A Gradual Scheduling Framework for Problem Size Reduction and Cross Basic Block Parallelism Exploitation in High-level Synthesis

Hongbin Zheng

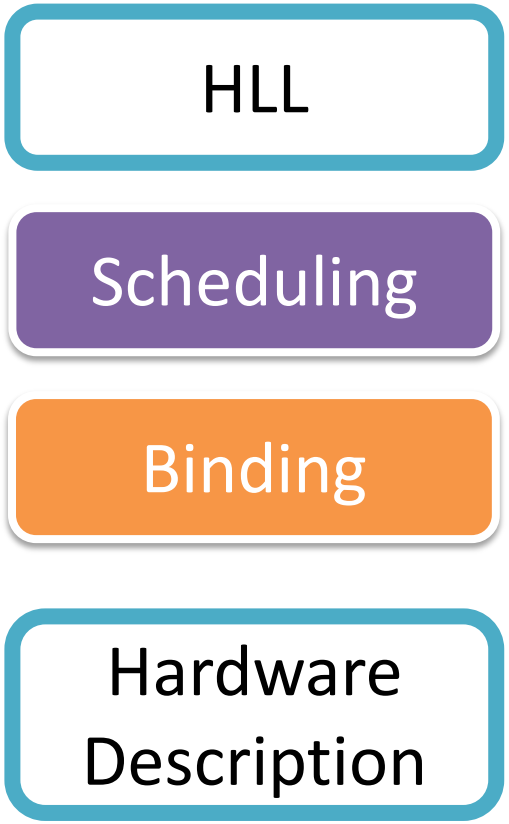
Q. Liu, J. Li, D. Chen, Z. Wang

School of Physics and Engineering

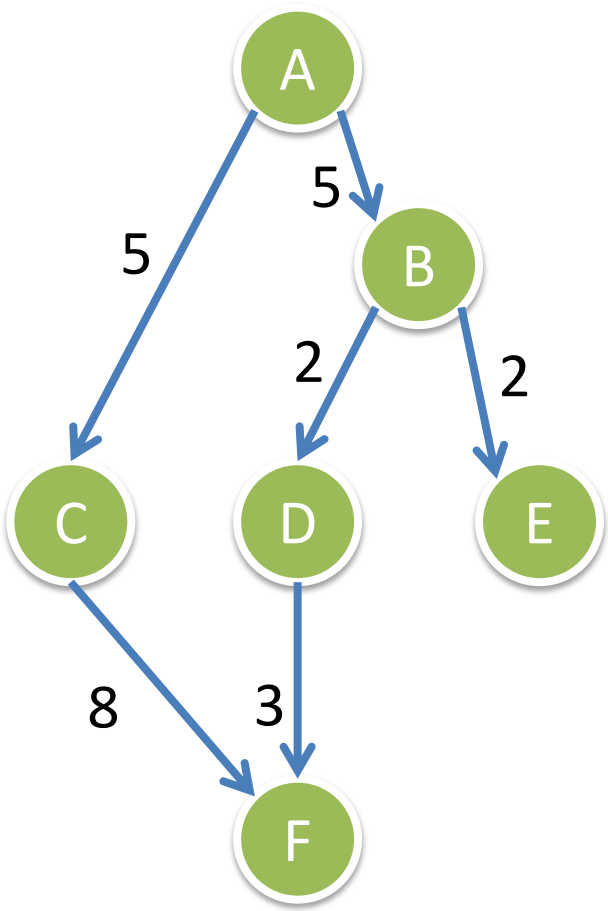
Sun Yat-sen University, P. R. China

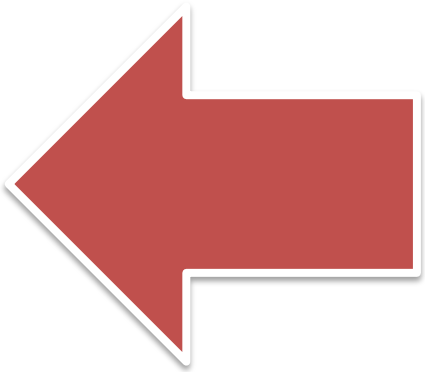
High-level Synthesis

- From high-level language:
  - C, C++, C#, Java
- **Scheduling** and binding
- Generate hardware description



Control-Data Flow-Graph





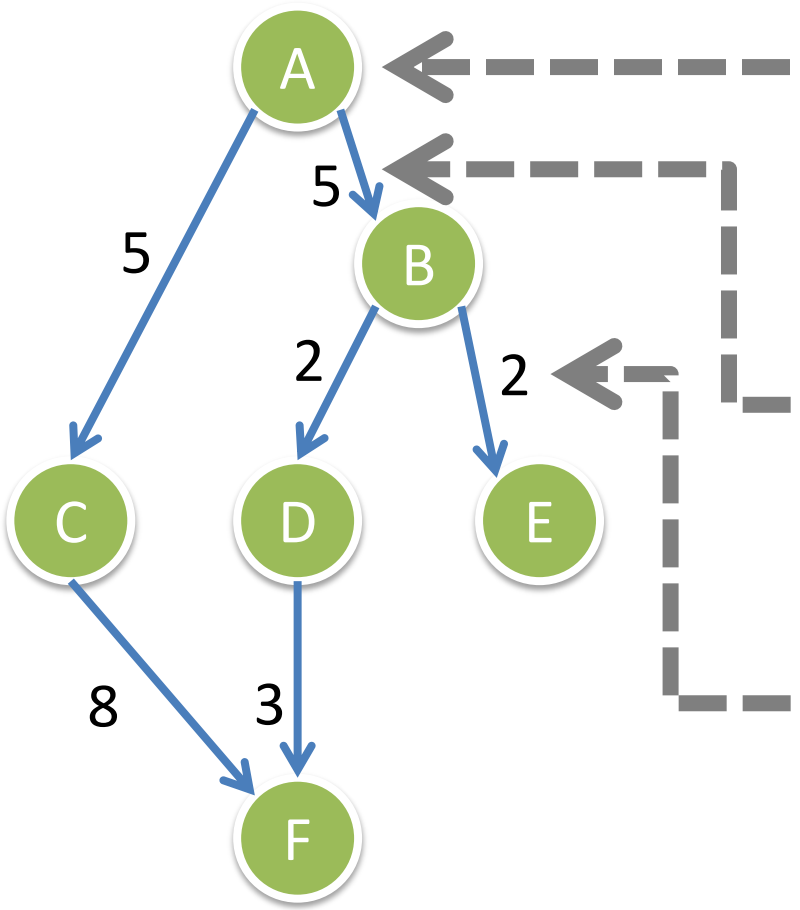
HLL

Scheduling

Binding

Hardware Description

Control-Data Flow-Graph

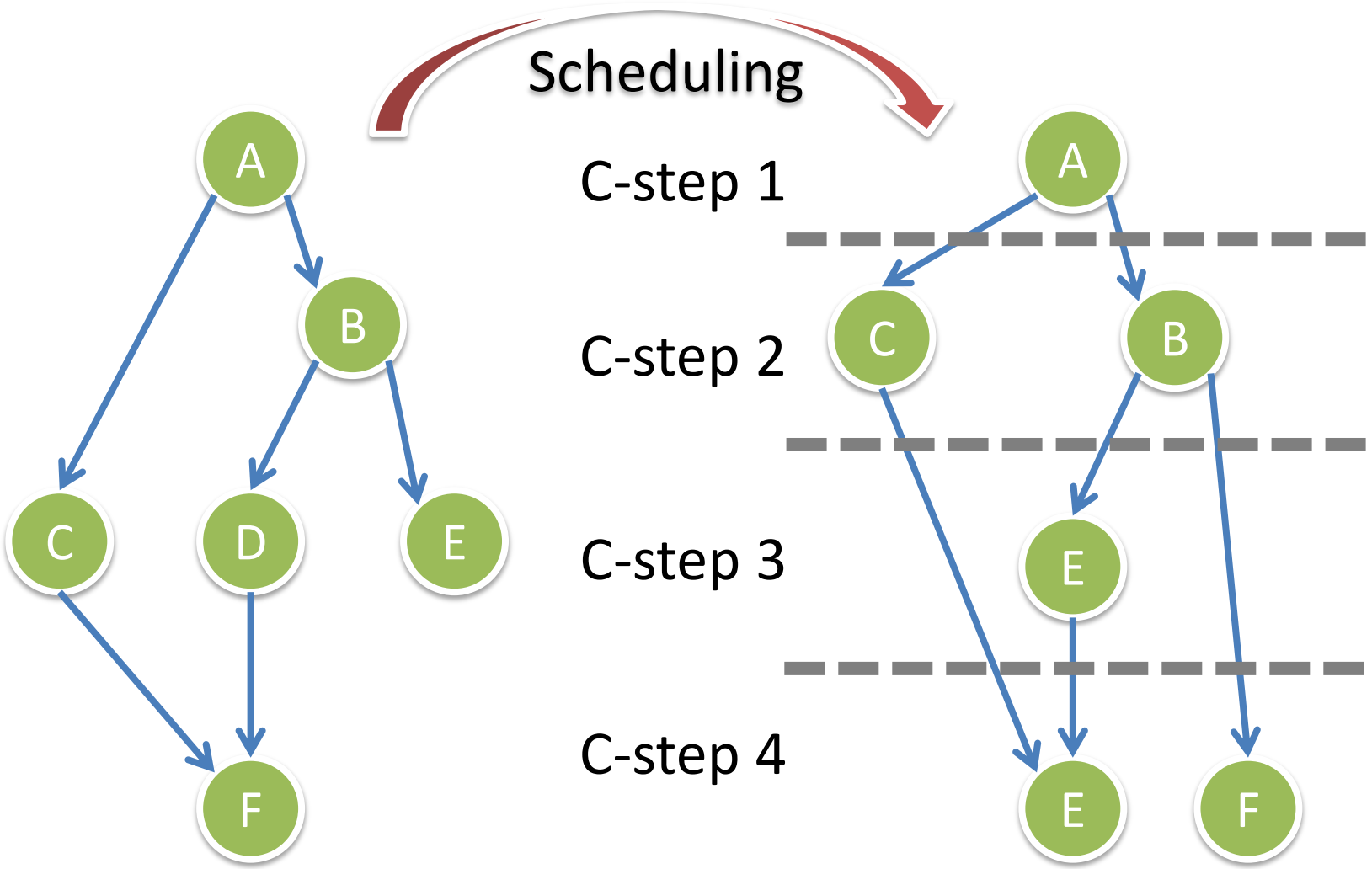


Nodes represent operations,
e.g. addition, load/store, goto

Edges represent dependencies
- Control-flow dependencies
- Dataflow dependencies

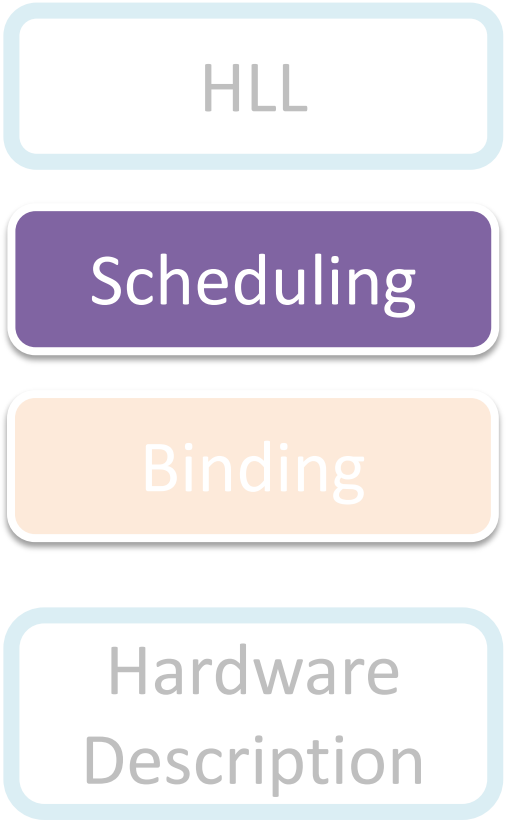
Weights represent latency
constraints between operations

Scheduling in High-level Synthesis

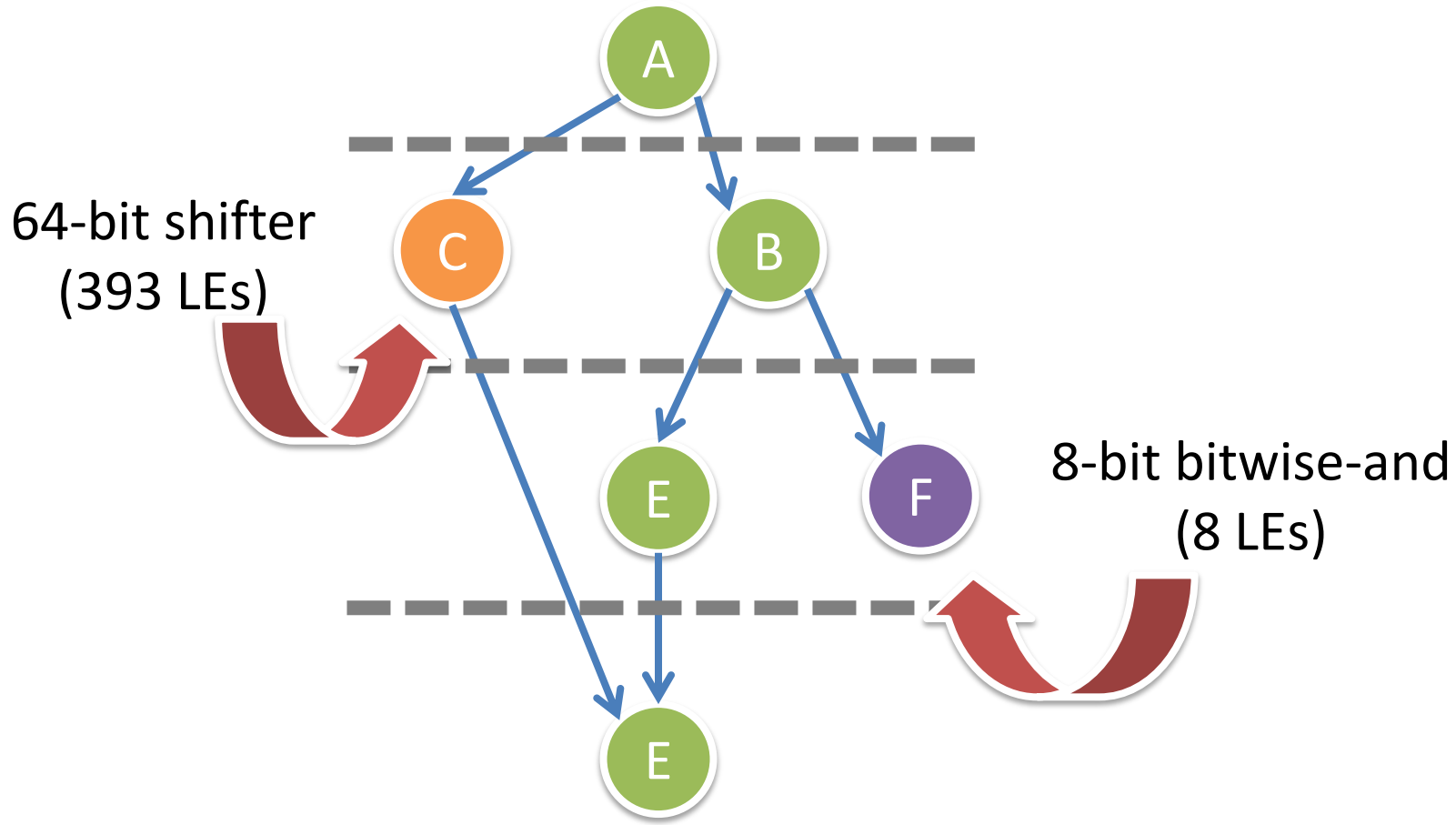


Scheduling in High-level Synthesis

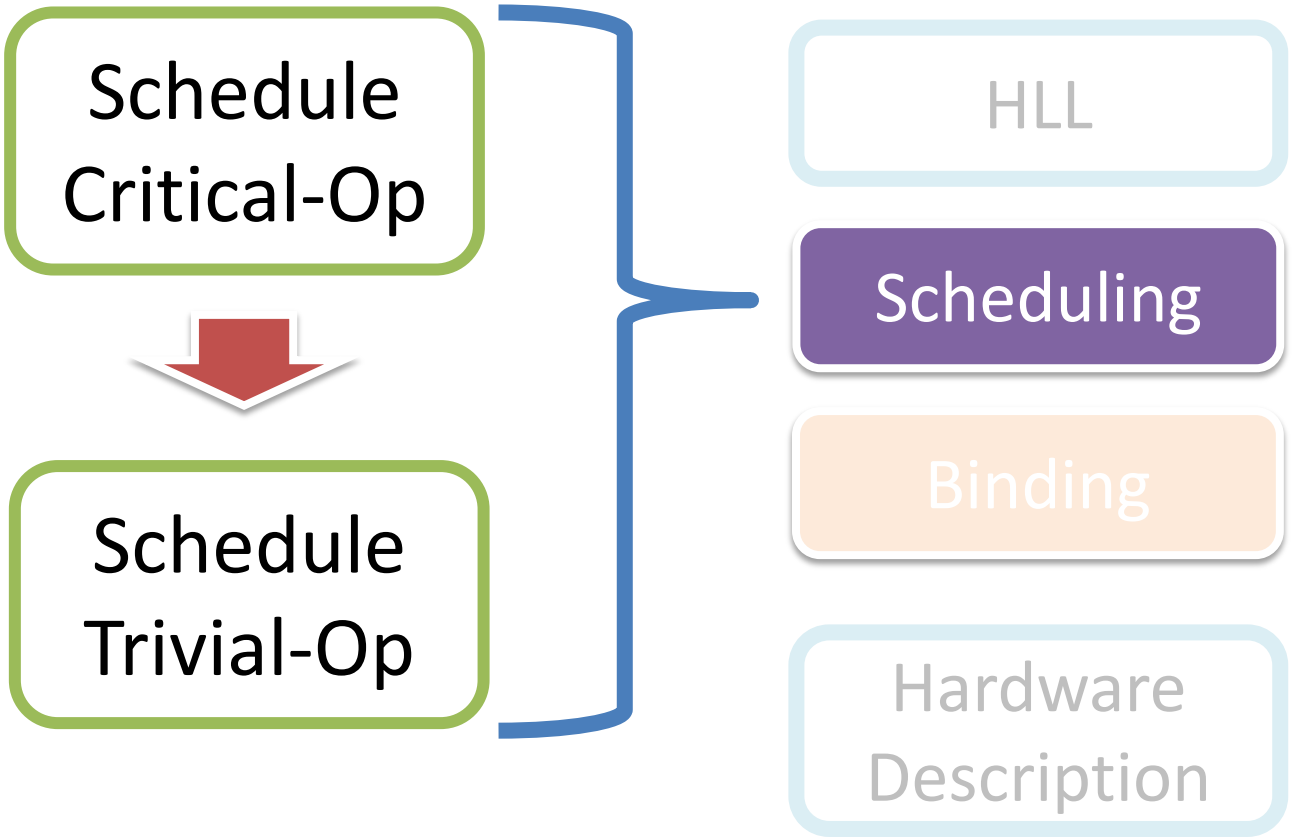
- Have big impact on synthesis quality
  - Speed performance
  - Resource usage
  - Energy consumption
- Time consuming!



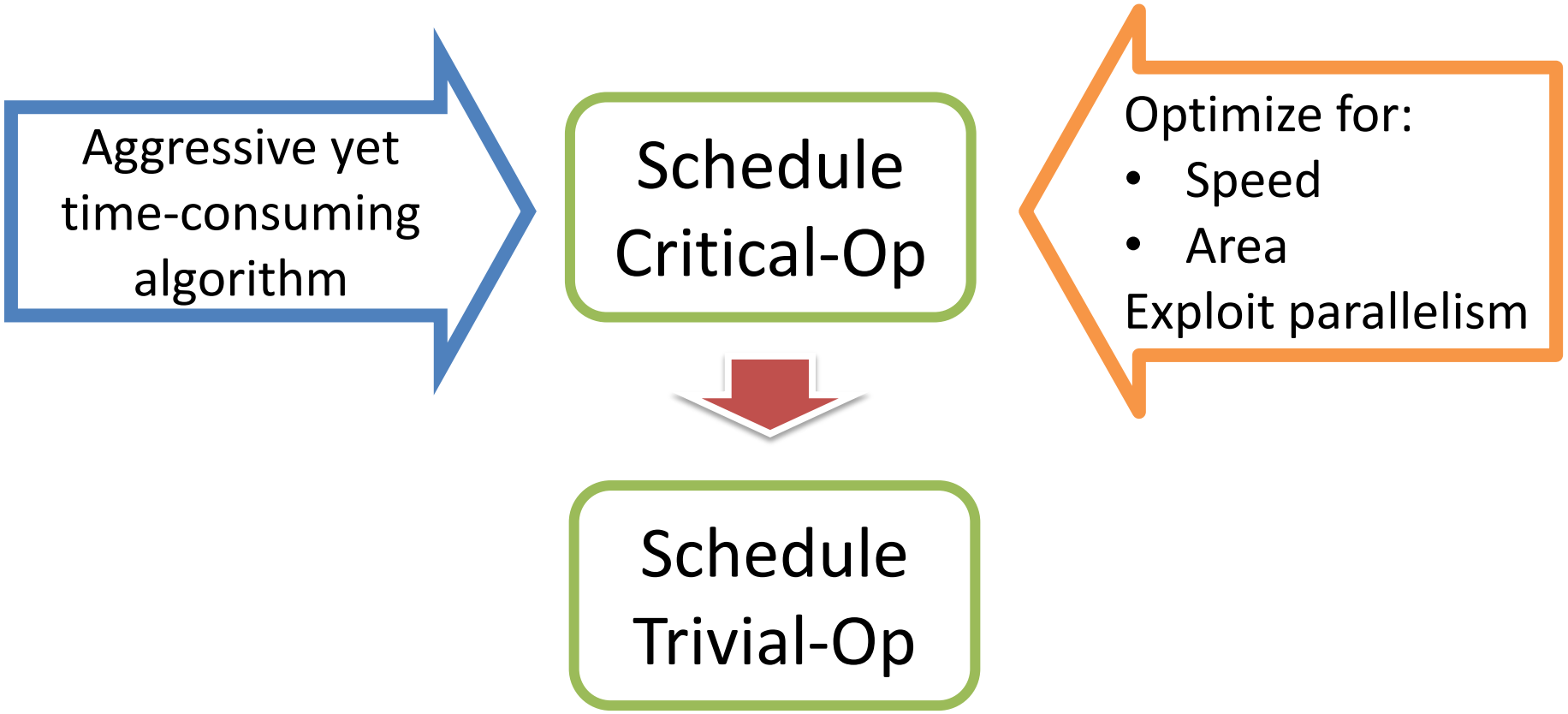
Critical v.s. Non-critical



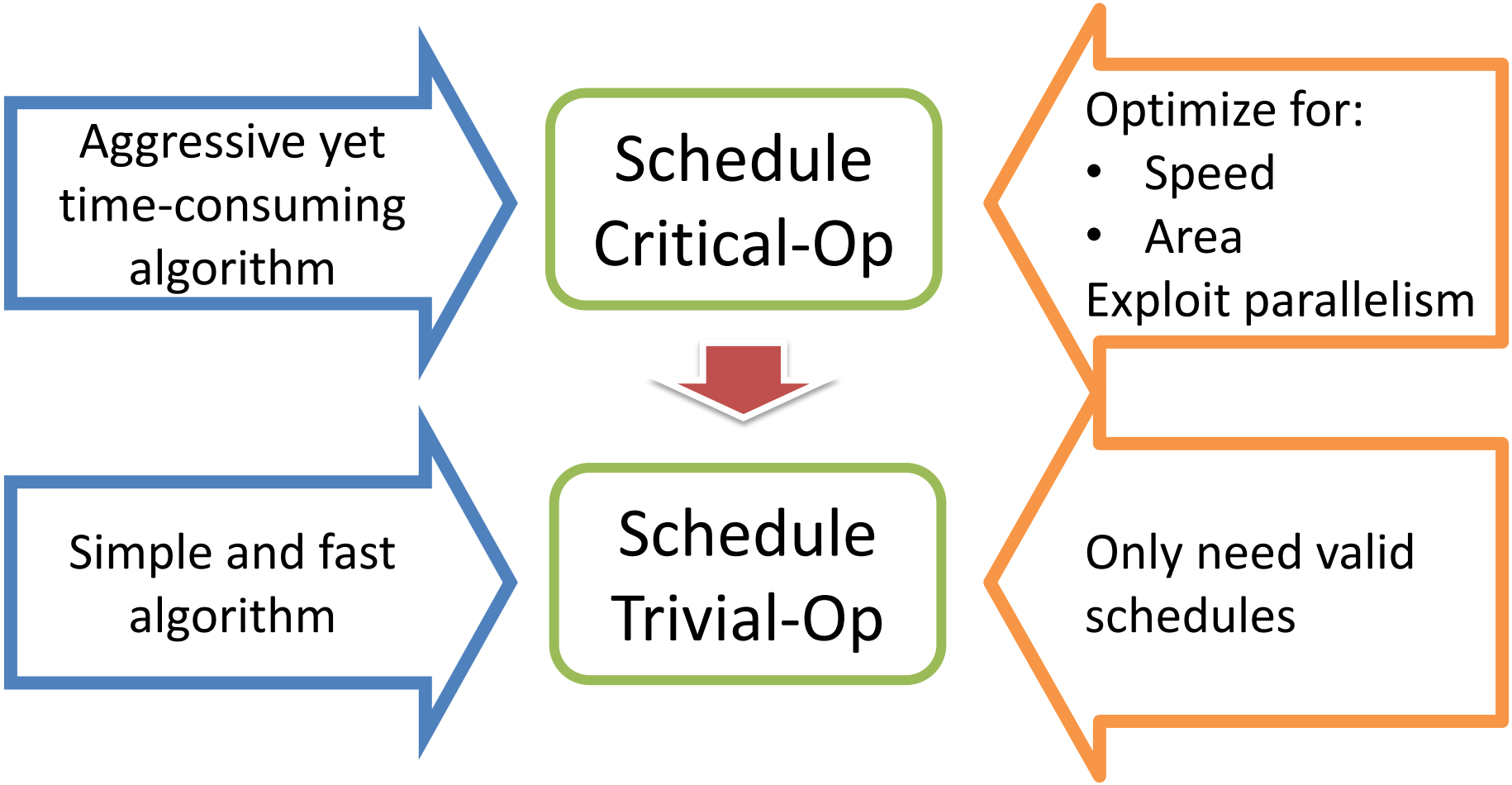
Gradual Scheduling Framework



Gradual Scheduling Framework



Gradual Scheduling Framework

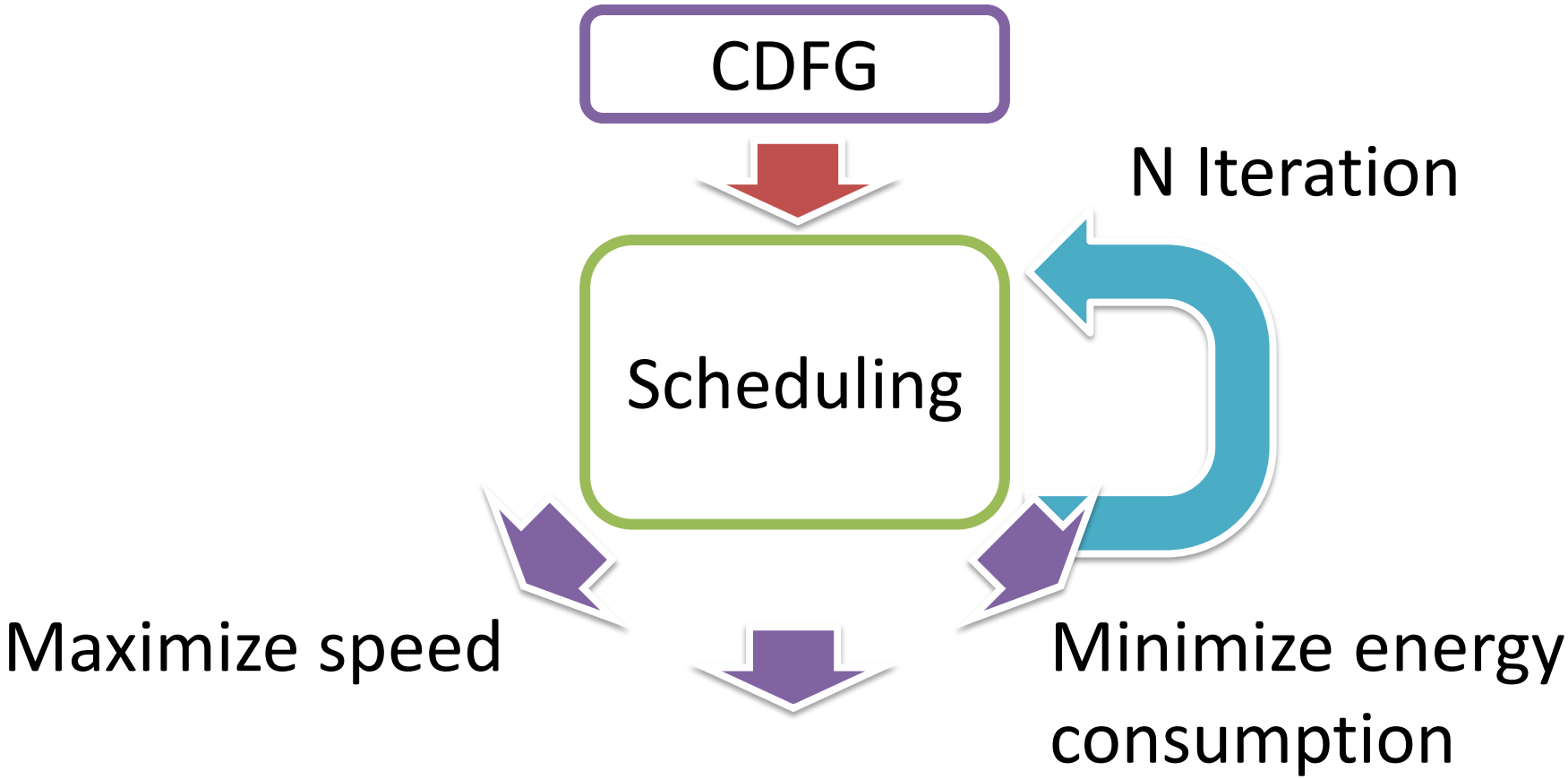


Evaluation Metrics

- Scheduling problem size reduction
  - The size of the critical part and the noncritical part
- Latency reduction, exploiting global parallelism
  - In terms of number of cycles

BACKGROUND

Scheduling for the Best QoR



Related Work

- Force-directed scheduling
  - Balance resource usage
- Path-based scheduling, HCDG-based scheduling
  - Identify mutual exclusive operations for parallelism and FU sharing
- Global Code Motion, Hyper-block Formation
  - Exploit global parallelism
- SDC scheduling
  - Optimize the latency for the whole design
  - Use soft-constraints to model other design goals

Related Work

- Force-directed scheduling
  - Balance resource usage
  - Optimize the latency for the whole design
  - Use soft-constraints to model other design goals

Schedule the critical and non-critical operations with same effort

Global Parallelism Exploiting limited by conditional dependencies in the CDFG

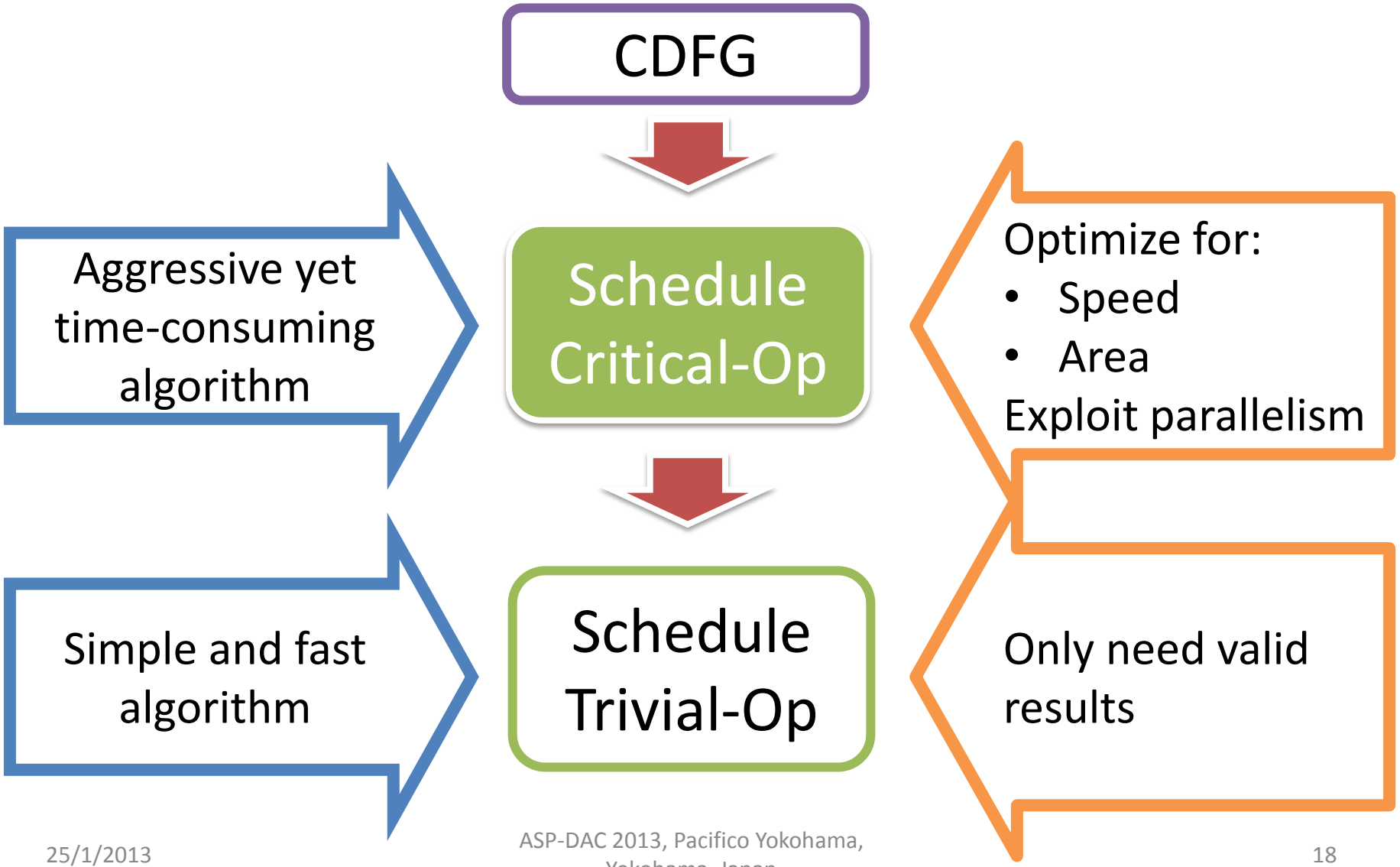
Distribution of Scheduling Effort

- The scheduling effort is distributed equally
- But, schedule of Ops have different impact:
  - Sharing large FUs is more important than small FUs
- Can we do something to make the effort distribution match the importance?

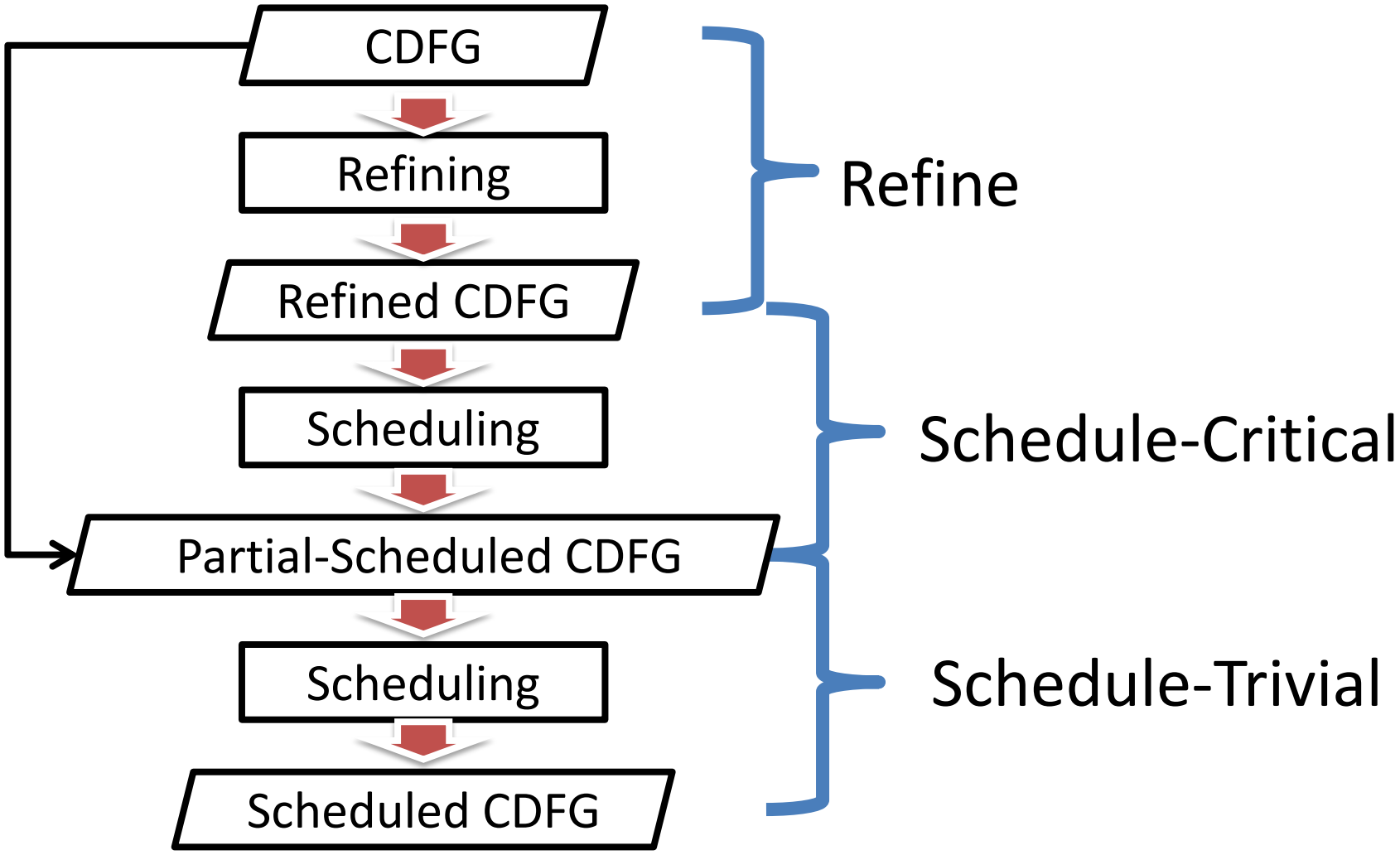
Gradual Scheduling Definition

- Given:
  - Control-Data Flow-Graph to be scheduled
  - Criticality partitioning constraints
    - In terms of area of the functional unit
- Goal:
  - Schedule the critical-operations separately from noncritical-operations

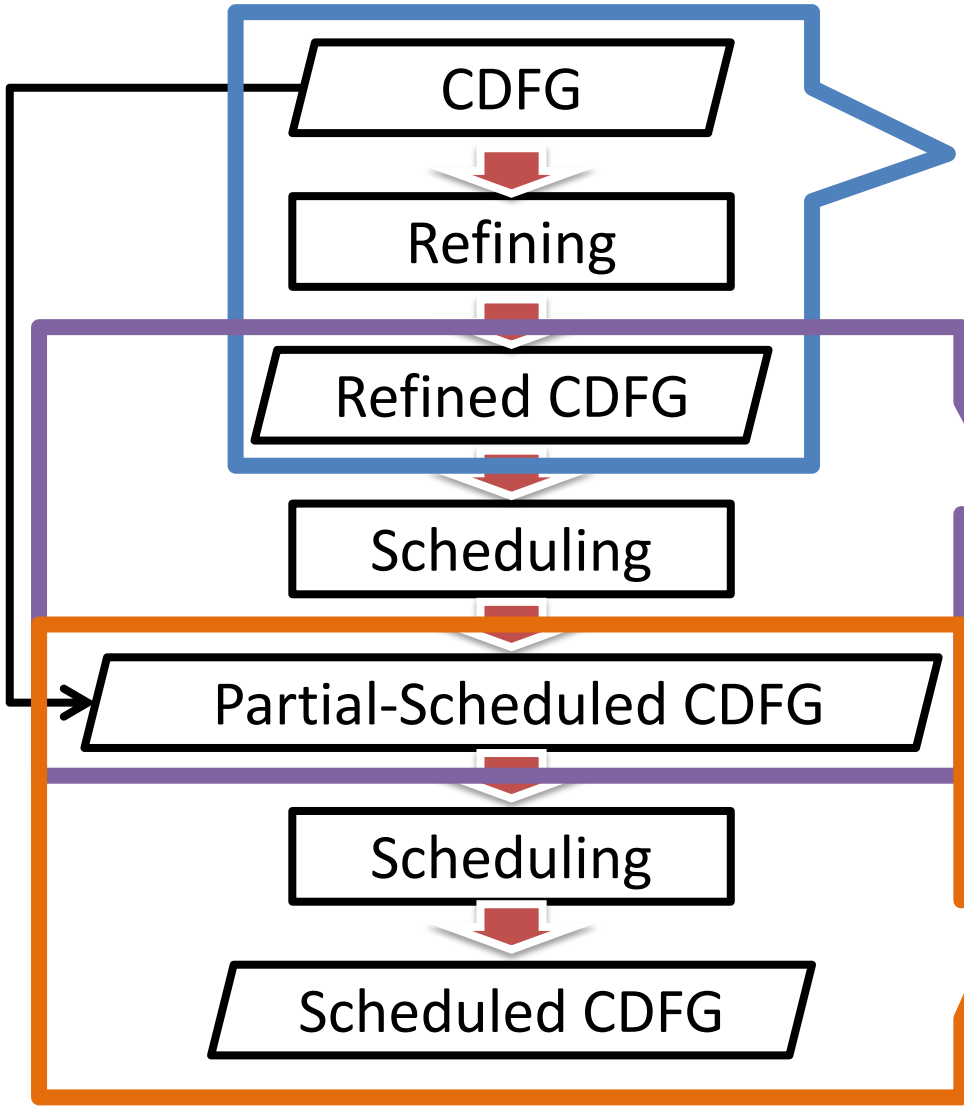
Gradual Scheduling Framework



Overview


CDFG
Refining
Refined CDFG
Scheduling
Partial-Scheduled CDFG
Scheduling
Scheduled CDFG
Refine
Schedule-Critical
Schedule-Trivial

Overview

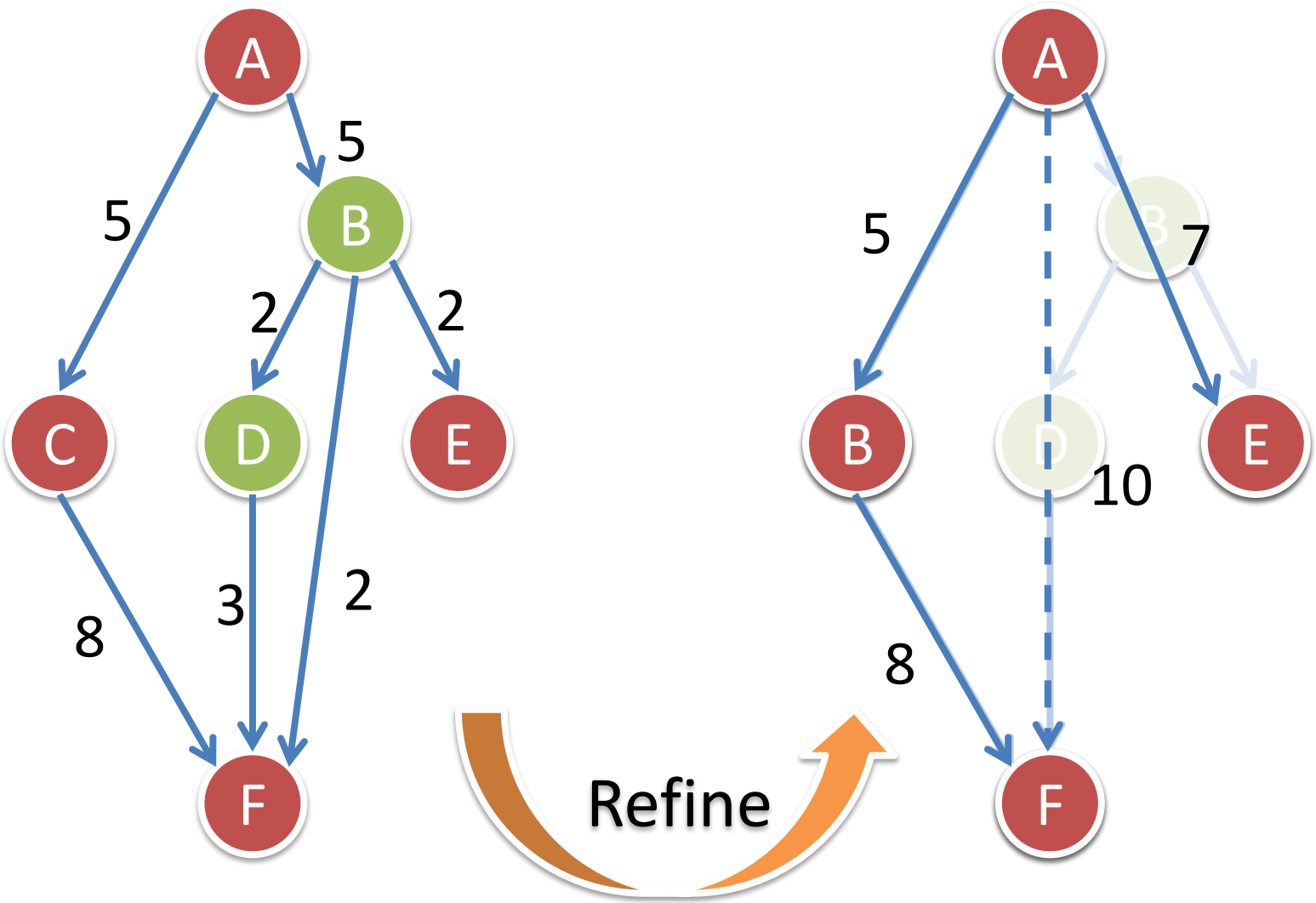


- Build new CDFG that only contains critical-operations
- Schedule the newly built CDFG
- Schedule non-critical operations

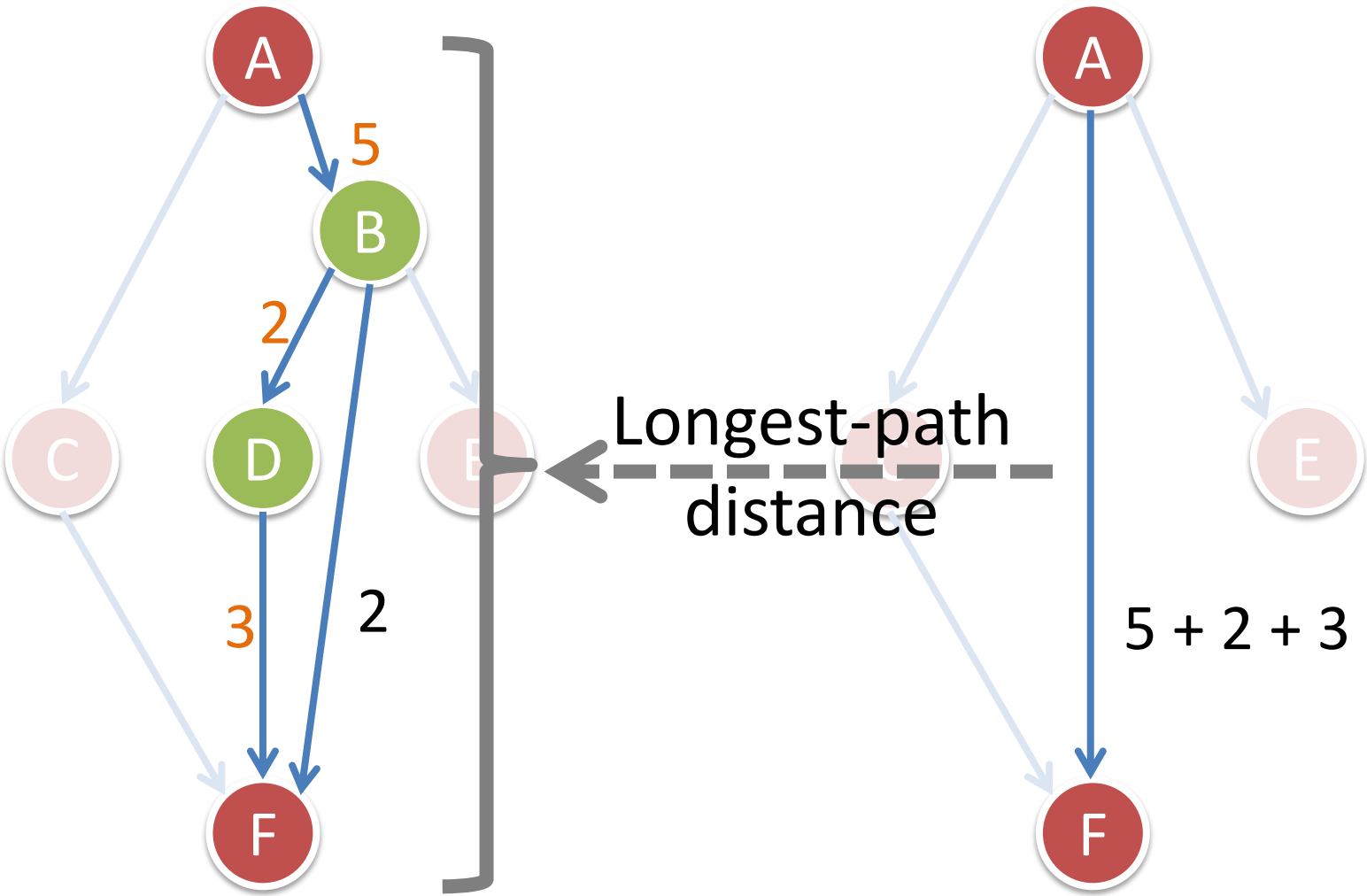
CDFG Refining

- Given:
  - CDFG
  - Criticality partitioning constraints
    - In terms of area of the functional unit
- Goal:
  - Build a critical-operation-only CDFG
  - Preserve the constraints between critical operations

CDFG Refining Example

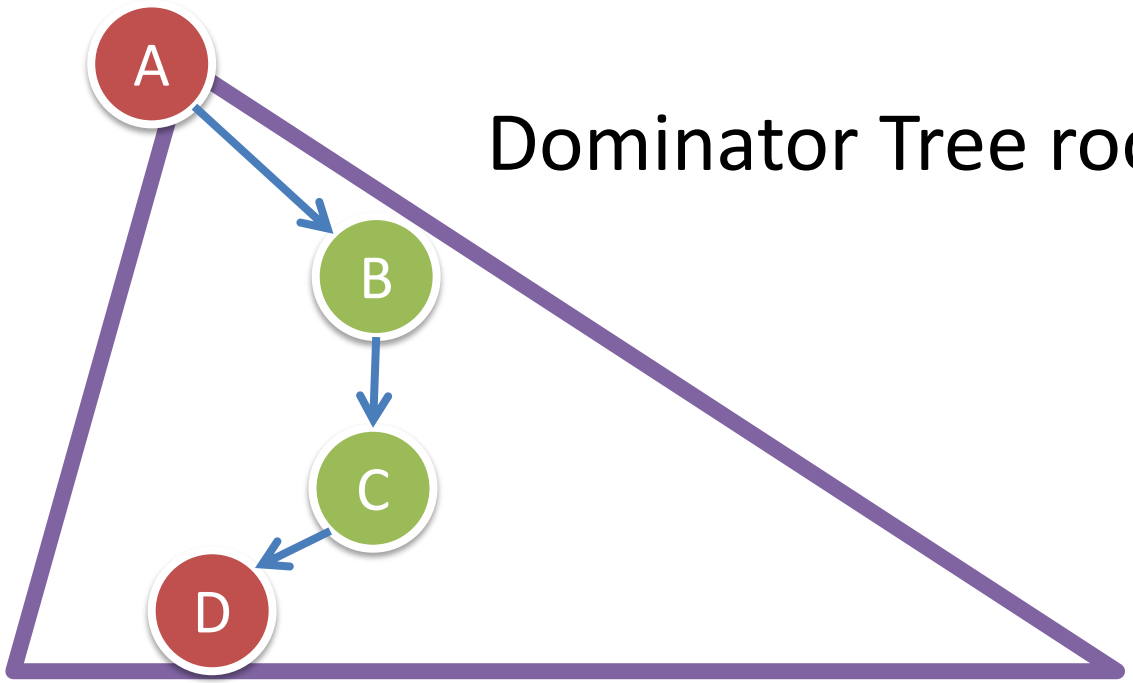


CDFG Refining Example



CDFG Refining Requirement

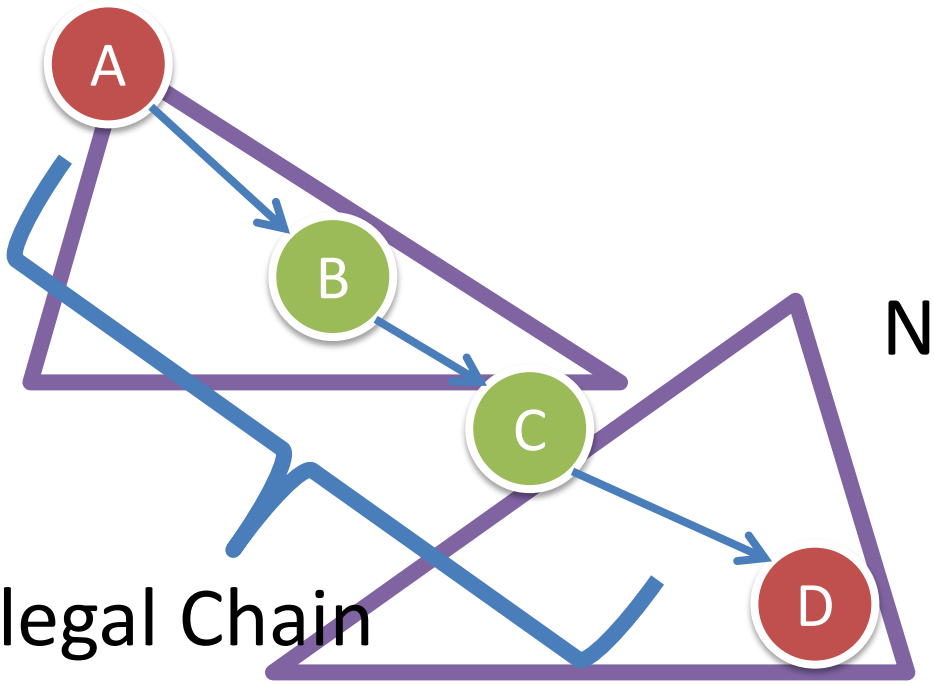
- The source of the noncritical chain should dominate the whole chain



Dominator Tree rooted A

CDFG Refining Requirement

- The source of the noncritical chain should dominate the whole chain

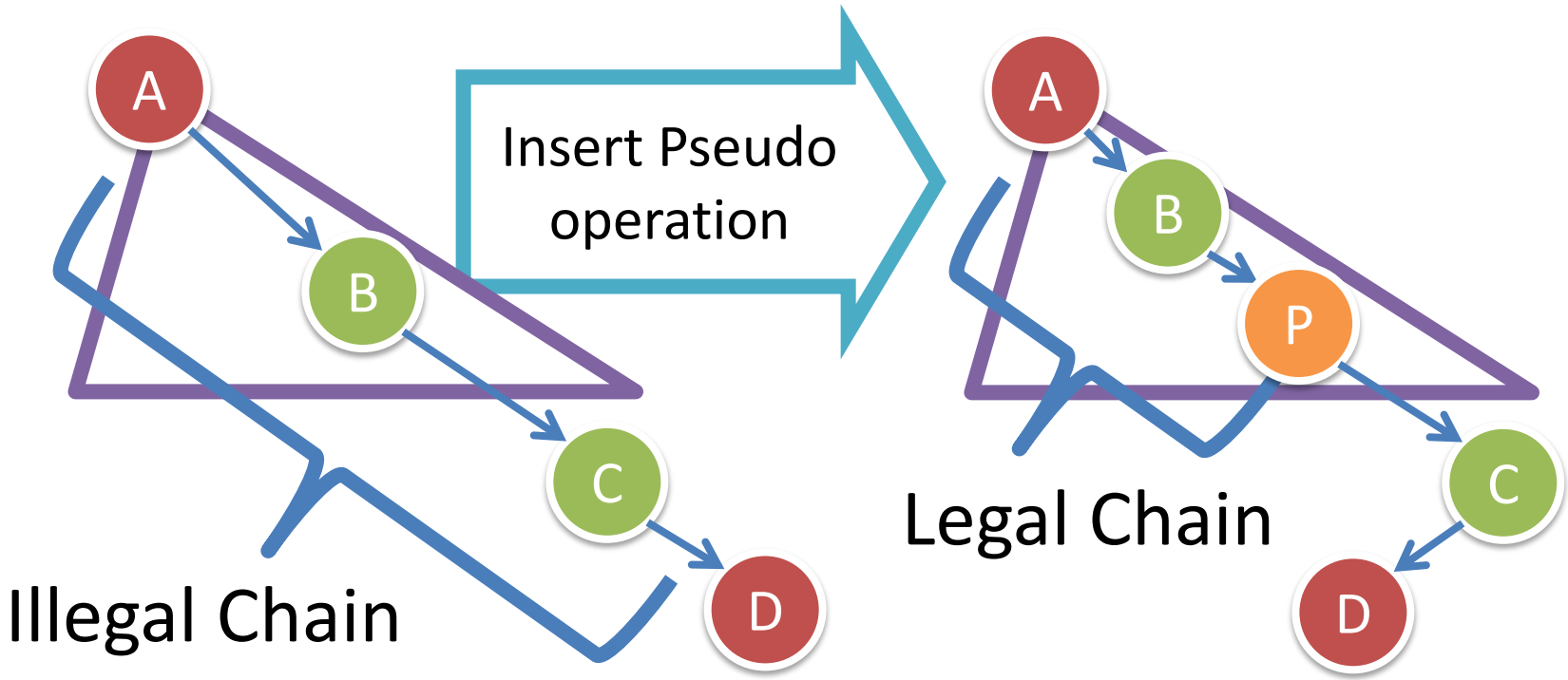


Not dominated by A

Illegal Chain

CDFG Refining Requirement

- The source of the noncritical chain should dominate the whole chain



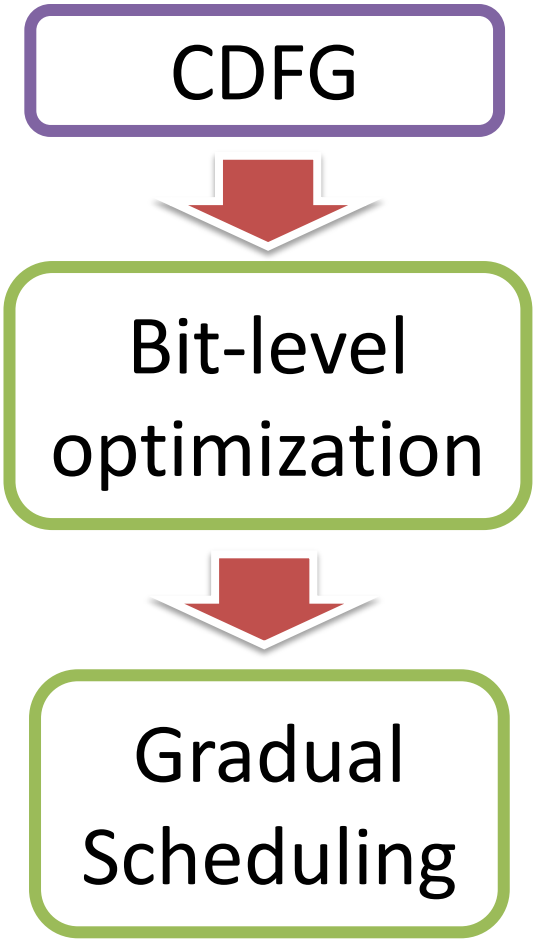
PROBLEM SIZE REDUCTION EXPERIMENT

Size of Refined CDFG vs. Partitions

- Size of Refined CDFG depends on the Partition

- Show the % of critical operations for:
  - **Chained**: load/store, gotos; Minimal Set.
  - **S16M16**: Including **Chained**, mults and shifts bigger than 16 bits
  - **All**: Including **Chained**, all arithmetic, shifts and comparisons; Maximal Set.

Experimental Setup

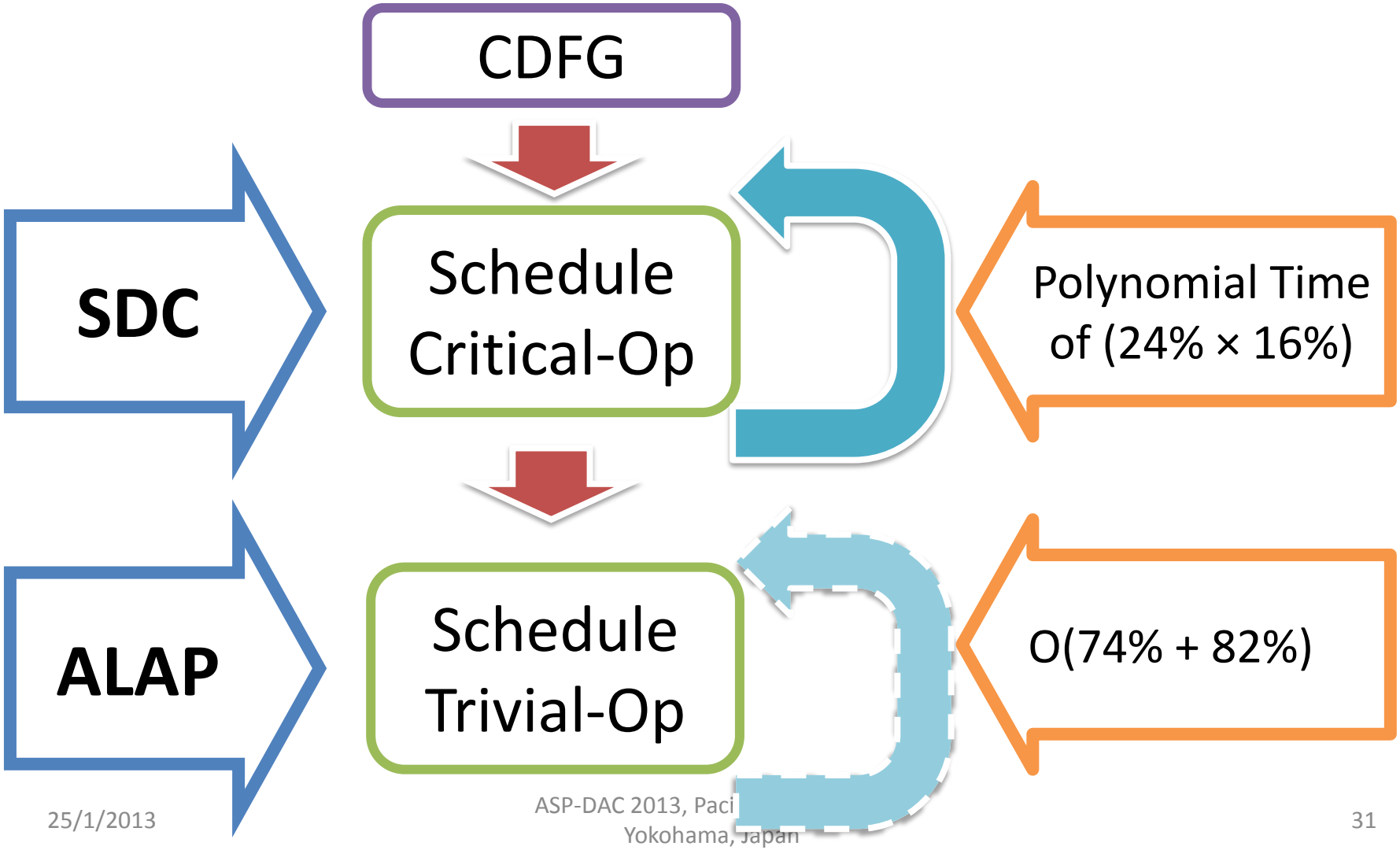


- LLVM-based HLS framework
- Targeting Altera Cyclone-II FPGA (available on DE2-70 board)
- Run on CHStone benchmarks

Size of the Refined CDFG (Geomean)

	Refined CDFG		Partial-Scheduled CDFG	
	|N|	|E|	|N|	|E|
Chained	12%	8%	87%	91%
S16M16	13%	9%	86%	90%
All	24%	16%	74%	82%

Problem Size Reduction



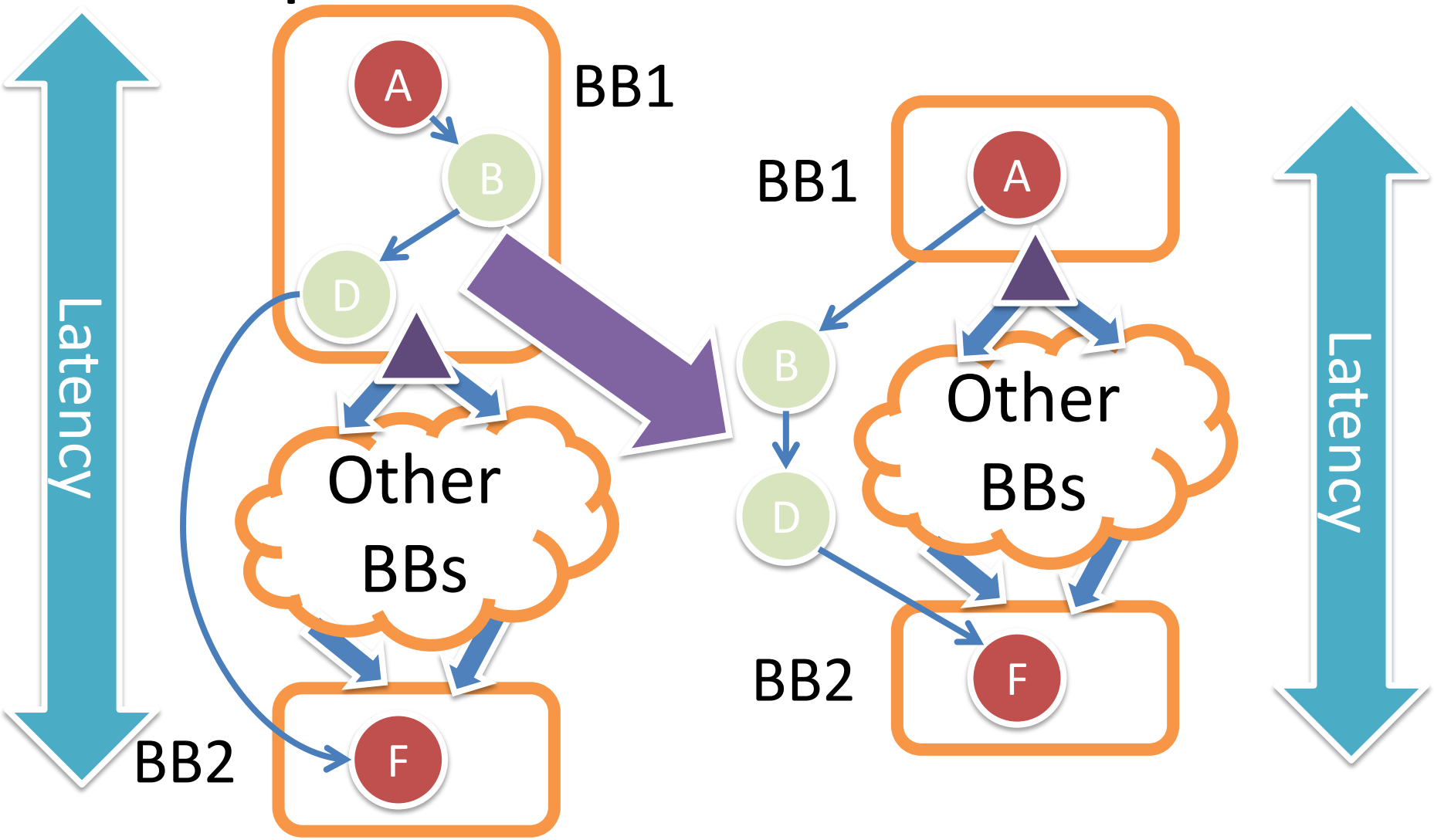
EXPLOITING GLOBAL PARALLELISM

BY THE GRADUAL SCHEDULING FRAMEWORK

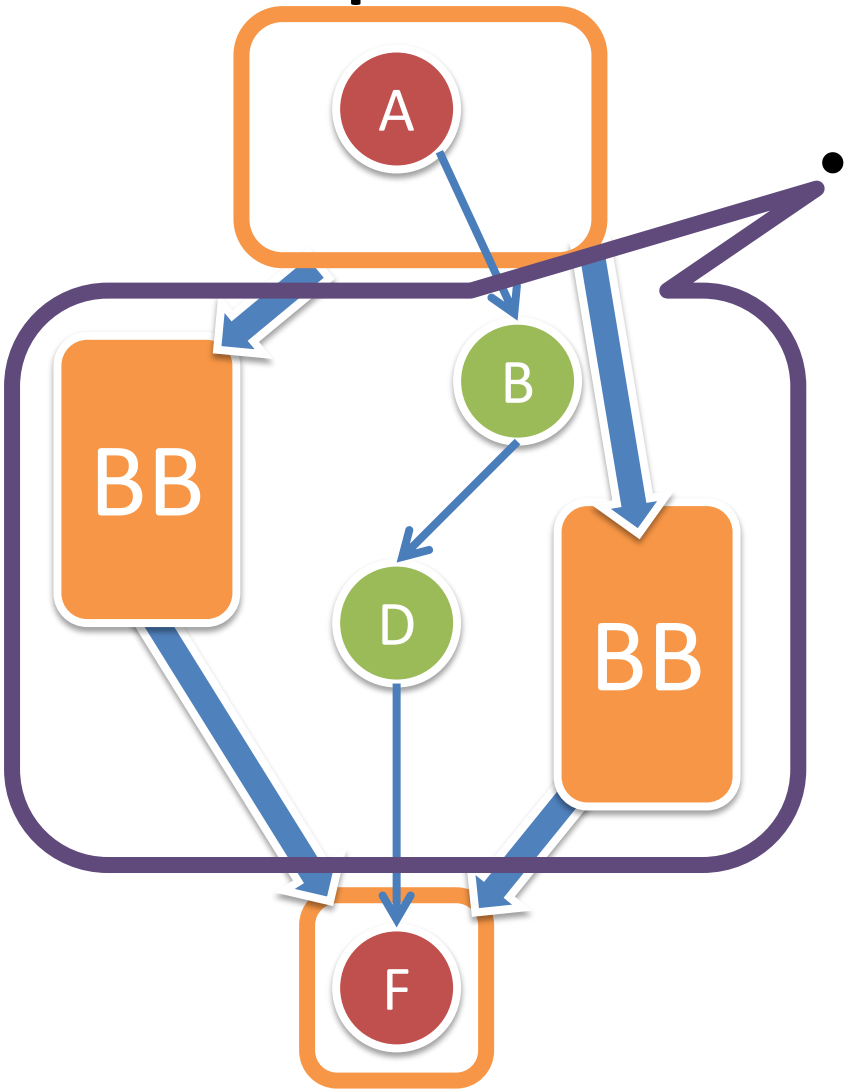
Parallelism Exploiting Techniques

- Hyper-block Formation
  - Build a bigger BB by if-conversion, may introduce lots of idle states
- (Traditional) Global Code Motion
  - Move the operations across BBs, but still restrict them inside a BB
- This work: No need to restrict non-critical operations inside a BB

“Implicit” Global Code Motion

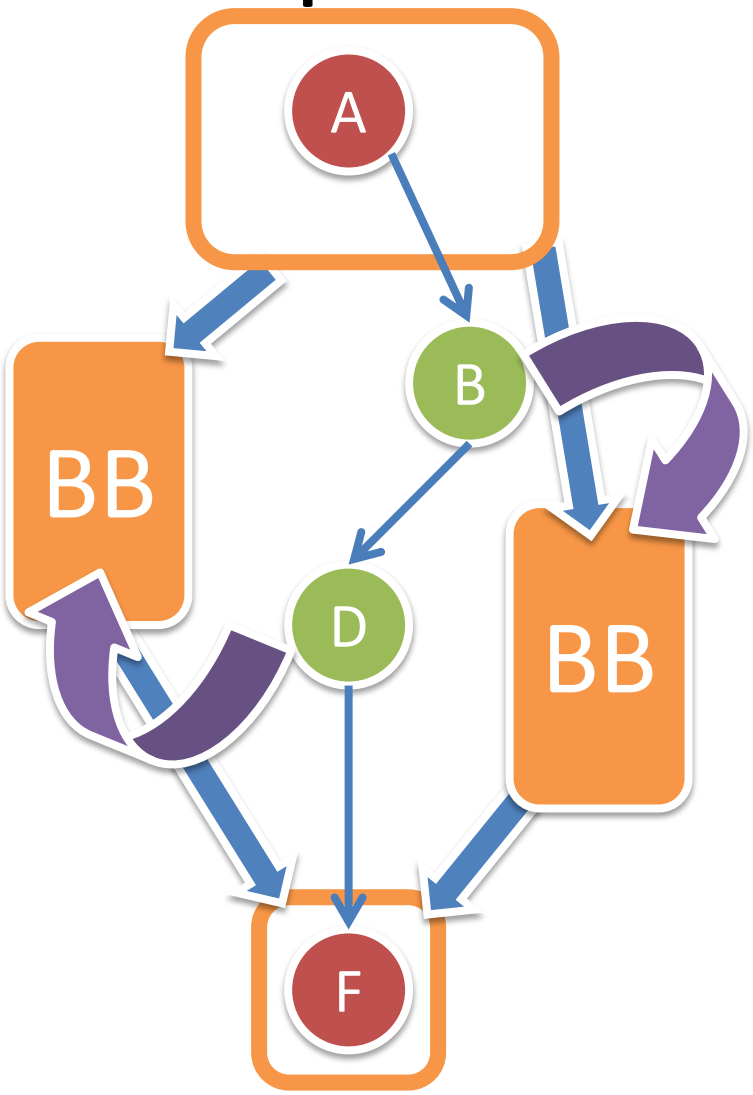


Implicit Global Code Motion



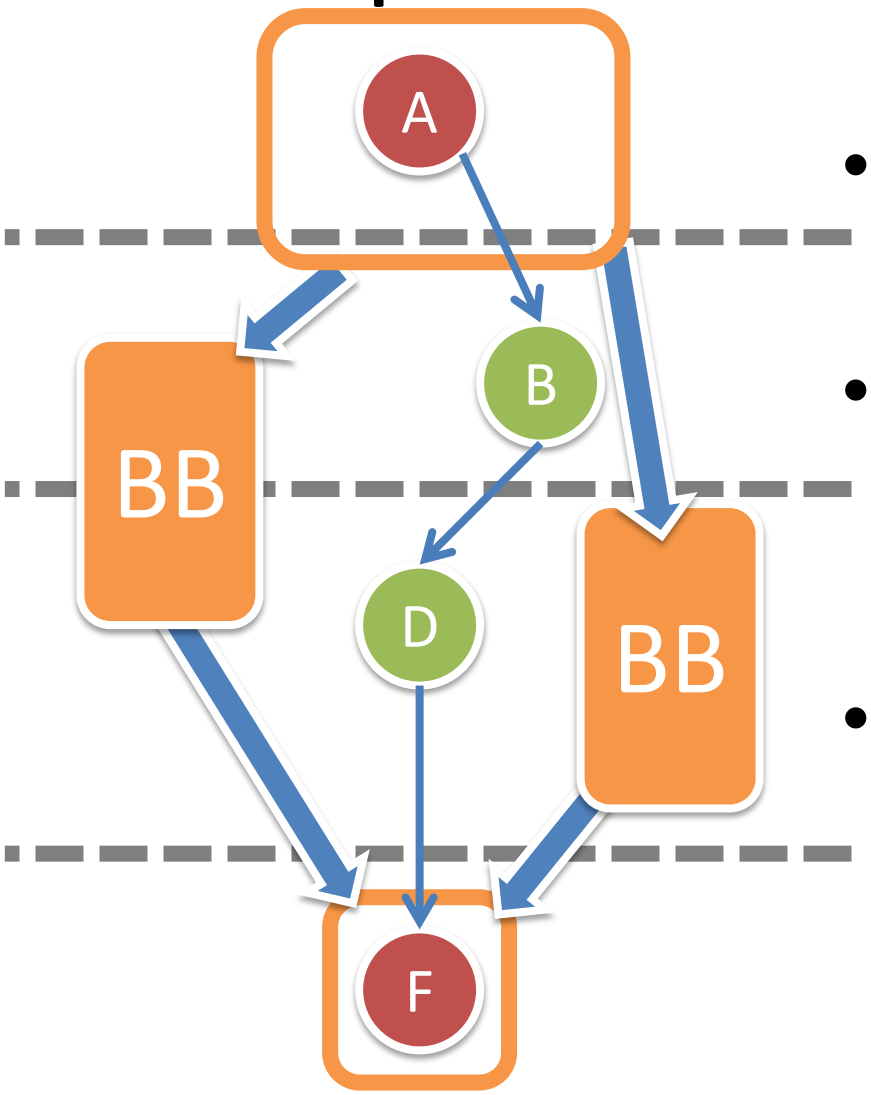
- Execute in parallel with other BB
  - Not necessarily restricted in a specific BB

Implicit Global Code Motion



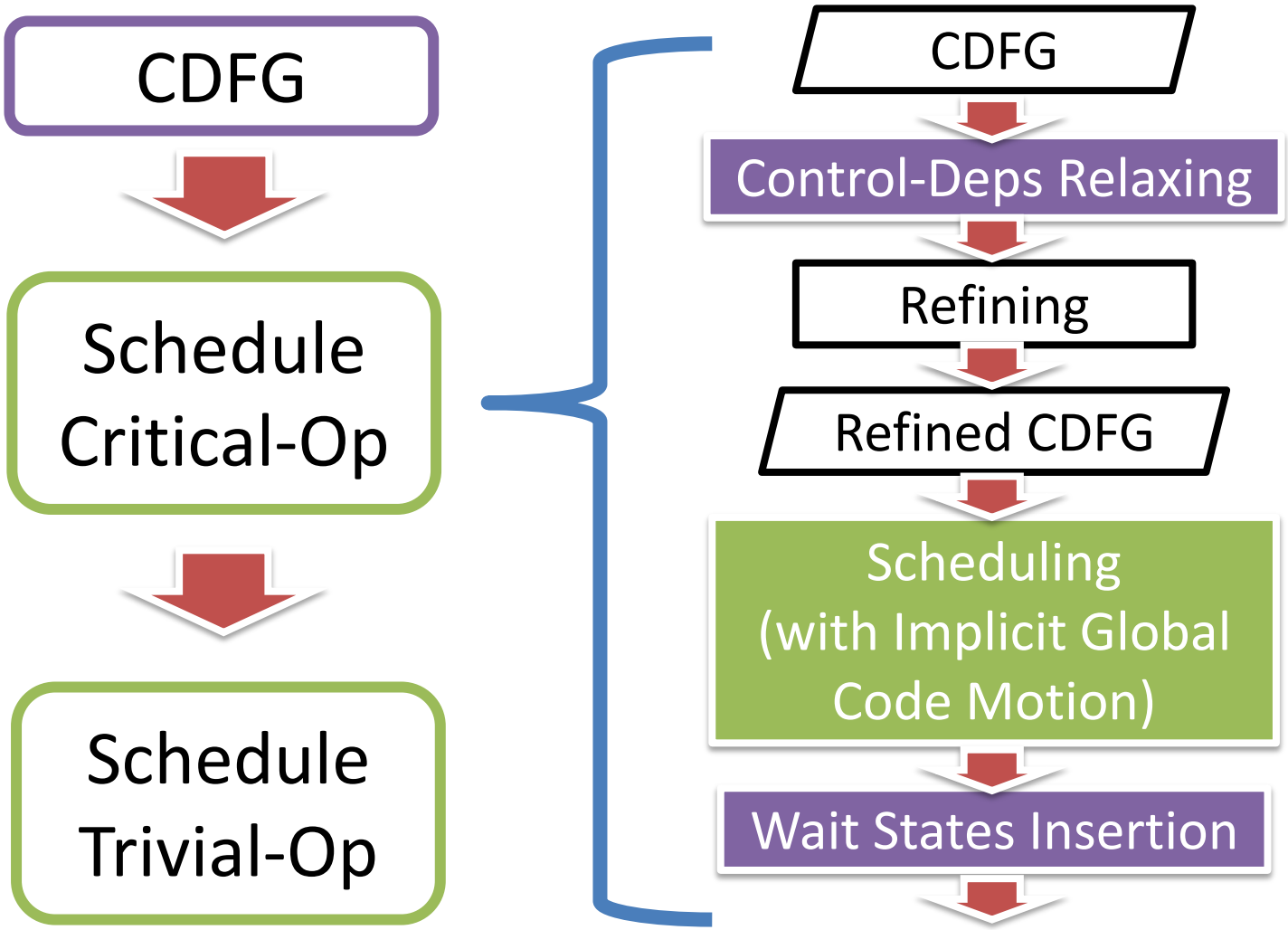
- Execute in parallel with other BB
- No need to duplicate the operations into BBs in each path
  - Confuse FU binding
  - Too many paths

Implicit Global Code Motion

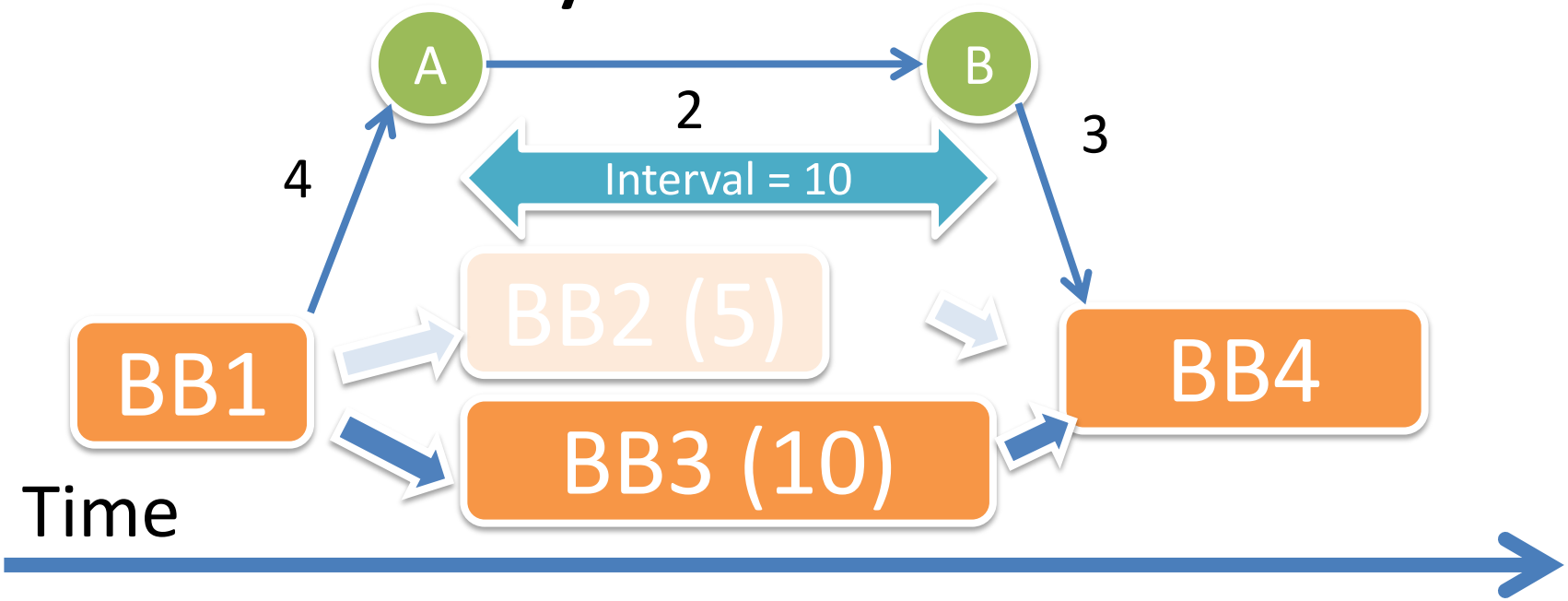


- Execute in parallel with other BB
- No need to duplicate the operations into BBs in each path
- Completely integrated with scheduling algorithm

Implicit Global Code Motion

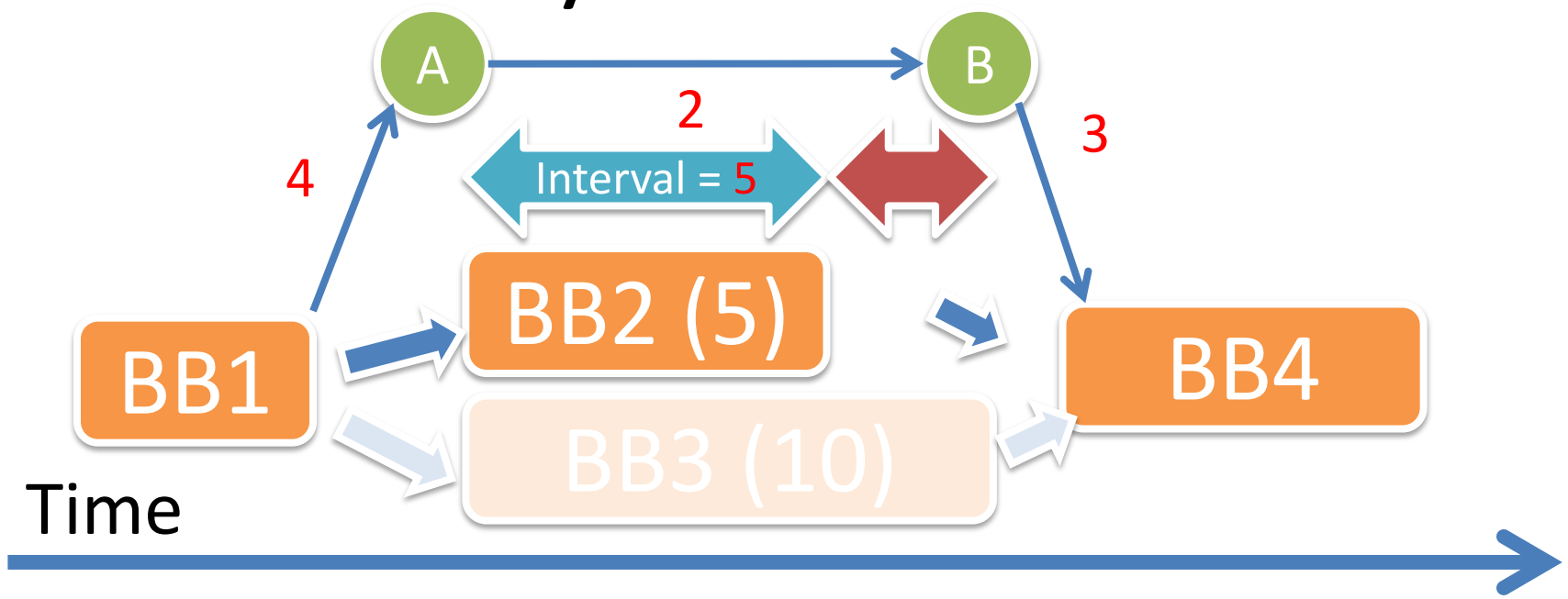


Why wait states?



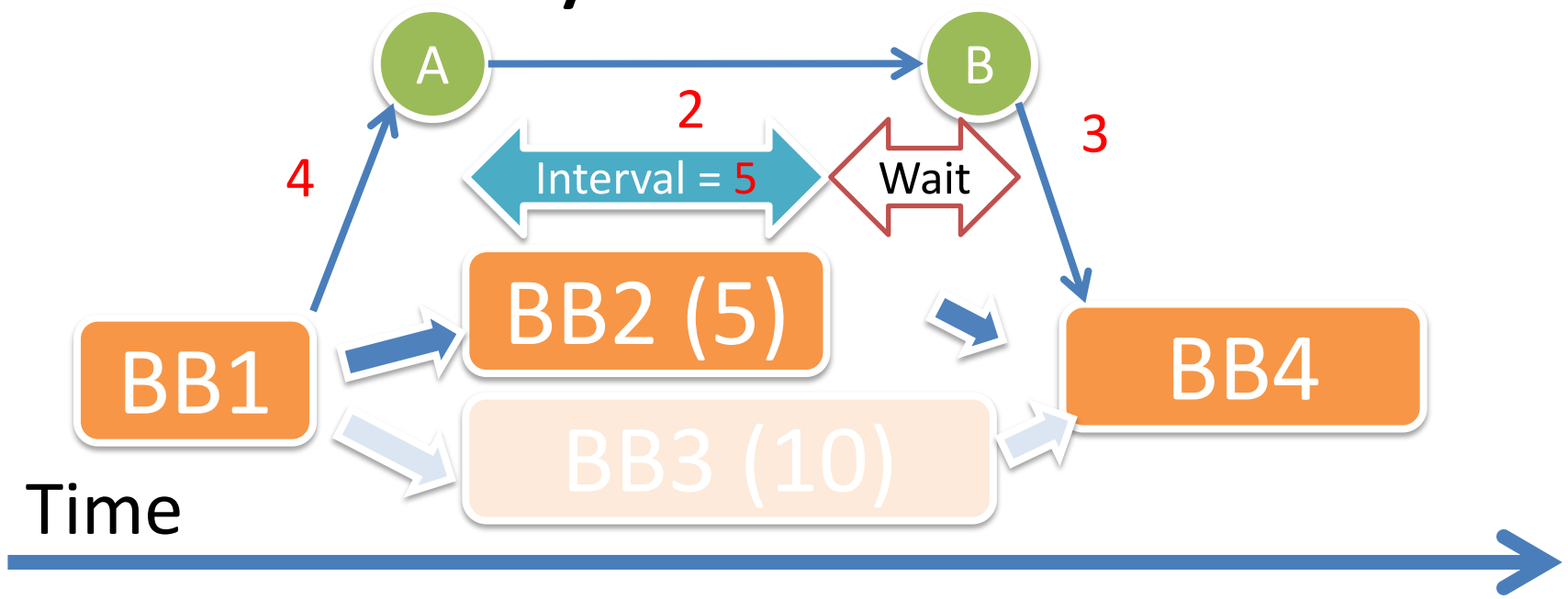
- All cross BB chains are scheduled according to the longest-path in the CDFG
  - Without knowing the deps between BBs are conditional

Why wait states?



- But a shorter path maybe taken
- The latency of cross BB chains are NOT preserved
  - 4 + 2 + 3 <= 5

Why wait states?

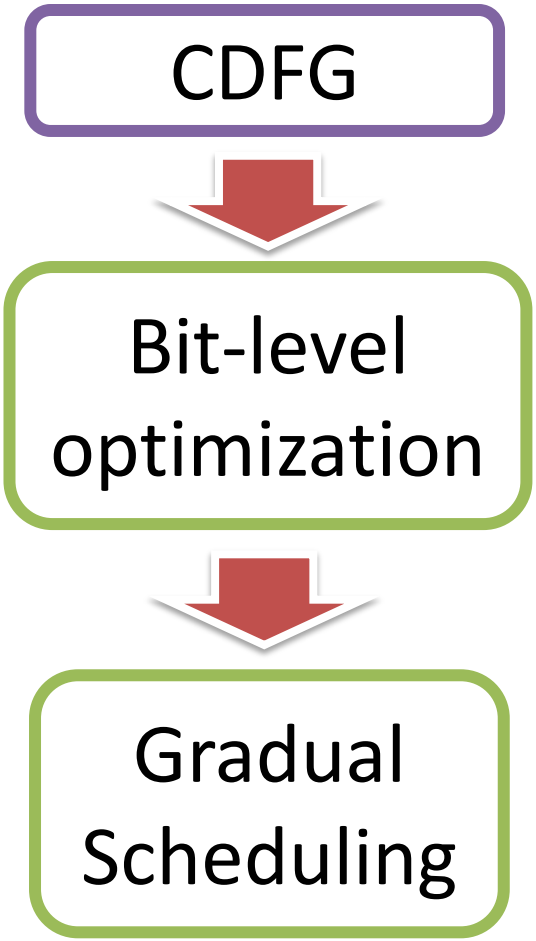


- But a shorter path maybe taken
- The latency of cross BB chains are NOT preserved
  - 4 + 2 + 3 <= 5 + [wait states]

Wait States Insertion

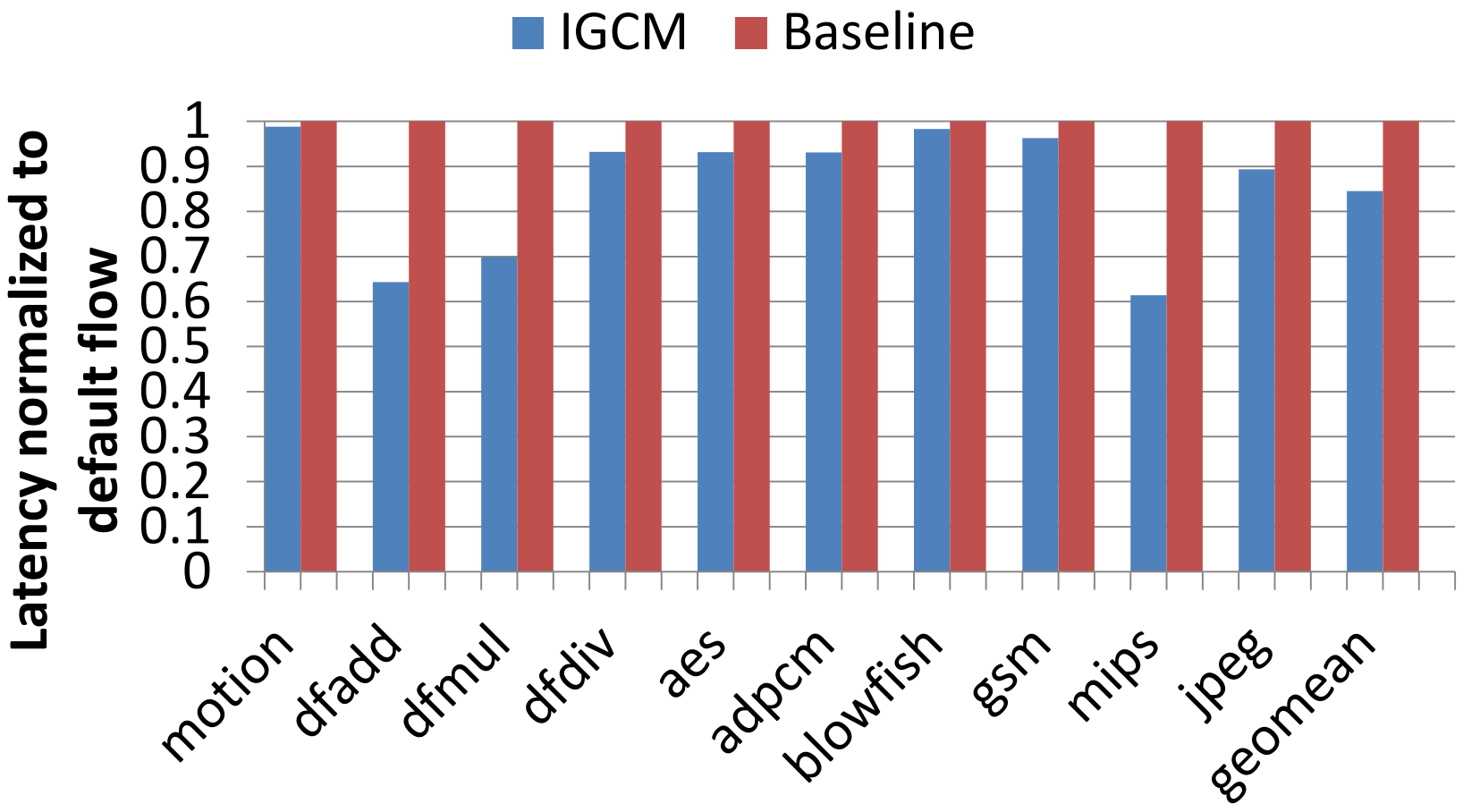
- Fix the cross BB constraints
- Based on Shortest Path Distance
  - #States = Expected SPD - Actual SPD
- Wait states are inserted as late as possible

Experimental Setup - Reminder

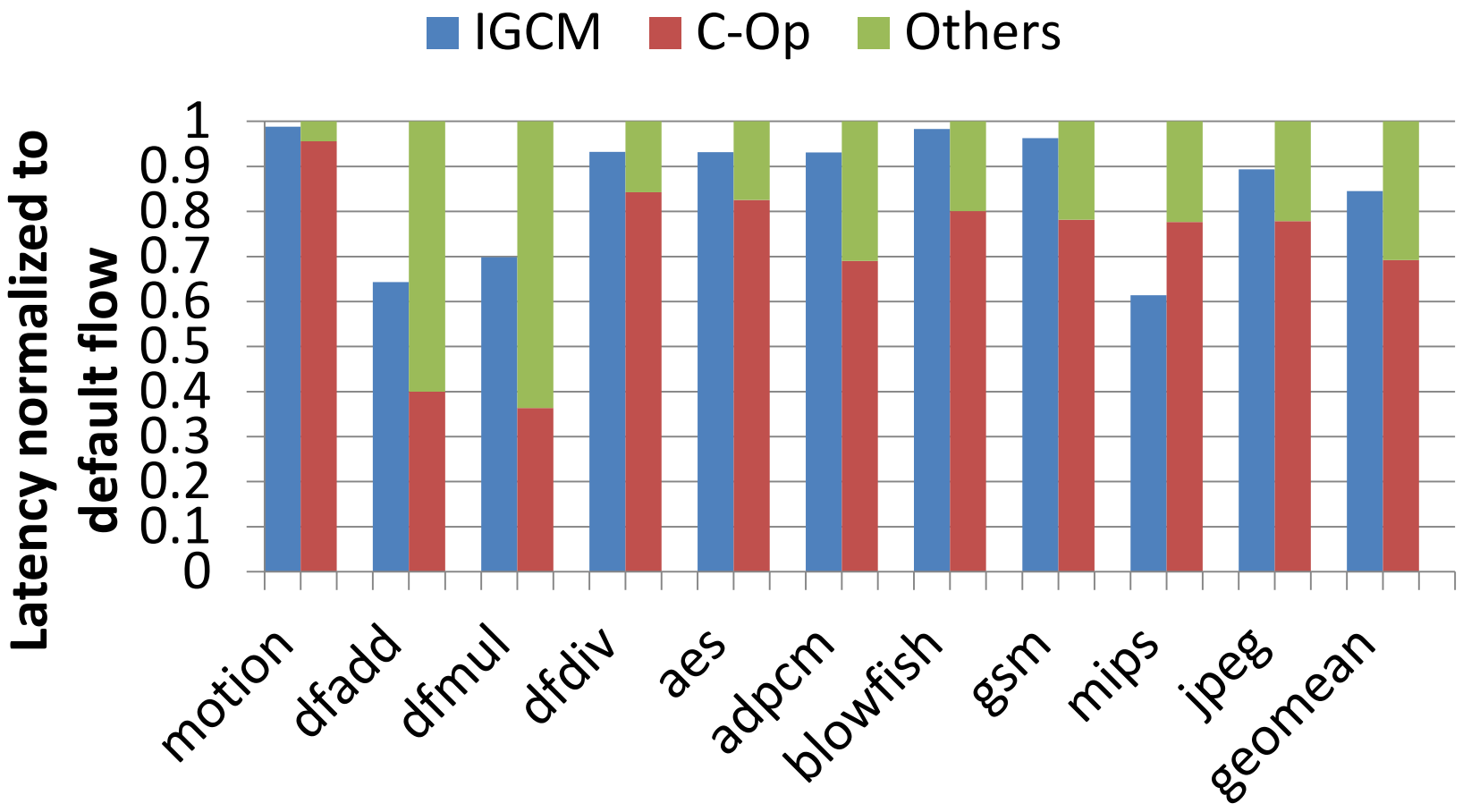


- Evaluate the latency reduction by Implicit Global Code Motion (IGCM)
- Run on CHStone benchmarks

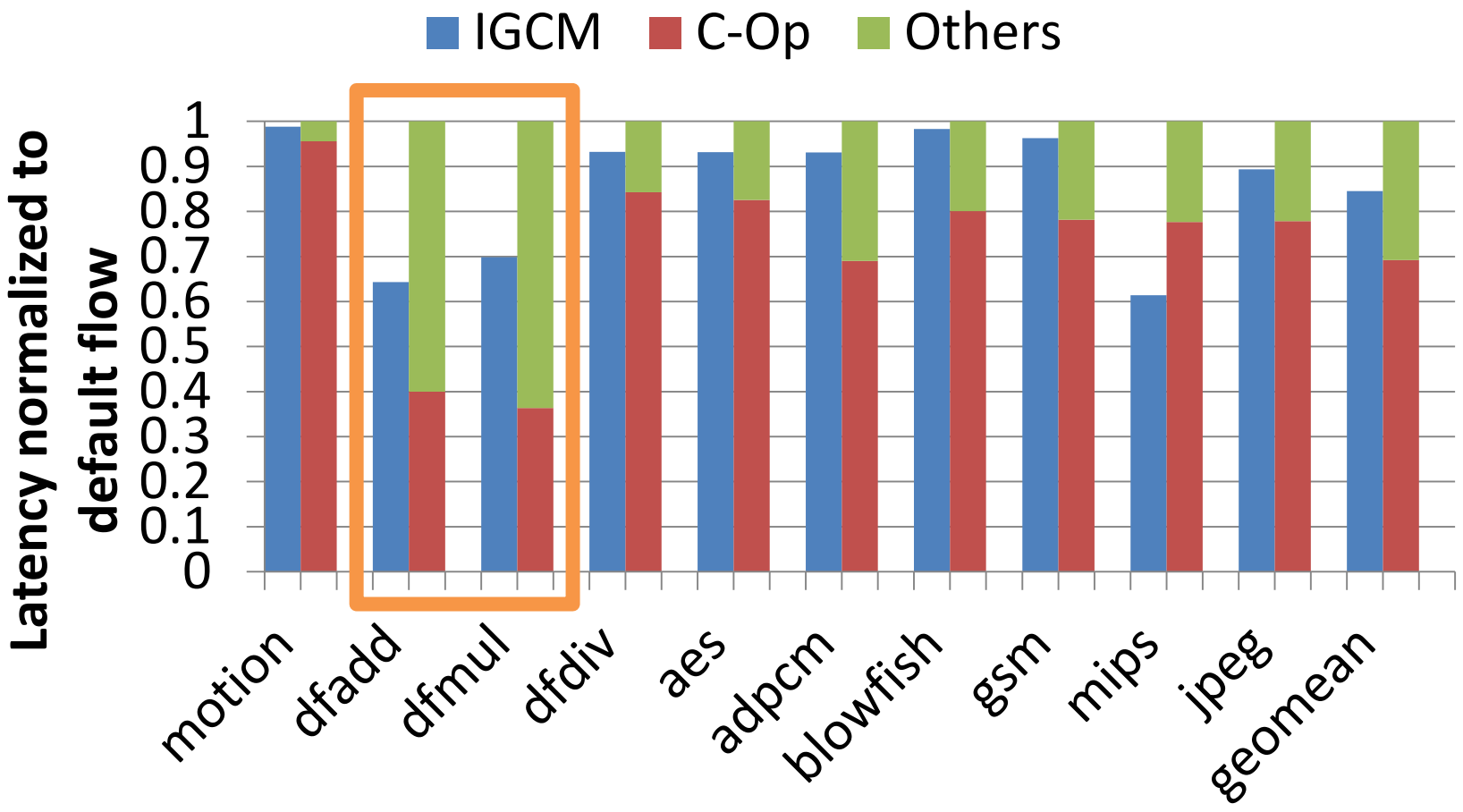
Latency Reduction by IGCM



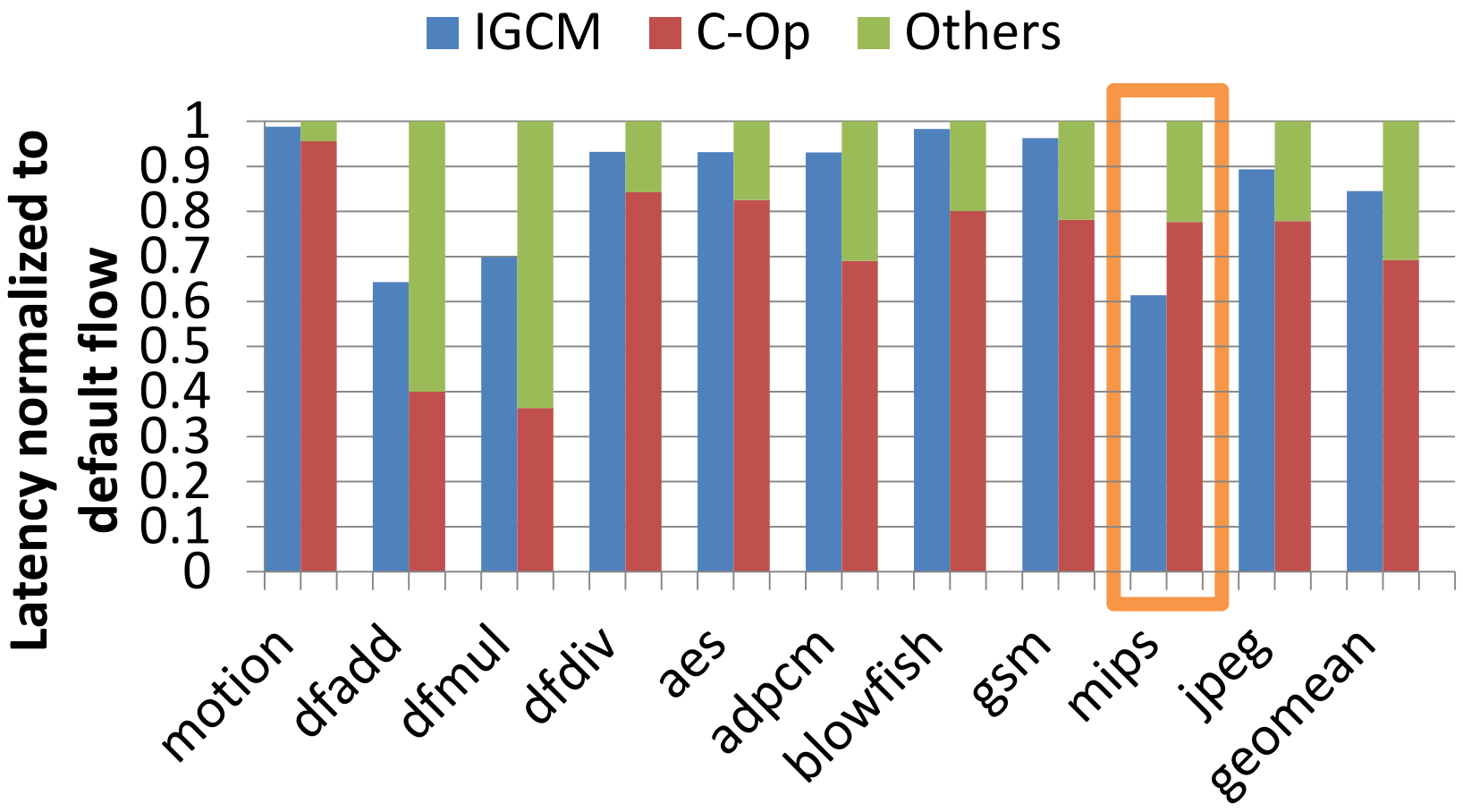
Latency Reduction by IGCM



Latency Reduction by IGCM



Latency Reduction by IGCM



Summary

- Gradual scheduling framework
  - Schedule the critical/noncritical operations separately
- Reduced the problem size of scheduling
  - Size reduced to **24%** of Nodes and **16%** of Edges
  - Corresponds to **96.8%** reduction in SDC scheduling
- Exploited cross-BB parallelism
  - Reduced run-time up to **37.7%** and **15.5%** on average

Thanks and Acknowledgement

- Thanks for the EDA group in SYSU for participating the project:
  - Q. Liu, J. Li, D. Chen, Z. Wang
- Thanks for helpful discussions with colleagues at ADSC:
  - K. Rupnow, S. Gurumani, T. Satria
- Thanks for listening!

Refining Time Less Than 0.05s!