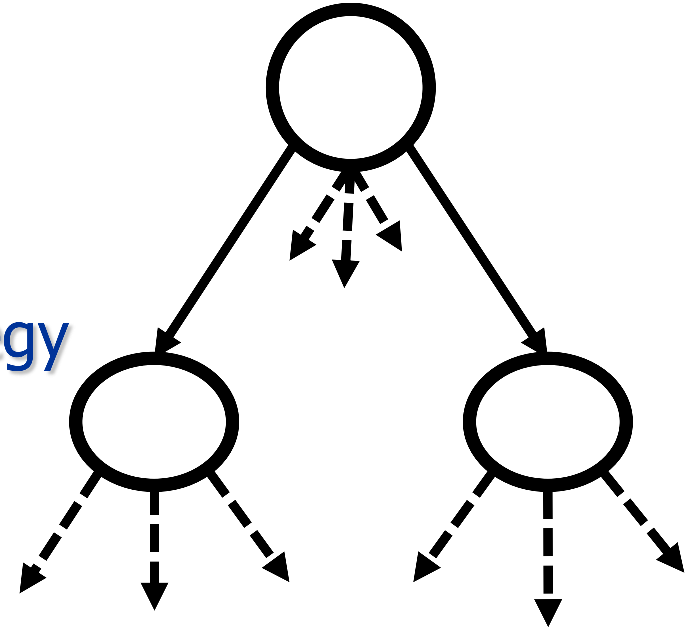
Buffers needed for 2 reasons:

2. Speed requirements
   - too few buffers can cause slowdown
   - "slack mismatch" in reconvergent paths



Search Space

- The following are the unknowns:
  - placement and weights of resource arcs
    - determine schedule
  - weights of reverse arcs
    - determine buffering
- Our Algorithm: 2-level
  - Top-level: Branch-and-bound strategy
    - schedules operations
    - allocates function units
  - Bottom-level: ILP strategy
    - ensures performance constraint is met (cycle metric)
    - add optimal number of pipeline buffer stages to help!

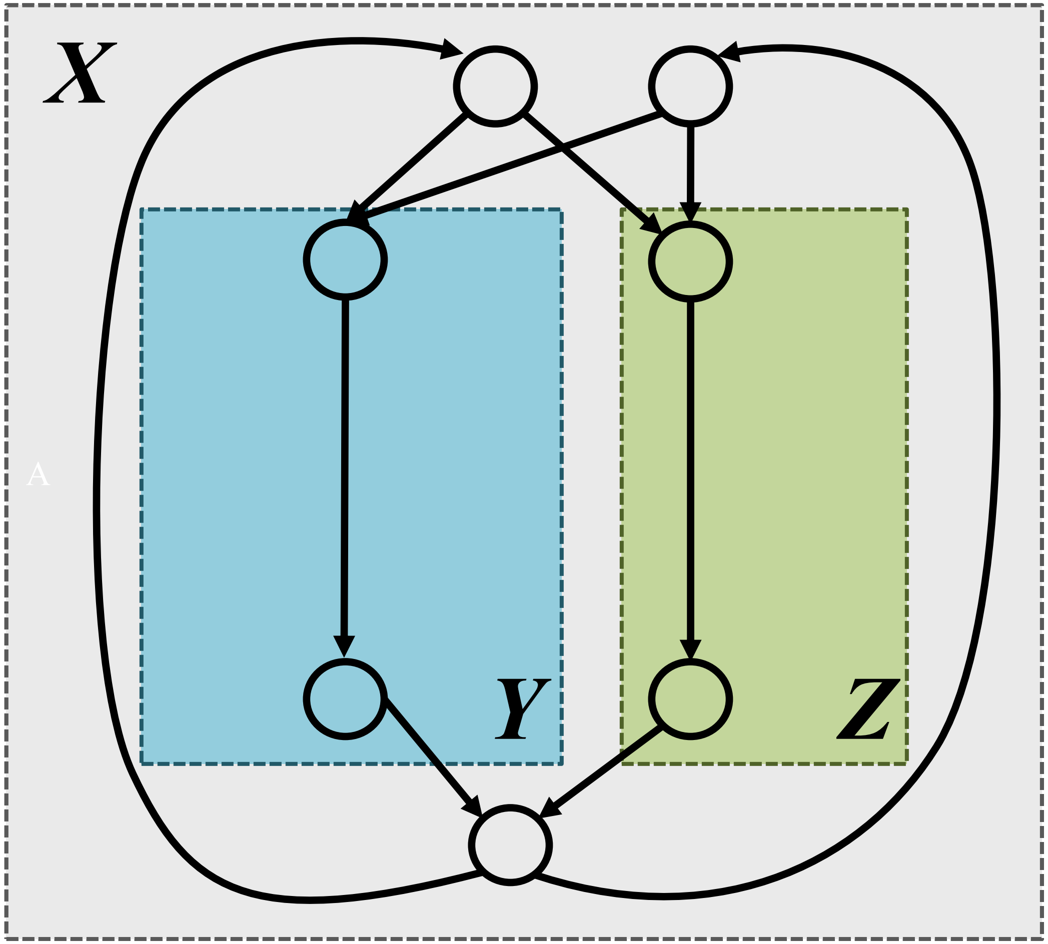


Outline

- Previous Work
- Our Approach
  - Basic approach [DATE 2012]
  - ➔ Hierarchical approach [this paper]
- Results & Conclusions

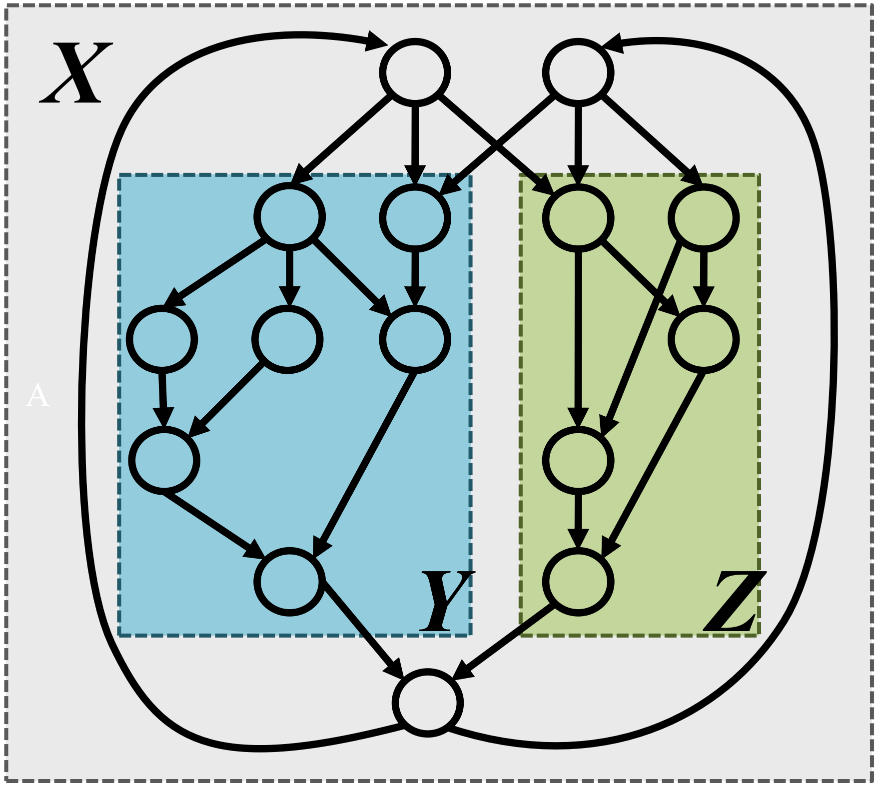
Hierarchical Scheduling

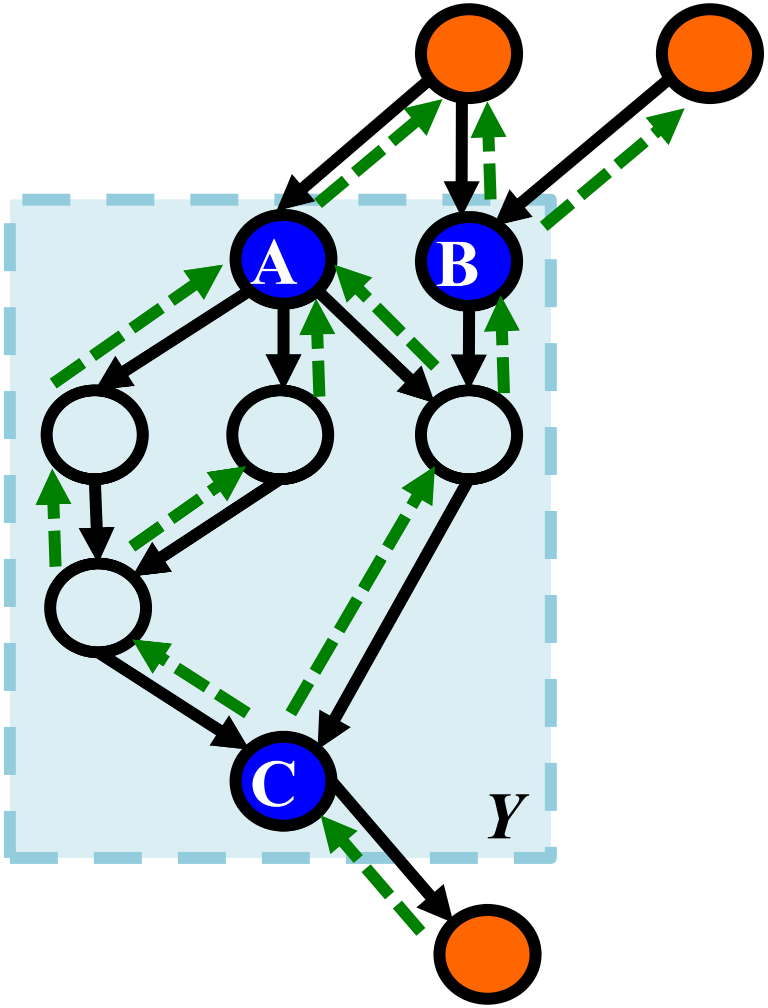
- Optimal scheduling is NP-complete
  - Need scalable method
- Hierarchical algorithm
  - Faster, scalable
- Algorithm steps:
  - Decompose
  - Schedule
  - Simplify



Simplifying block internals

- Propagate a simpler/abstract model of block
  - Only subset of nodes interact with other blocks
  - Hide internal nodes!





What do we replace a block with?

Can we replace a block (subgraph of the DFG)

with

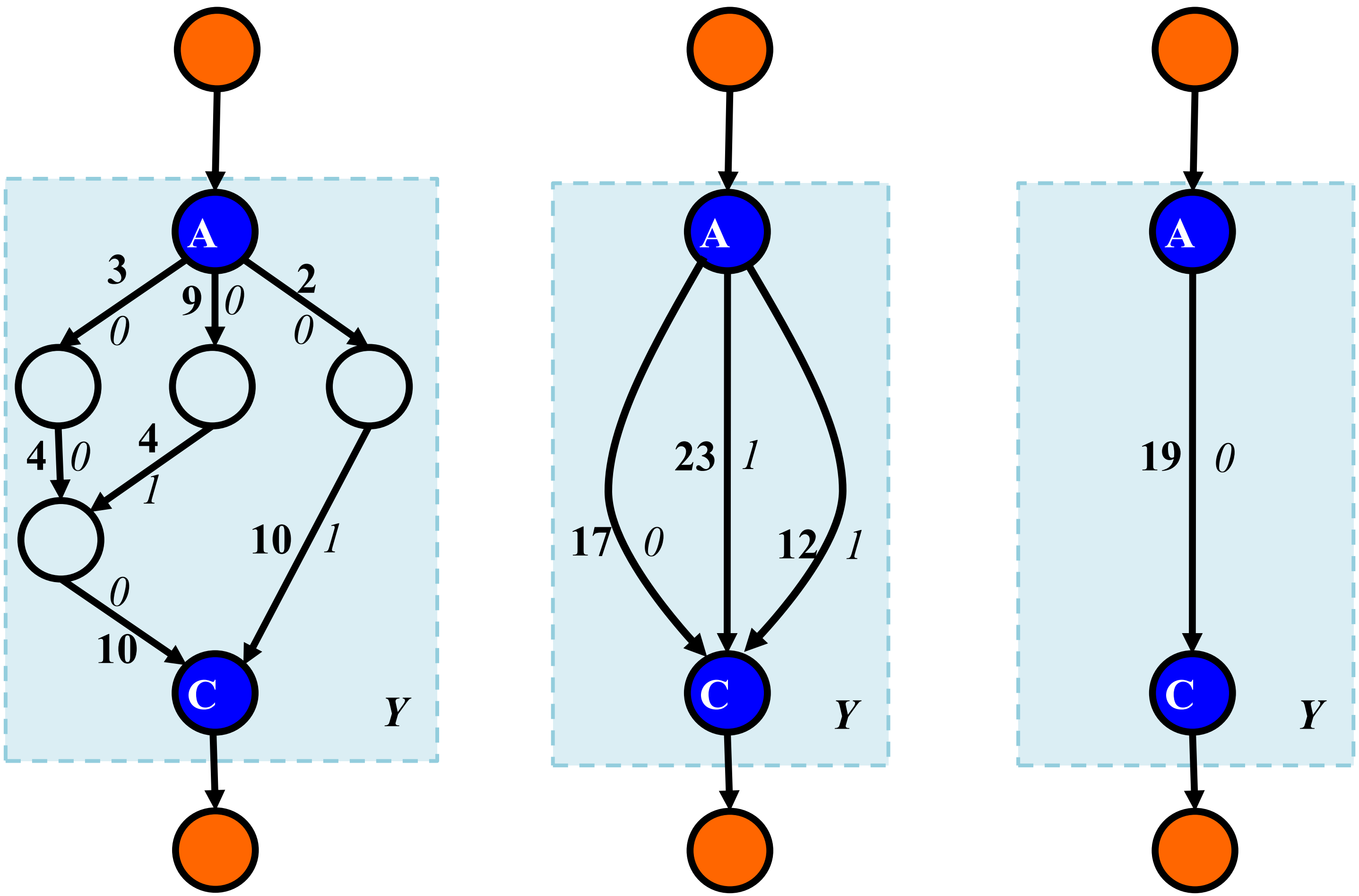
a

FIFO?

YES!

Single-Path (FIFO) Approximation

- Simplify path between a pair of interface nodes

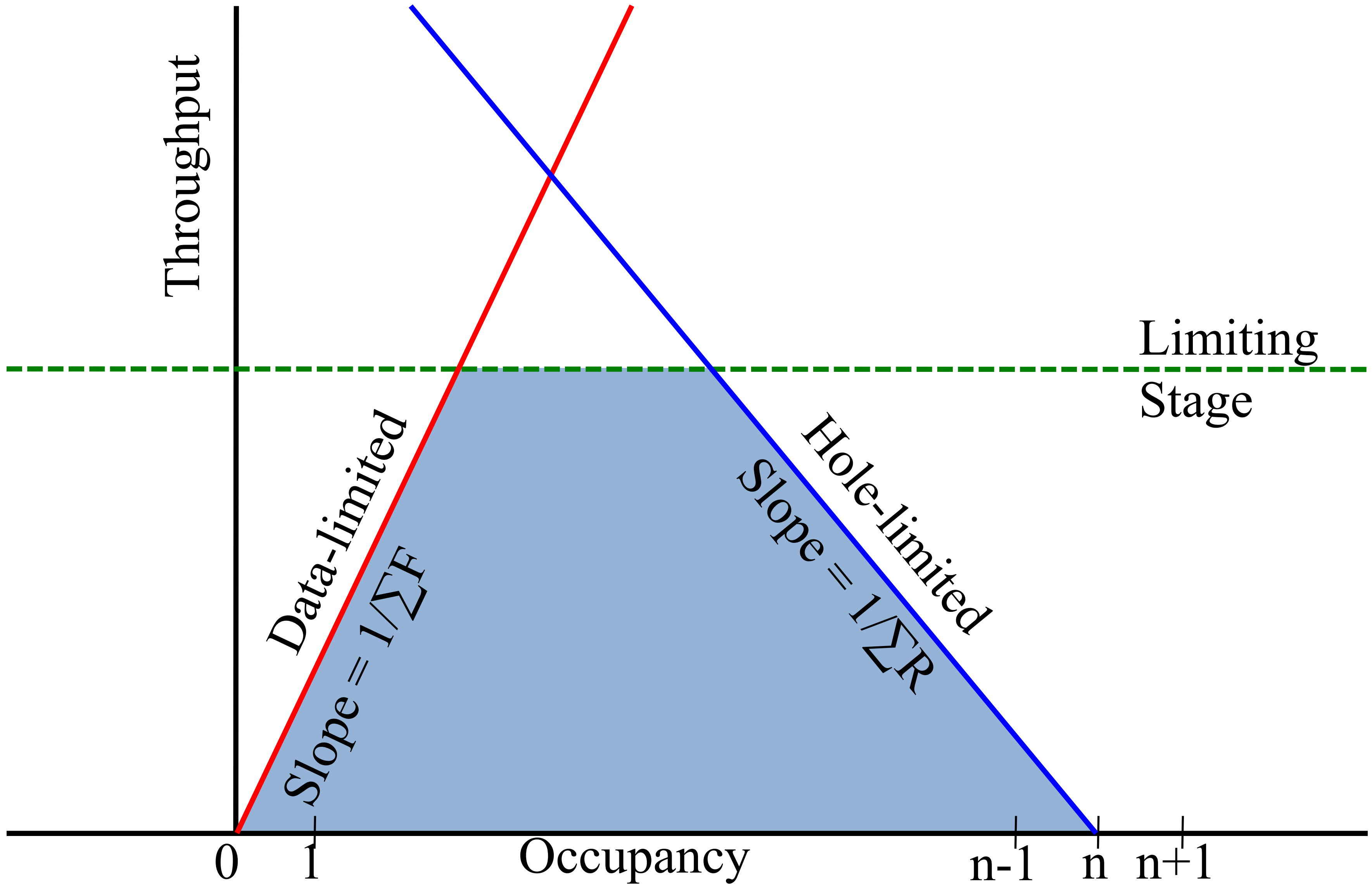


Original Paths
(A to C only)

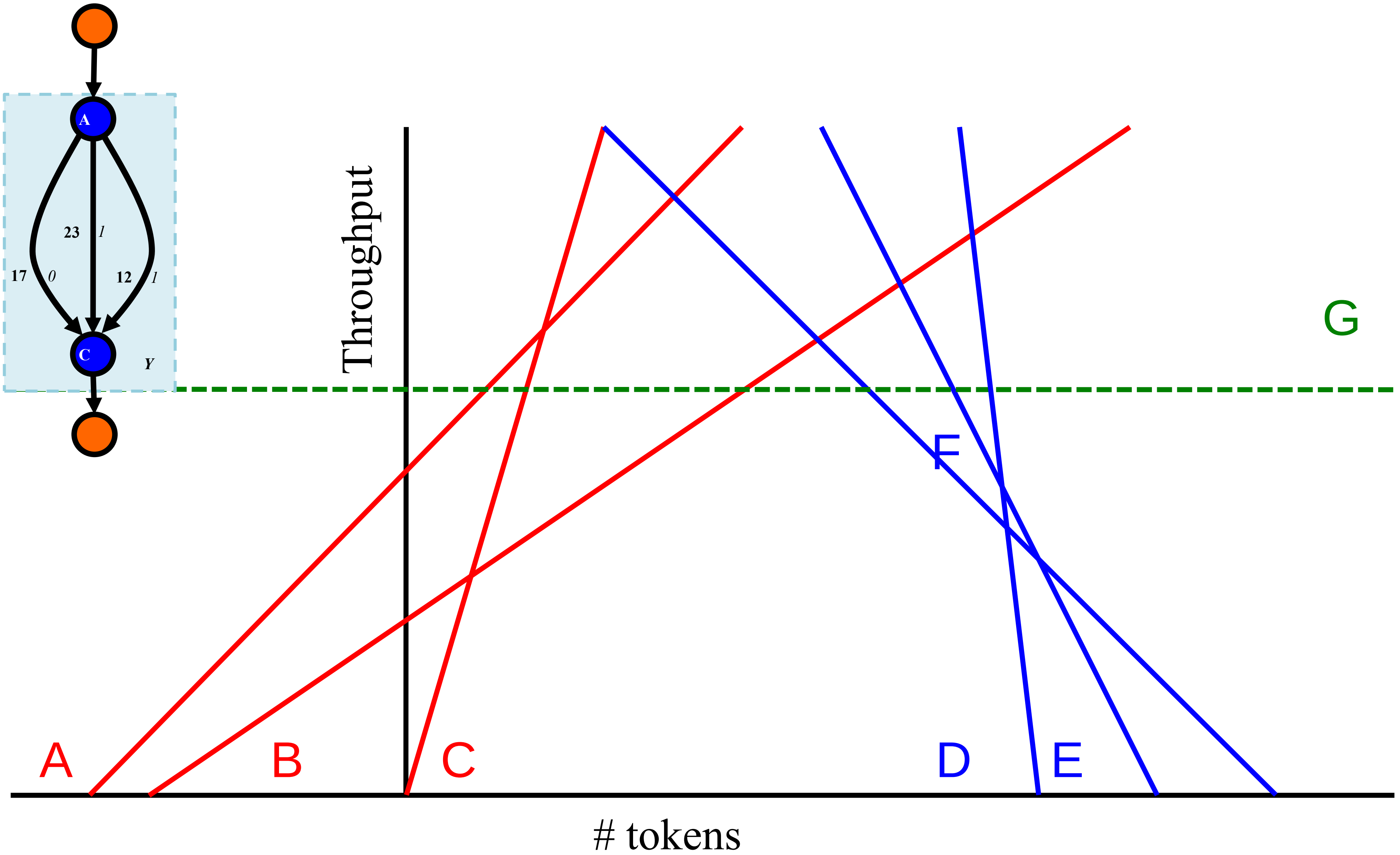
Simplified Internals

Single Path Approximation

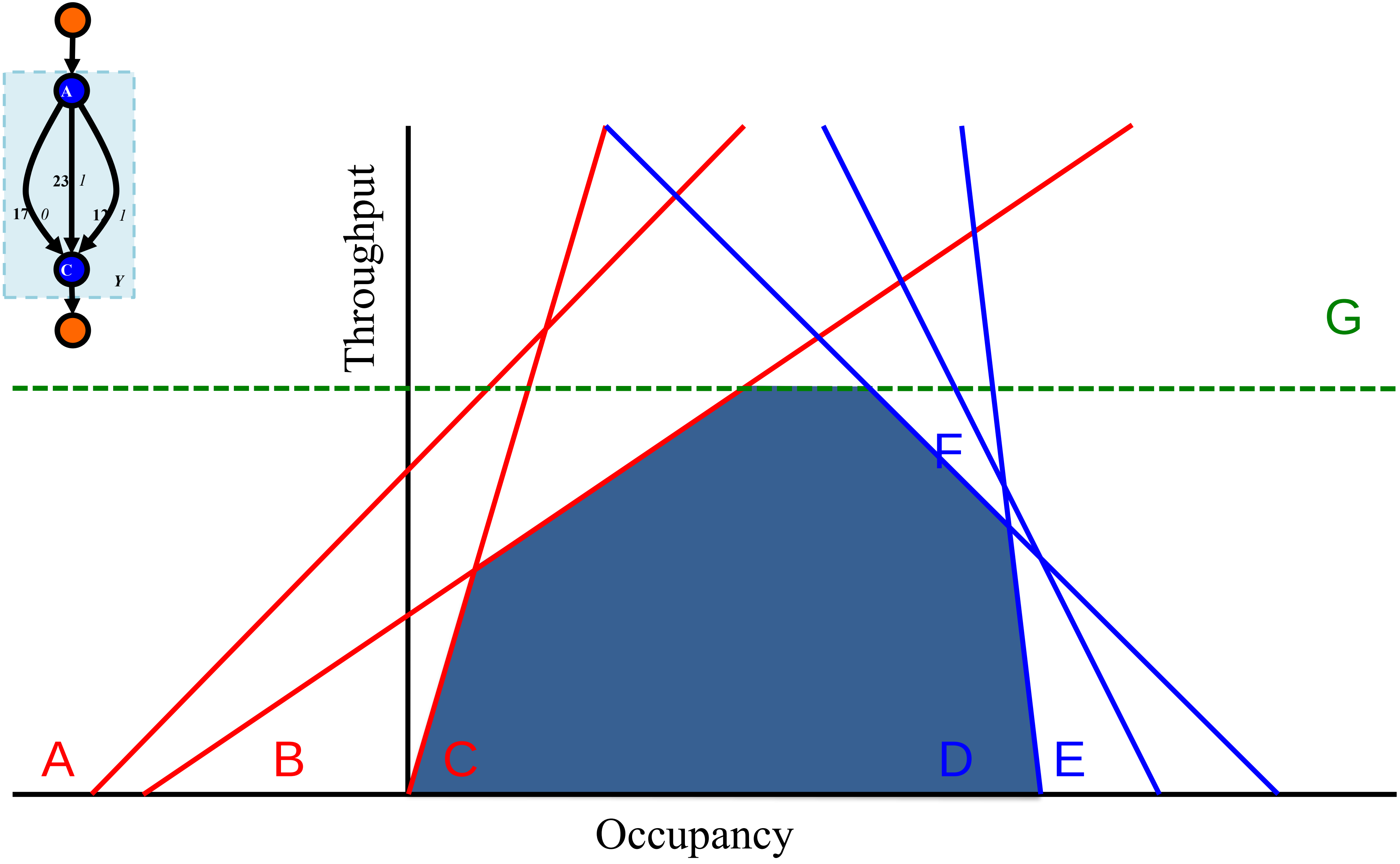
Review: Canopy Graph Analysis



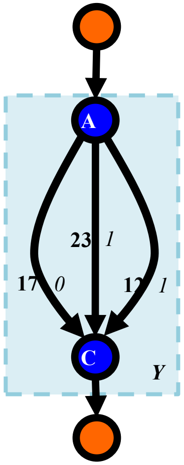
Canopy Graph: Single-Path Approx.

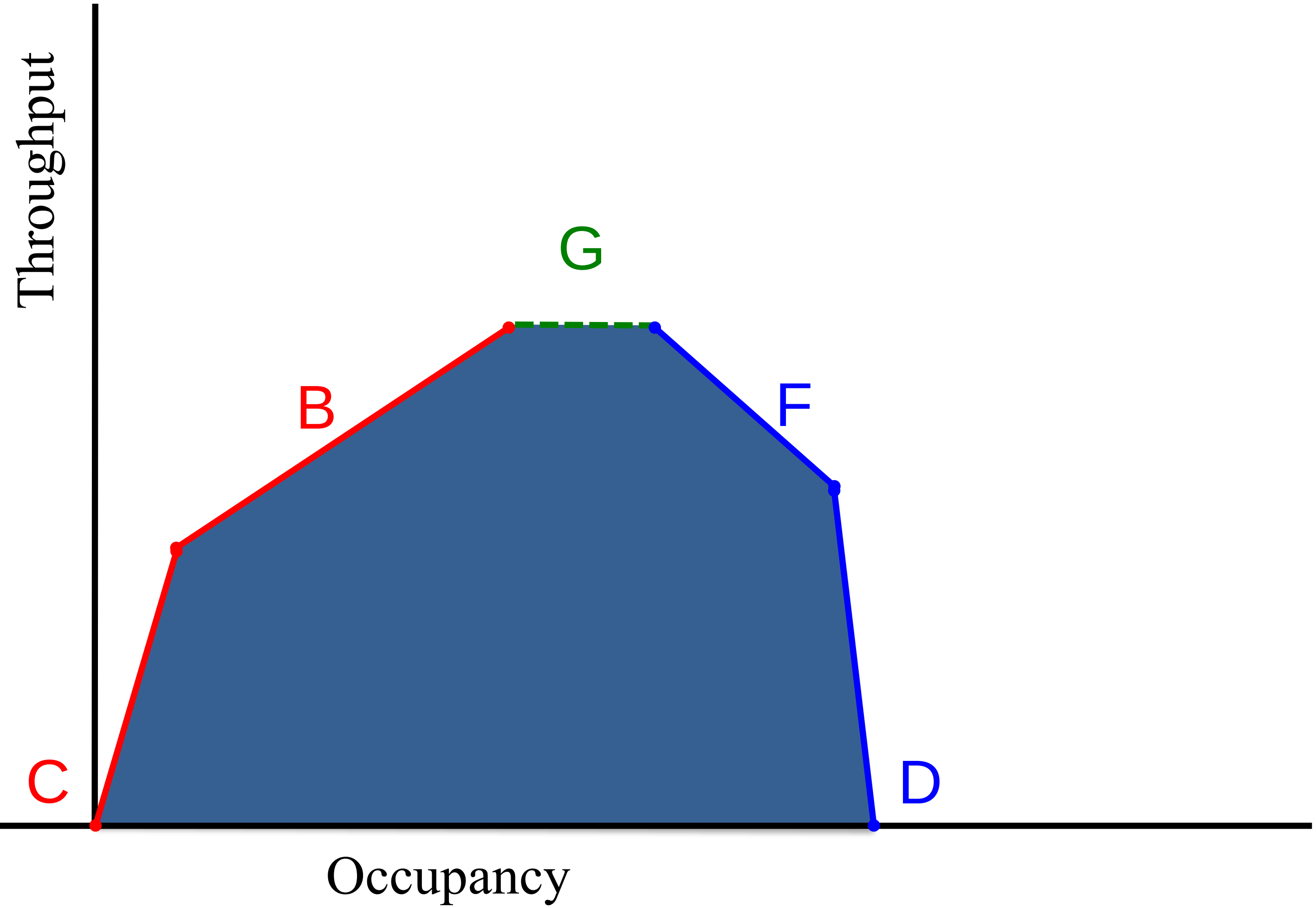


Canopy Graph: Single-Path Approx.

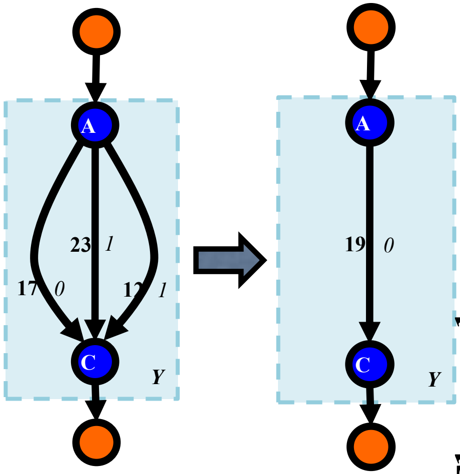


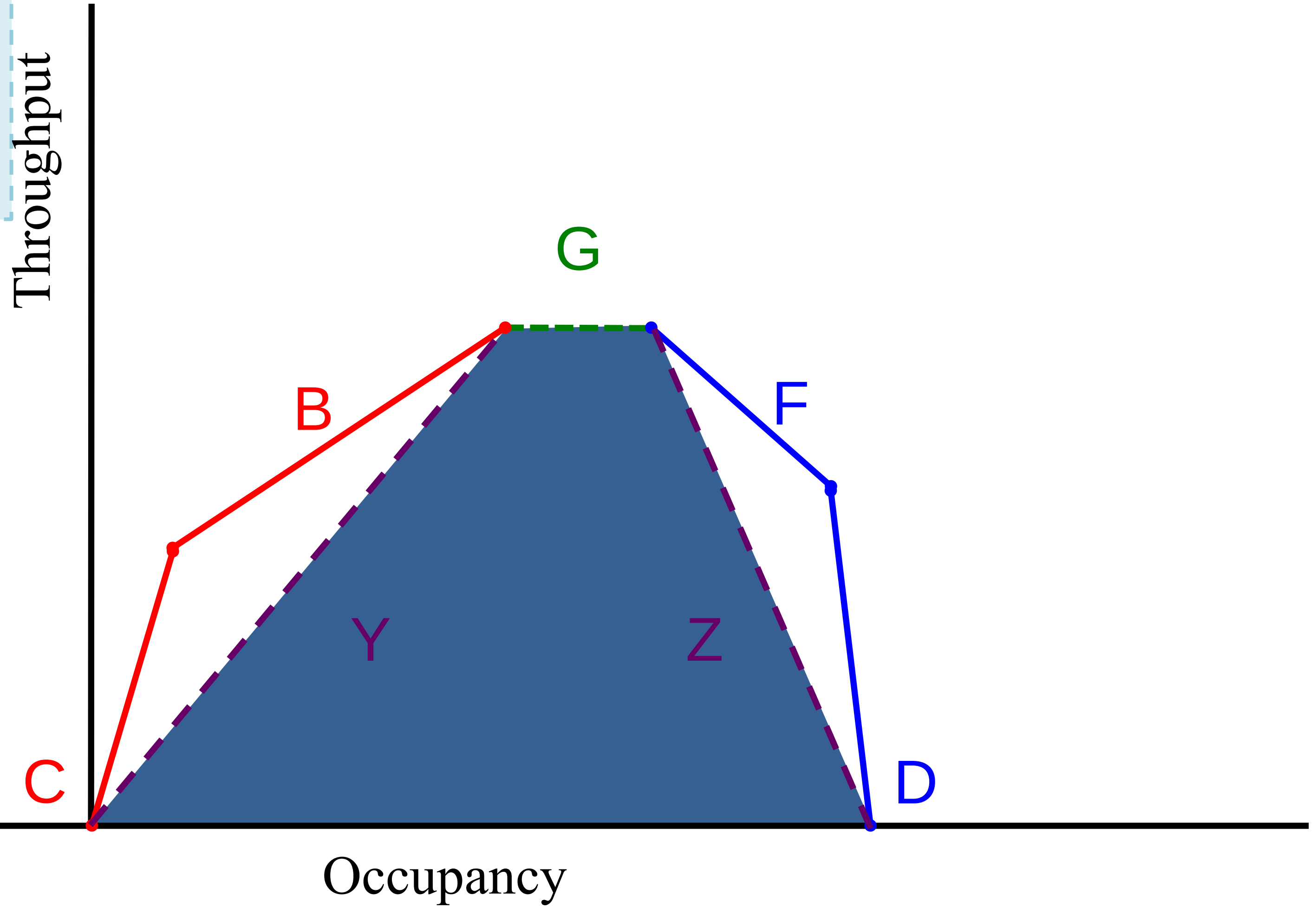
Canopy Graph: Single-Path Approx.



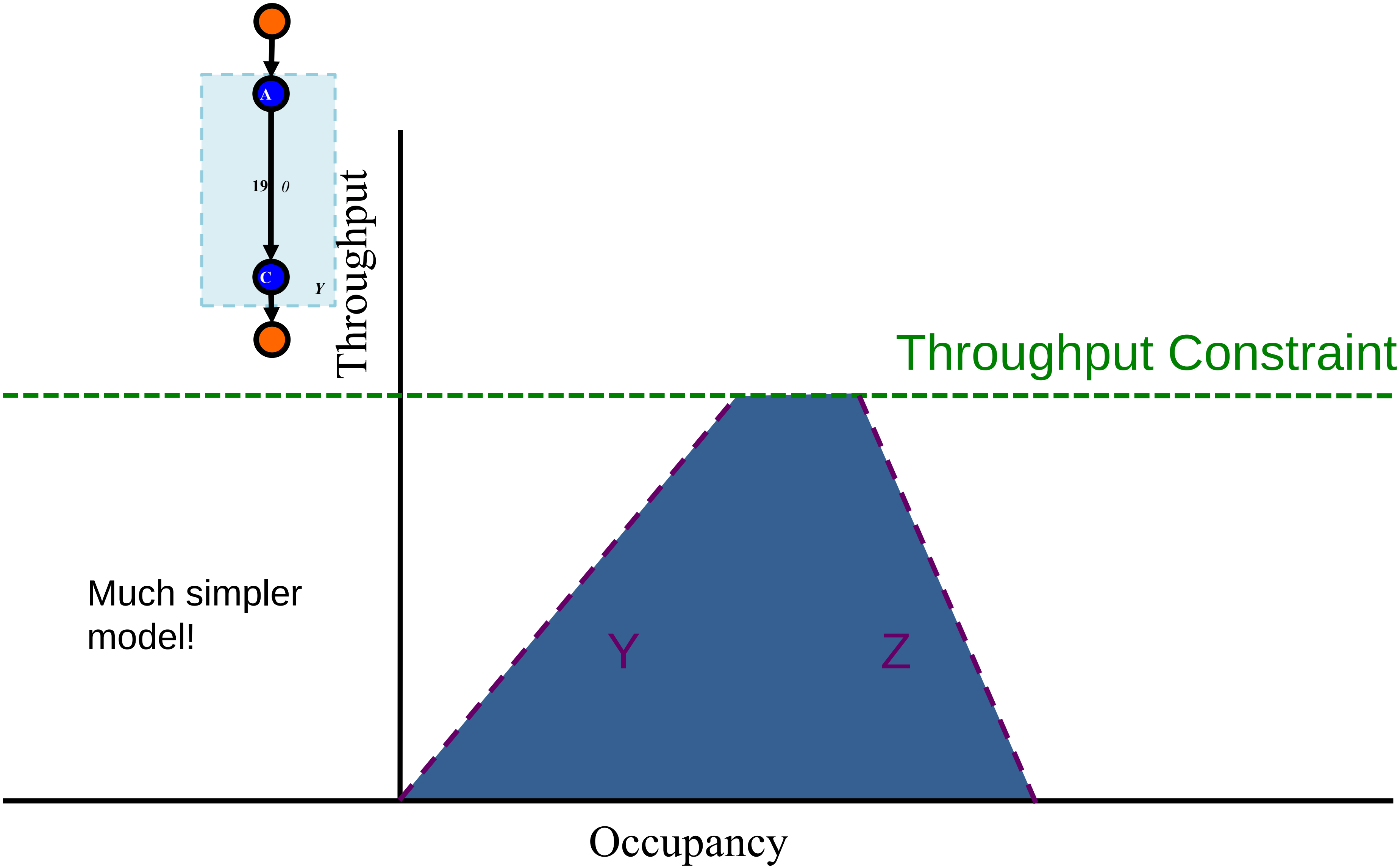


Canopy Graph: Single-Path Approx.



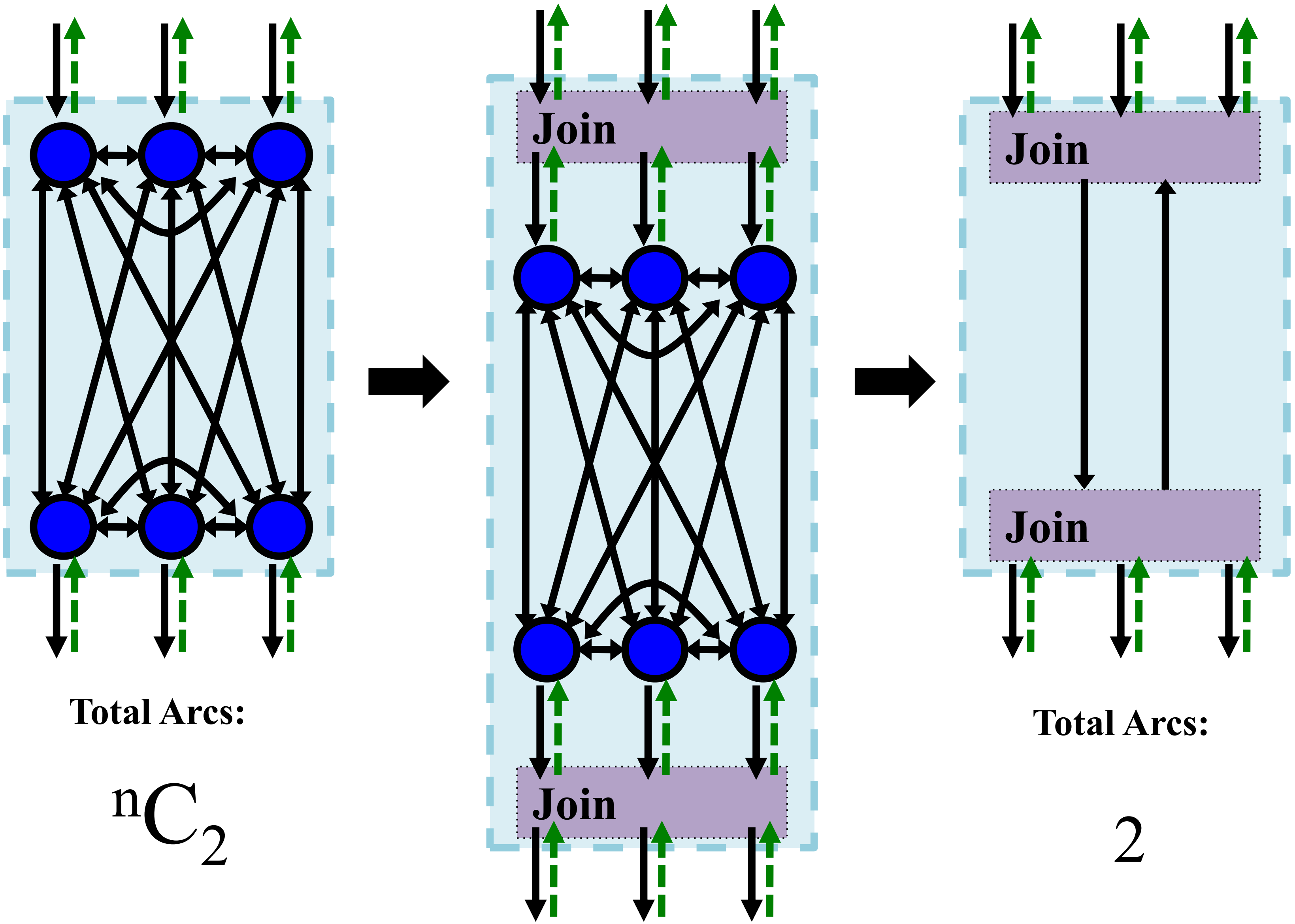


Canopy Graph: Single-Path Approx.



Much simpler model!

Two-port Transformation



Experimental Setup

- Examples
  - 6 DFGs, 20 test cases
  - Throughput specified → minimize area

- Comparisons:
  - multi-token vs. single-token approaches
  - optimal multi-token vs. hierarchical multi-token
  - trends in hierarchical approach

- Tool implemented in java on Macbook Pro
  - for ILP, use CPLEX tool

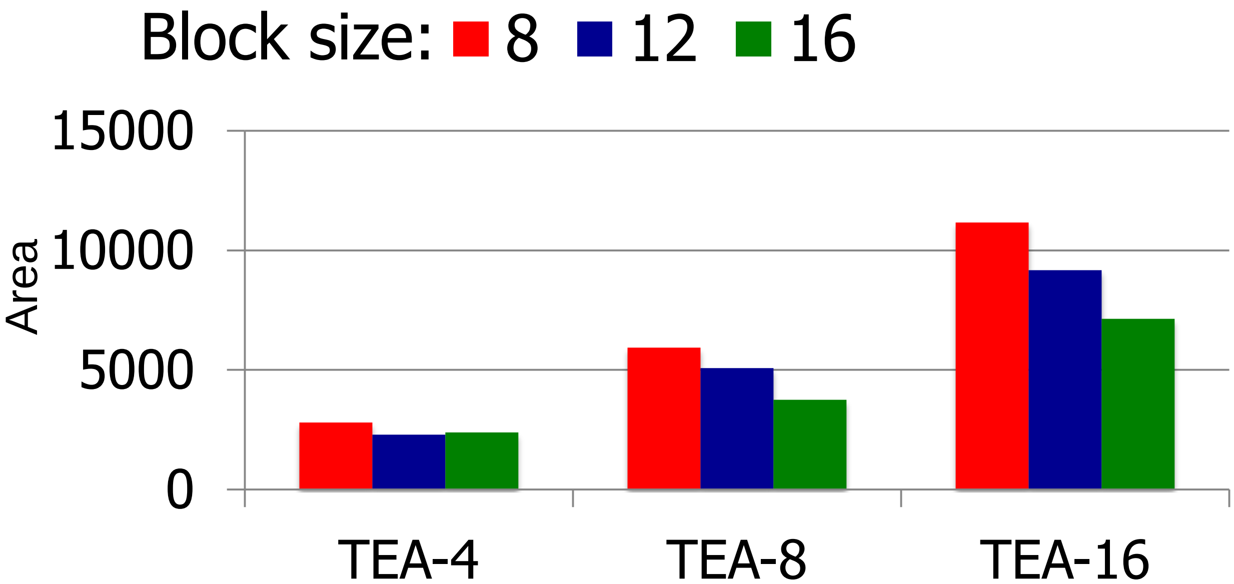
Results: Optimal vs. Hierarchical

Multi-Token Synthesis Results

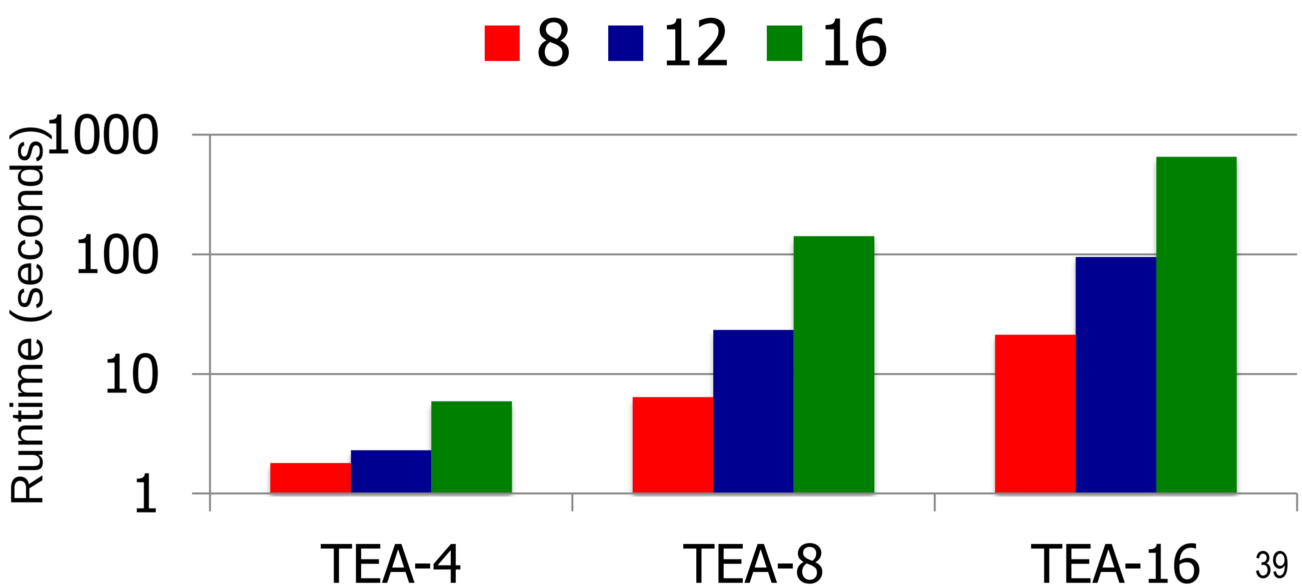
Benchmark	Cycle Time Constraint	Optimal [1] Area (unit)	Optimal [1] Runtime (s)	Hierarchical Area (unit)	Hierarchical Runtime (s)
ODE	130	814	0.99	826	0.38
ODE	258	556	0.98	556	0.35
DP8	66	2110	0.62	2314	0.23
DP8	258	806	0.40	1814	0.31
COS	66	4182	12.2	4196	0.68
COS	130	2124	1850	2136	31.1
7TH	98	3954	1100	3970	0.43
7TH	130	2154	* *3600*	2360	0.57
ELP	66	-	-	2352	16.7
ELP	130	-	-	1238	58.7

* indicates execution incomplete after an hour; best result found is shown.
"-" indicates tool did not produce any result within an hour.

Hierarchical Multi-Token Results

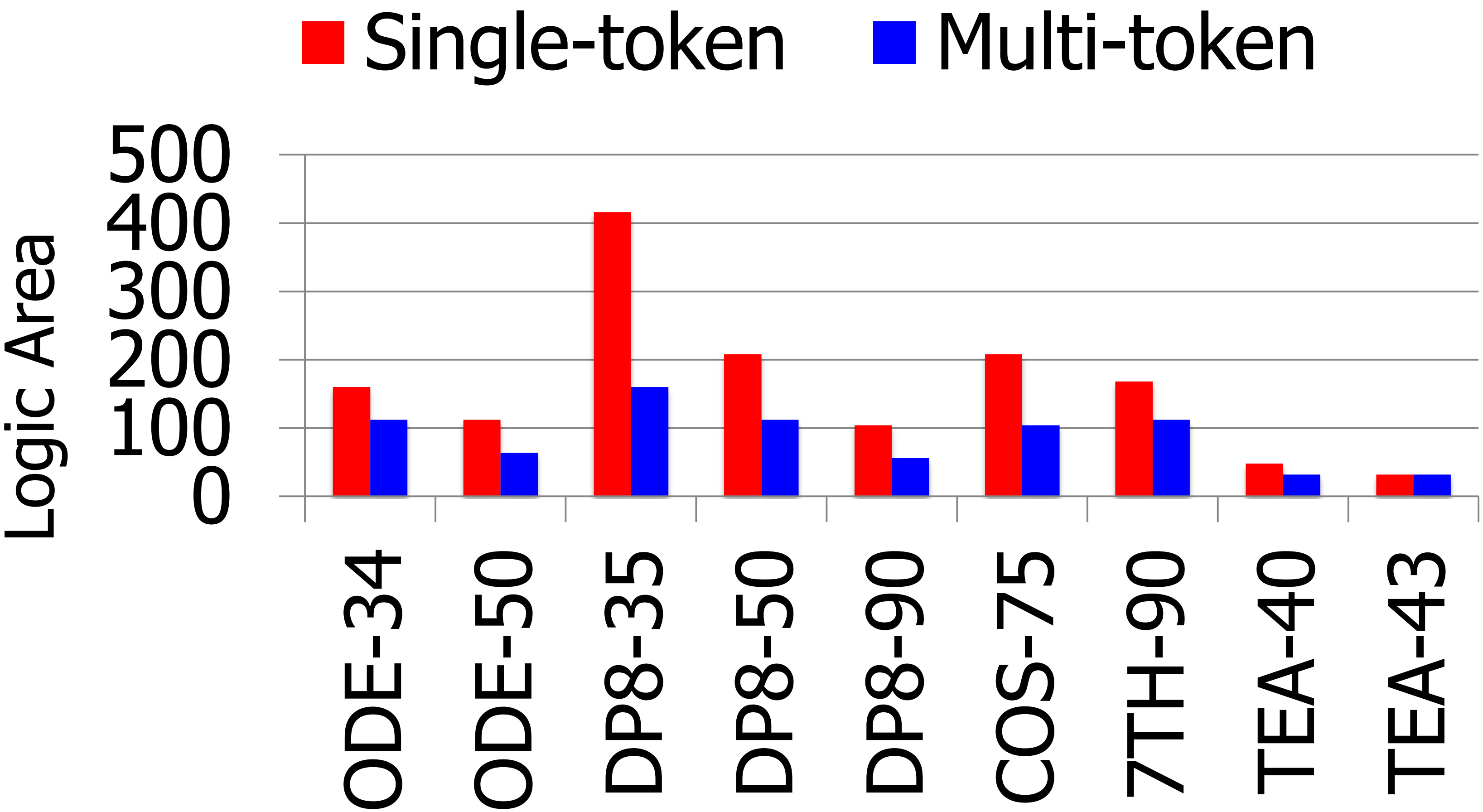


Larger block size
→ Lower area



Smaller block size
→ Lower runtime

Results: Single-token vs. Multi-token



Multi-token produces lower area solutions!
Multi-token solves problems single-token cannot!

Conclusions

- Summary of Contributions:

  - DATE 2012: First exact method for multi-token
    - async as well as sync
    - novel graphic model that captures buffering, scheduling, data dependencies
  - ASYNC 2012: Fast hierarchical method
    - can solve larger problems
    - promising experimental results
    - A Key Result: An arbitrary Marked Graph/DFG can be modeled as a FIFO!

Thank You

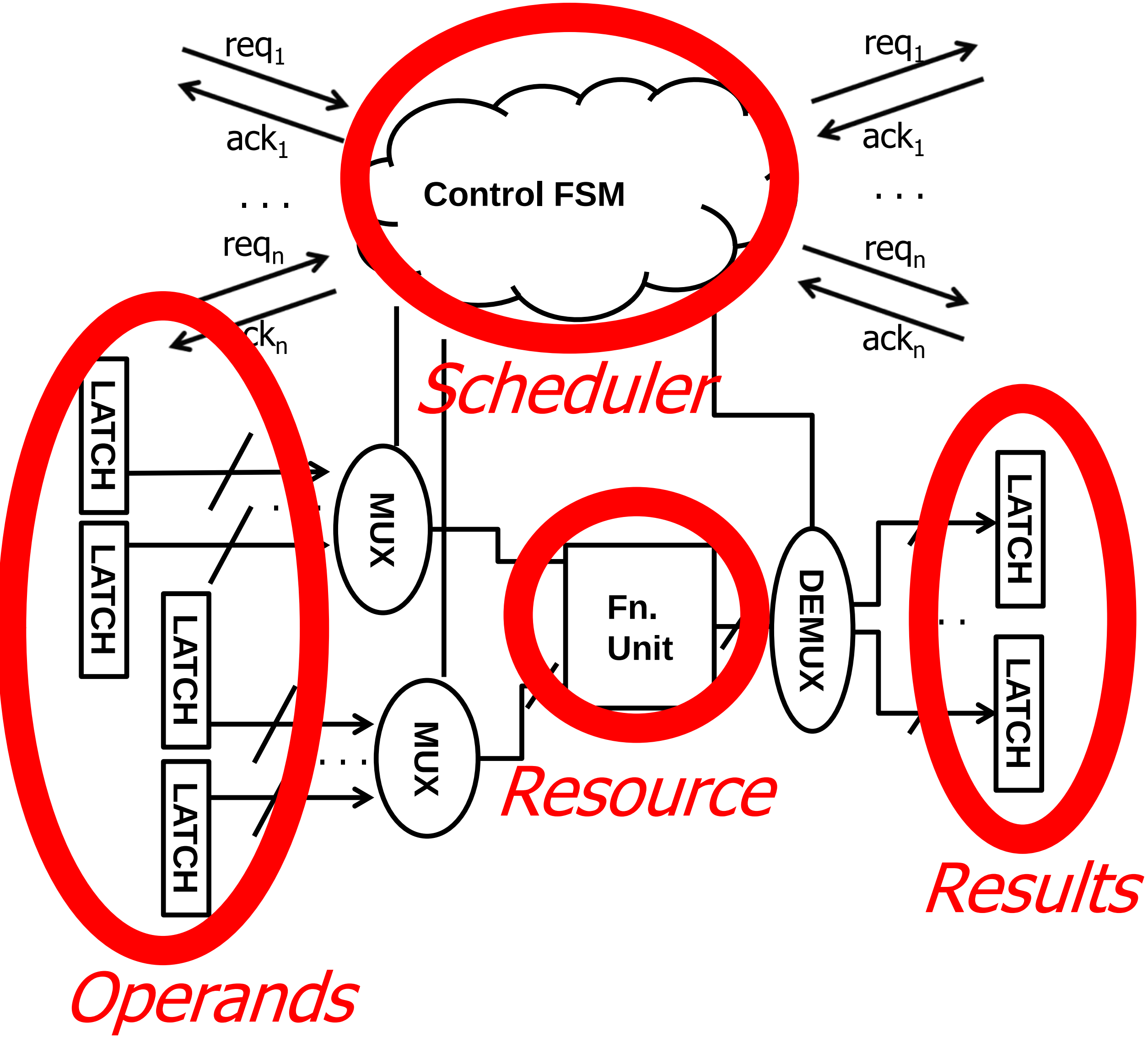
- Questions?

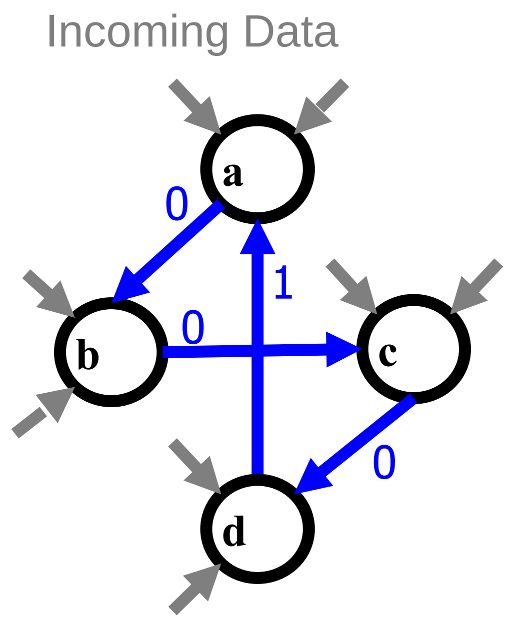
Search Algorithm

- Algorithm overview:
  - 1a. Pick an unscheduled operation
  - 1b. Allocate a compatible resource
  - 1c. Repeatedly schedule another compatible operation
  - 1d. Or, close this resource cycle
  - 2. With this partial allocation & schedule, run ILP…
    - … to determine optimal buffering and satisfy legality constraints

- Monotonicity:
  - Area and cycle time monotonically increase as you go down the search tree → Branch-and-bound
  - Several heuristics for pruning and ordering

Architectural Model





Cyclic Schedule:
a→b→c→d