Analysis for Supporting Real-Time Computer Vision Workloads using OpenVX on Multicore+GPU Platforms

Kecheng Yang
Glenn A. Elliott
James H. Anderson
Dept. of Computer Science
UNC-Chapel Hill

Motivation

- In industry, vision-based sensing through cameras is emerging.

- In the automotive industry, it is used to support features, such as
  - pedestrian detection,
  - automatic lane-keeping,
  - adaptive cruise control,
  - ultimately, full autonomy.

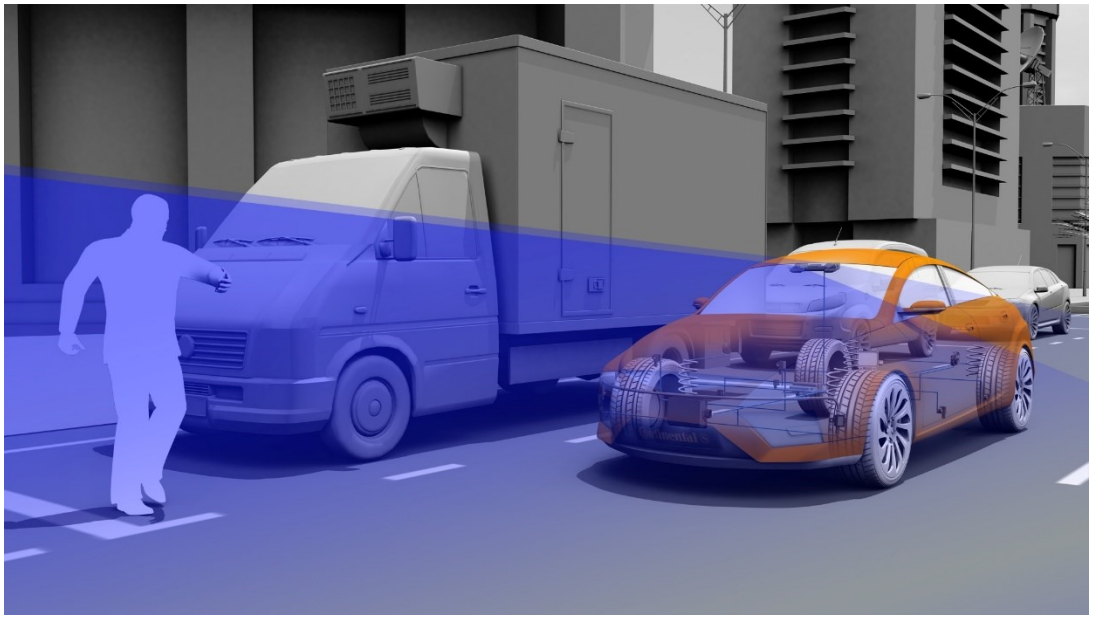
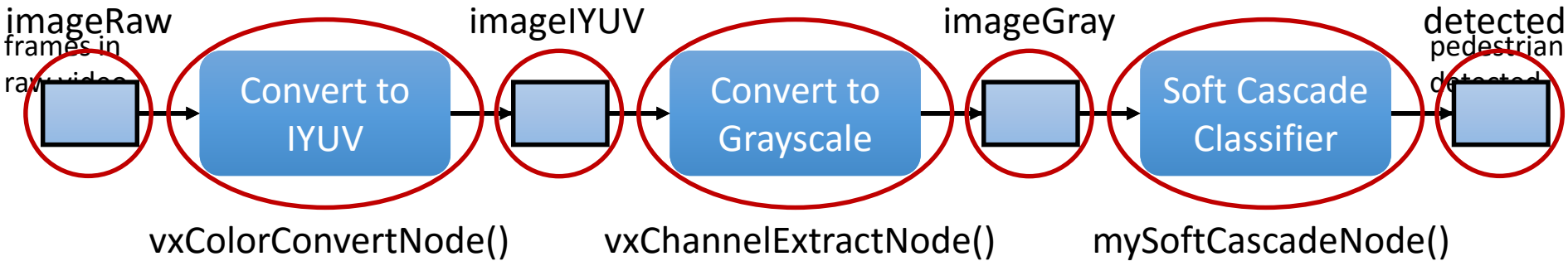


figure source: http://www.automotiveit.com/continental-develops-stereo-camera-to-improve-car-safety-systems/news/id-002720

OpenVX

- Computer vision algorithms are commonly expressed using dataflow graphs.
- A standard computer vision API – OpenVX – has been created that allows for the specification of such graphs.

Node dependencies (i.e., edges) are derived from how data objects are bound to the inputs and outputs of nodes.



Each node is a basic operation in computer vision algorithms.

A given basic operation has a set of well-defined inputs and outputs, and may be performed on either CPU or GPU.

OpenVX v.s. Real-Time

- OpenVX has a simple execution model that eases the development of computer vision applications on heterogeneous platforms.
- However, OpenVX does not fit any real-time scheduling model and lacks any framework for real-time analysis.
- In a recent paper[1], our group developed a new OpenVX implementation that extends a current OpenVX implementation by NVIDIA.
- Our new OpenVX implementation overcomes several problems that handicap real-time analysis.

[1] G. Elliott, K. Yang, and J. Anderson, "*Supporting real-time computer vision workloads using OpenVX on multicore+GPU platforms*," RTSS 2015, to appear.

OpenVX v.s. Real-Time

Sporadic DAGs?

Existing OpenVX implementation	Our extension ([1] provides details)
No notion of repeating (periodic or sporadic) task	The source node of each graph is invoked sporadically
Does not define a threading model	Each node is assigned a dedicated thread
Requires a graph to execute end-to-end before it may be re-executed	Graph execution can be pipelined

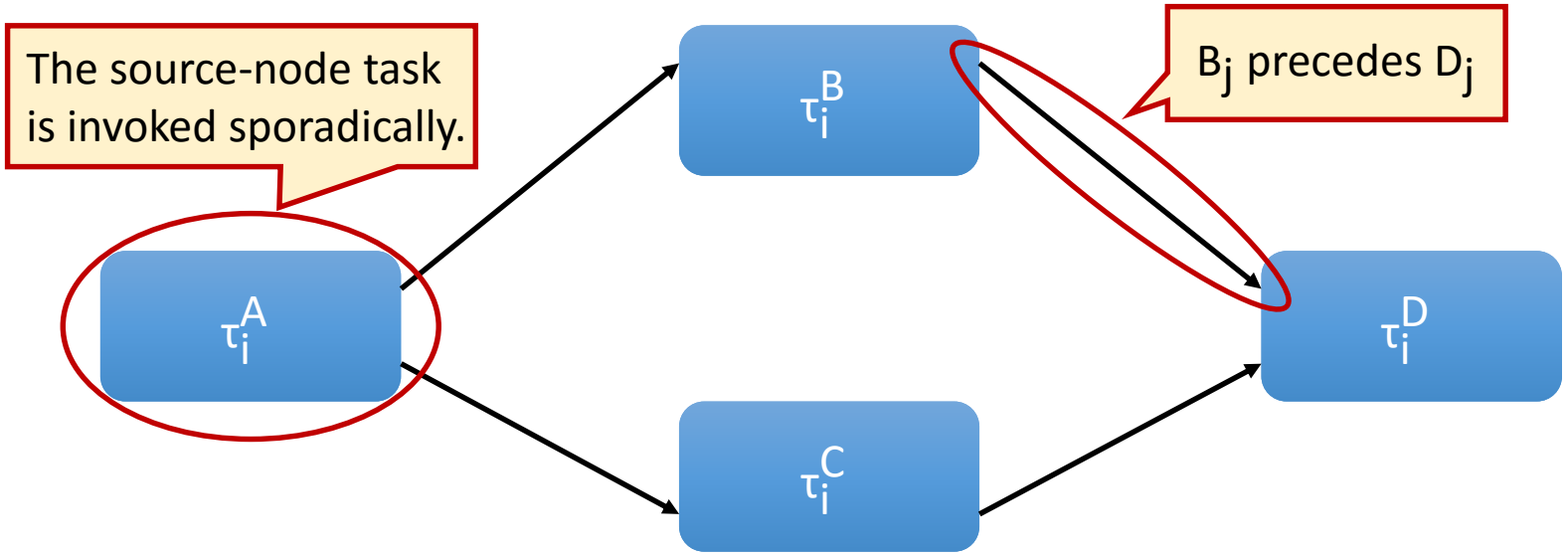
GPU accesses are managed by GPUSync [2].

GPUs are treated as shared resources and managed by real-time locking protocols. Priority-inversion blocking times are analytically modeled as CPU computation time through suspension-oblivious analysis.

[1] G. Elliott, K. Yang, and J. Anderson, "*Supporting real-time computer vision workloads using OpenVX on multicore+GPU platforms*," RTSS 2015, to appear.
[2] G. Elliott, "*Scheduling of GPUs, with applications in advanced automotive systems*," Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2015.

Sporadic DAG

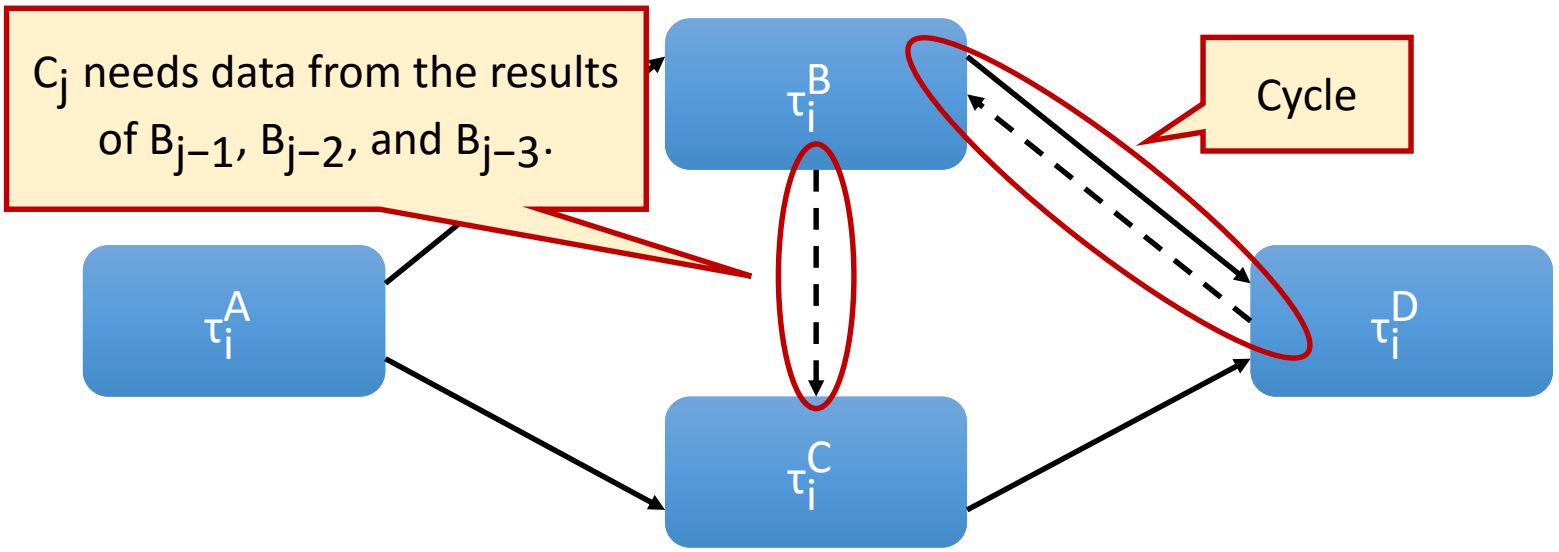
- Each graph is a Directed Acyclic Graph (DAG).
- τ_i^A denotes the task that implements node A in the i^{th} graph.
- A_j denotes the j^{th} invocation (job) of τ_i^A.*
- Each edge denotes a producer/consumer precedence constraint in the same invocation of the graph.



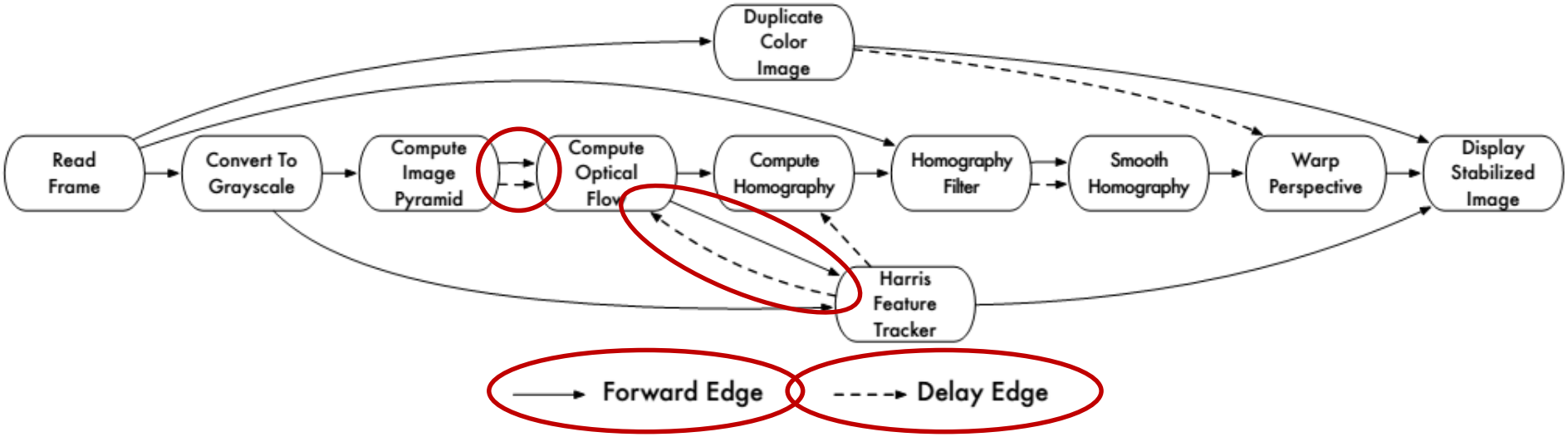
*In the paper, it is denoted as $J_{i,j}^A$.

OpenVX Graphs v.s. Sporadic DAGs

- In an OpenVX graph, two kinds of edges may exist.
- Some are the same as edges in DAGs. They are called forward edges.
- Another category of edges, denoted by dotted arrows, may exist, called delay edges.
- Each delay edge denotes a precedence constraint pertaining to prior invocations of the graph.
- Delay edges may cause cycles!

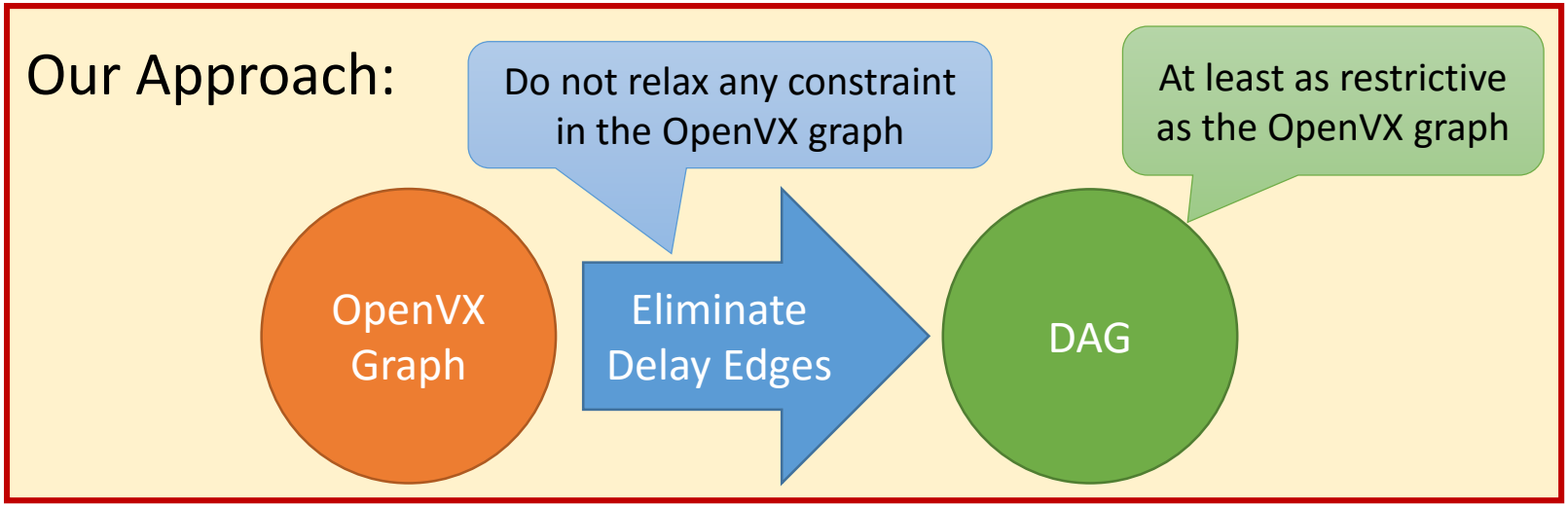


Example: Video Stablilization



From OpenVX Graphs to Sporadic DAGs

- Prior work[3][4] has shown that, with the sporadic DAG model and under global EDF scheduling, an end-to-end latency bound can be established for each DAG.
- However, due to the existence of delay edges and potential cycles caused by delay edges, this result cannot be applied directly.



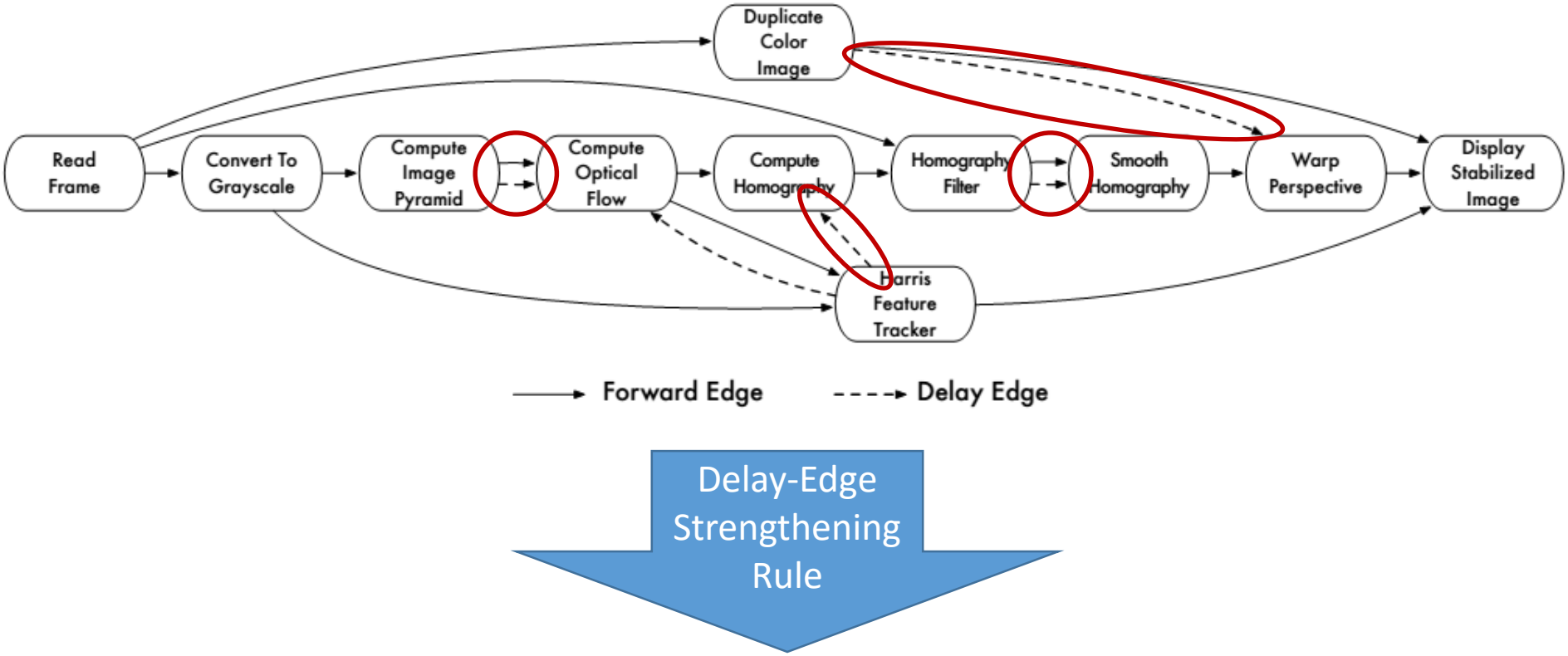
[3] C. Liu and J. Anderson, "Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss," in RTSS, 2010.
[4] G. Elliott, N. Kim, C. Liu, and J. Anderson, "Minimizing response times of automotive dataflows on multicore," in RTCSA, 2014.

Delay-Edge Strengthening Rule

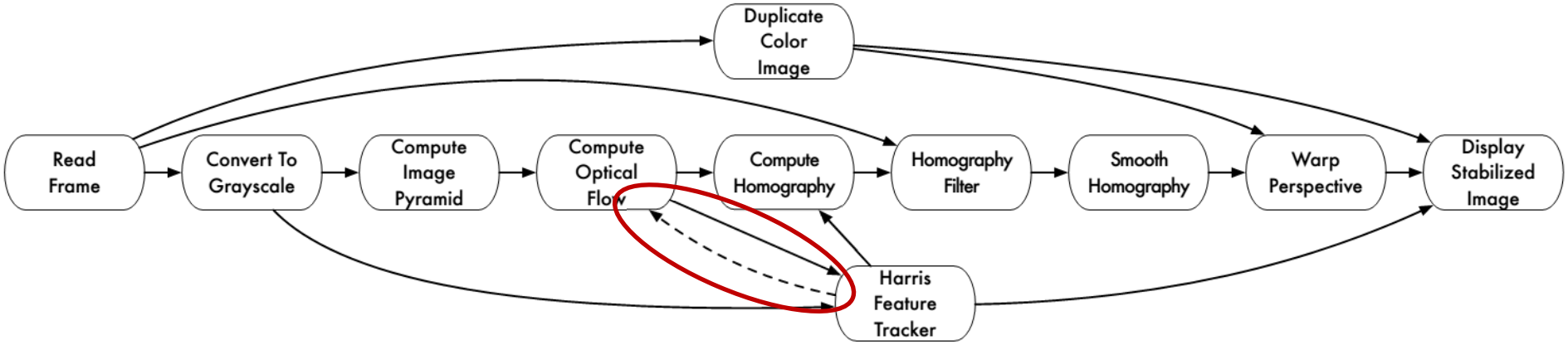
- Because each node is modeled as a sequential sporadic task, the completion of A_j implies the completions of all prior jobs of the same task (including A_{j-1}, A_{j-2},…).
- Therefore, between two nodes, a forward edge is always a more restrictive constraint than a delay edge.
- That is, 

.

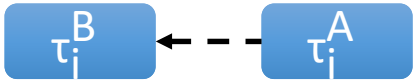
Replace a delay edge by a forward edge if it is not a part of a cycle.



Delay-Edge Strengthening Rule

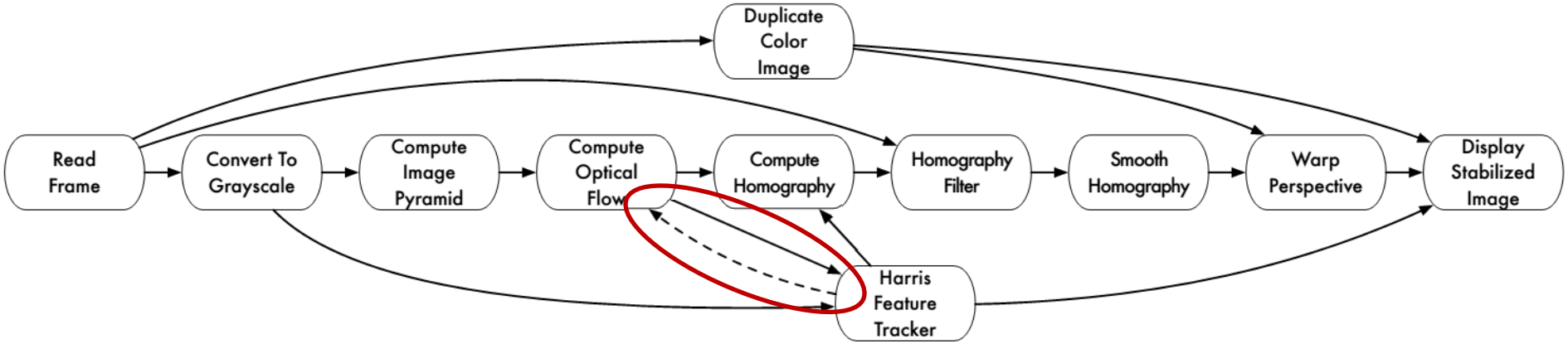


Delay-Edge Dropping Rule

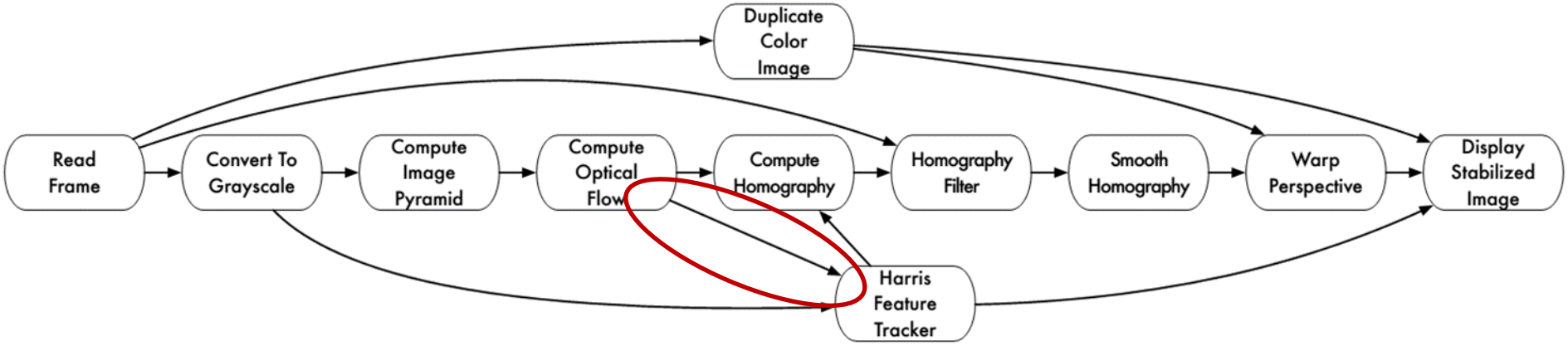
- Applies if the application code is adjustable, so that the dependency associated with the delay edge is not based on the immediate history, but rather on history "further in the past".
- E.g., 

means that B_j needs data from the results of A_{j-k}, A_{j-k-1}, A_{j-k-2}, instead of A_{j-1}, A_{j-2}, A_{j-3}.
- A safe k can be computed such that the end-to-end latency bound guarantees that A_{j-k} completes before B_j releases.
- Thus, in this case, the delay edge is not constraining and therefore can be ignored in terms of scheduling.

Drop a delay edge if the application code is adjustable.

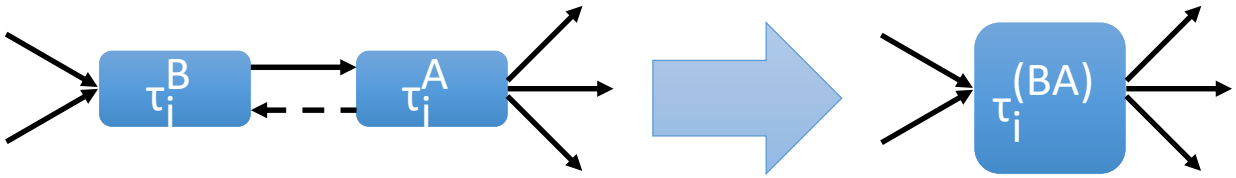


Delay-Edge Dropping Rule

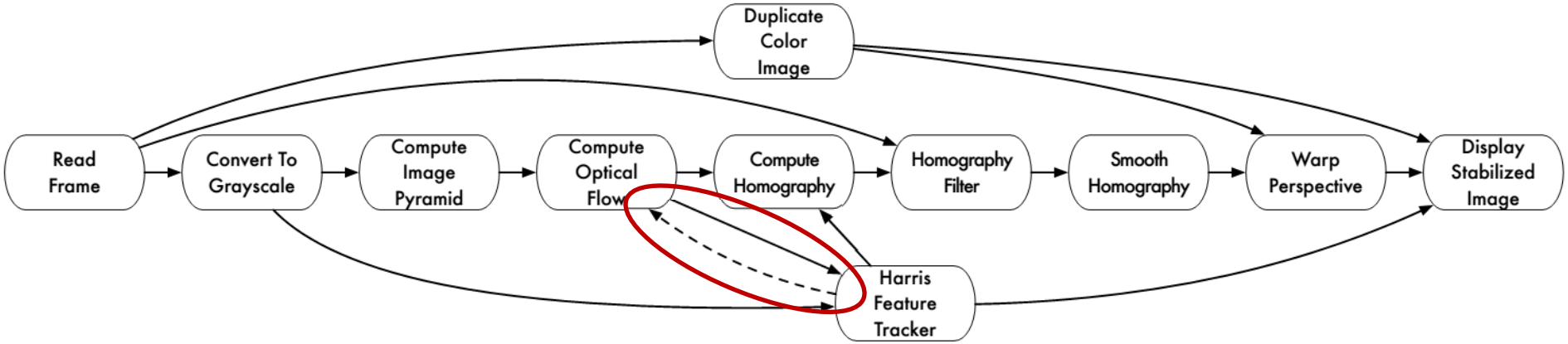


Super-Node Creation Rule

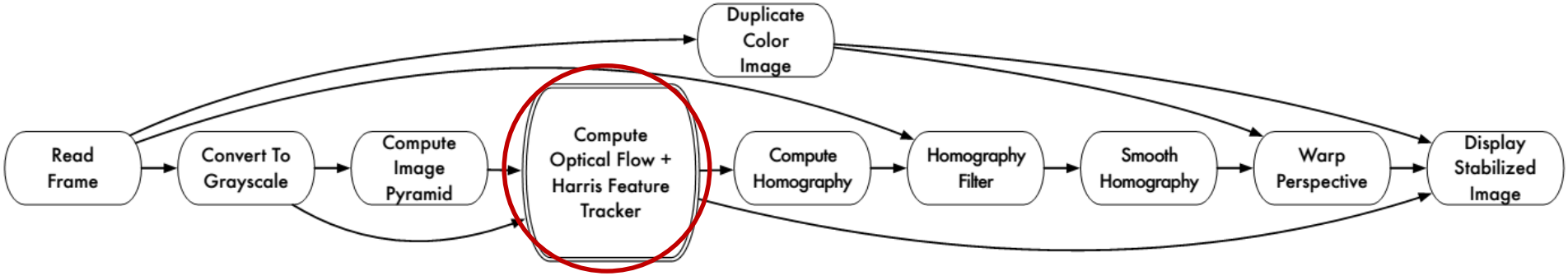
- Applies if adjusting the application code is infeasible.
- E.g., $\tau_i^B \leftarrow\!\!- \tau_i^A$ means that B_j must use data from the results of $A_{j-1}, A_{j-2}, A_{j-3}$
- A super-node is created, and everything in the cycle is serialized.
- I.e., $B_{j-3}, A_{j-3}, B_{j-2}, A_{j-2}, B_{j-1}, A_{j-1}, B_j, A_j$, … execute sequentially.



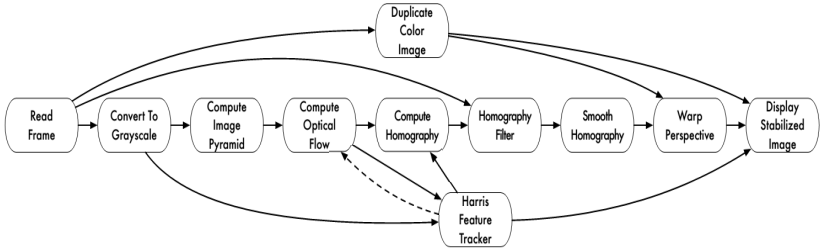
Create a super-node if the application code is not adjustable.



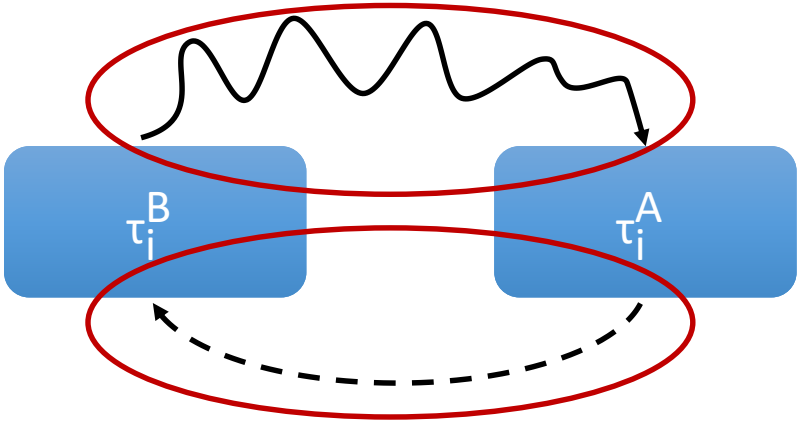
Super-Node Creation Rule



Super-Node Creation Rule

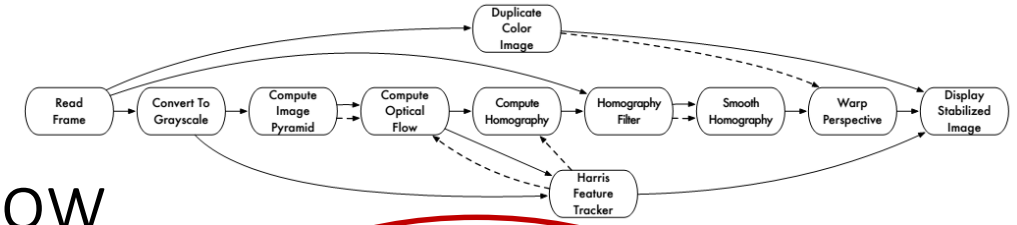


- The super-node creation rule may introduce additional pessimism by sacrificing potential parallelism.
- However, suppose the delay edge requires immediate history.
- If it causes only one cycle, and is the only delay edge in that cycle, which is a common case in many computer vision algorithms, then the sequential execution is actually enforced anyway.



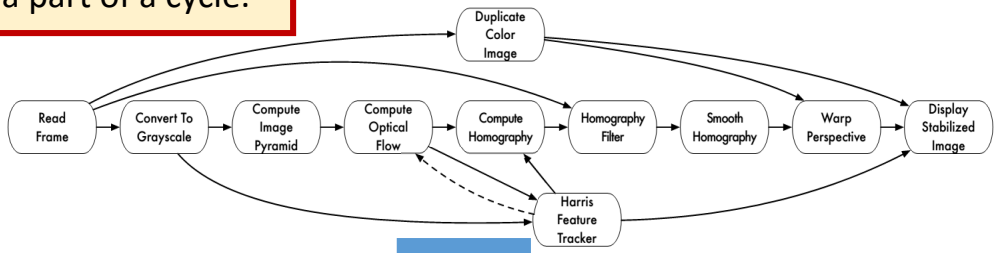
$B_{j-2} \rightarrow A_{j-2} \rightarrow B_{j-1} \rightarrow A_{j-1} \rightarrow B_j \rightarrow A_j, \ldots$

Transformation Flow



Delay-Edge Strengthening Rule

Eliminate delay edges that are not a part of a cycle.



Eliminate delay edges that are a part of a cycle.

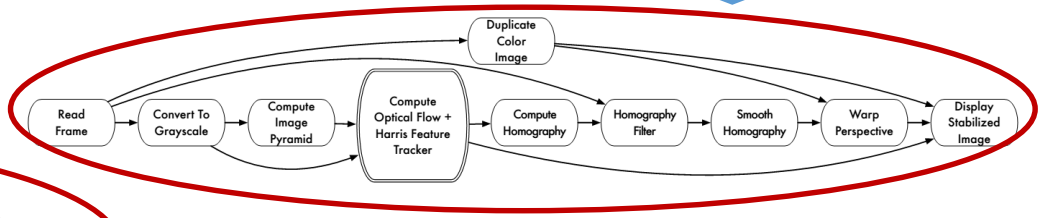
YES

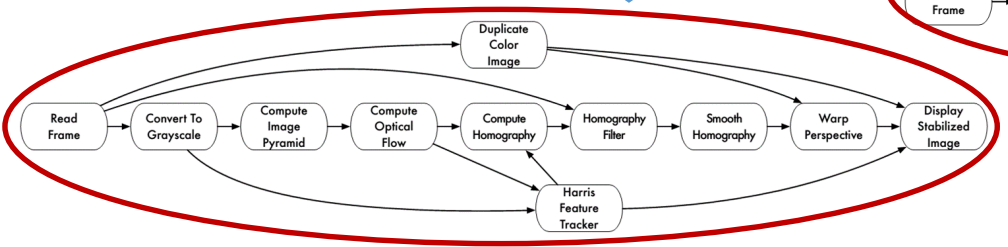
If the application code is adjustable

NO

Delay-Edge Dropping Rule

Super-Node Creation Rule

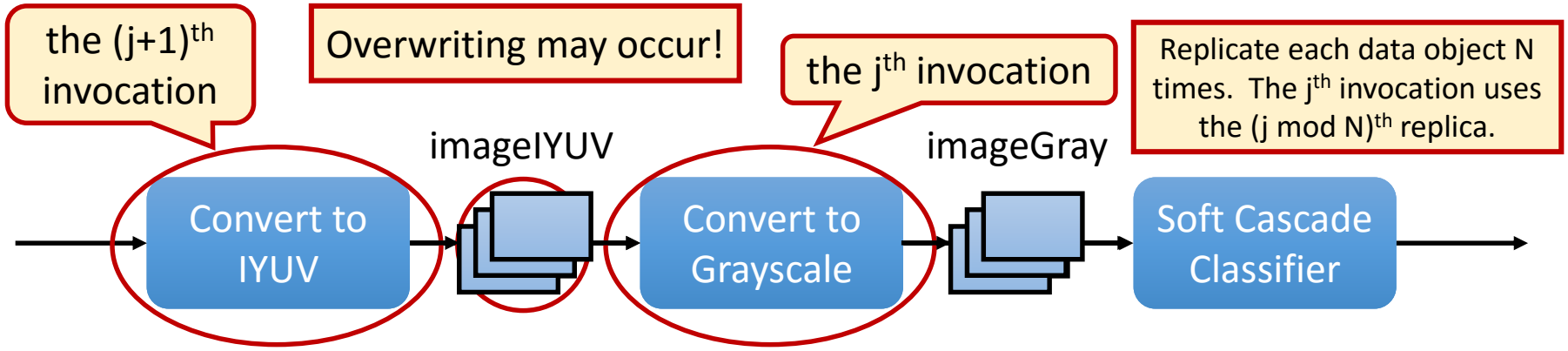




DAGs!

Data Overwriting

- The transformation techniques above are only for scheduling and for deriving end-to-end latency bounds.
- Recall that OpenVX specifies dependencies by bounding data objects.
- Existing implementation: a graph is required to execute end-to-end before it may be re-executed.
- Our extension: a graph may execute in a pipelined fashion.



Our solution: replicate the data objects.

Data Replica Bounds

- A safe bound on the number of replicas, N, can be derived.
- Intuition:
  - There is a minimum separation between DAG invocations.
  - End-to-end latency of the DAG is bounded.
  - At a time, the number of active invocations is bounded.

- Instead of being associated with a data object, each delay edge is associated with a ring buffer that stores the “history” that may be needed in the future.
- Similar techniques can be applied to delay edges to derive a bound on the size of those ring buffers.

Conclusion

- OpenVX is a recently ratified standard that has a graph-based processing model.
- Our recent work[1] extended an existing NVIDIA OpenVX implementation by adding real-time support.
- This paper provides more detailed analysis for that implementation.
  - Transform OpenVX graphs to DAGs, so that end-to-end latencies can be guaranteed by applying prior work on DAGs.
  - Derive upper bounds on data object replicas and buffers, i.e., ensure that our techniques use finite memory.

[1] G. Elliott, K. Yang, and J. Anderson, "*Supporting real-time computer vision workloads using OpenVX on multicore+GPU platforms*," RTSS 2015, to appear.

Thank you!

Questions?