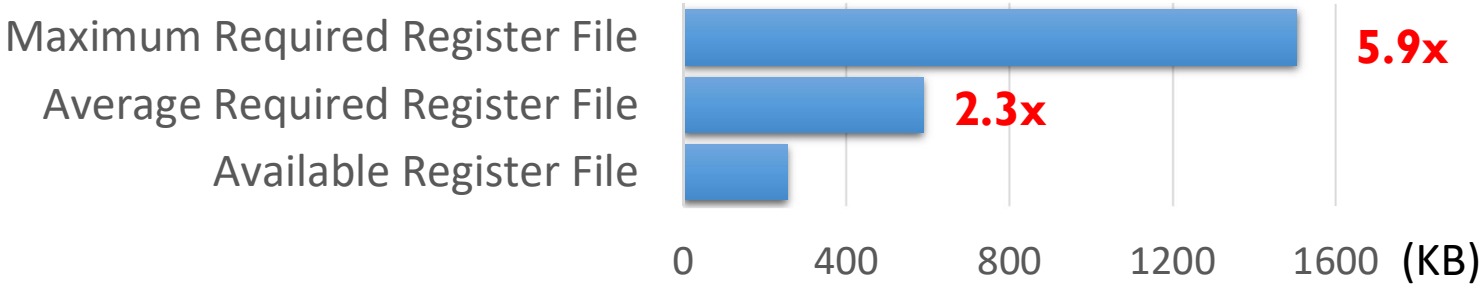
LTRF: Enabling High-Capacity Register Files for GPUs via Hardware/Software Cooperative Register Prefetching

Mohammad Sadrosadati
Seyed Borna Ehsani
Mario Drumond
Rachata Ausavarungnirun

Amirhossein Mirhosseini
Hamid Sarbazi-Azad
Babak Falsafi
Onur Mutlu

Register file size limits GPU scalability

- Register file (RF) already accounts for 60% of on-chip storage
- But, there is still demand for more registers to achieve maximum performance and concurrency



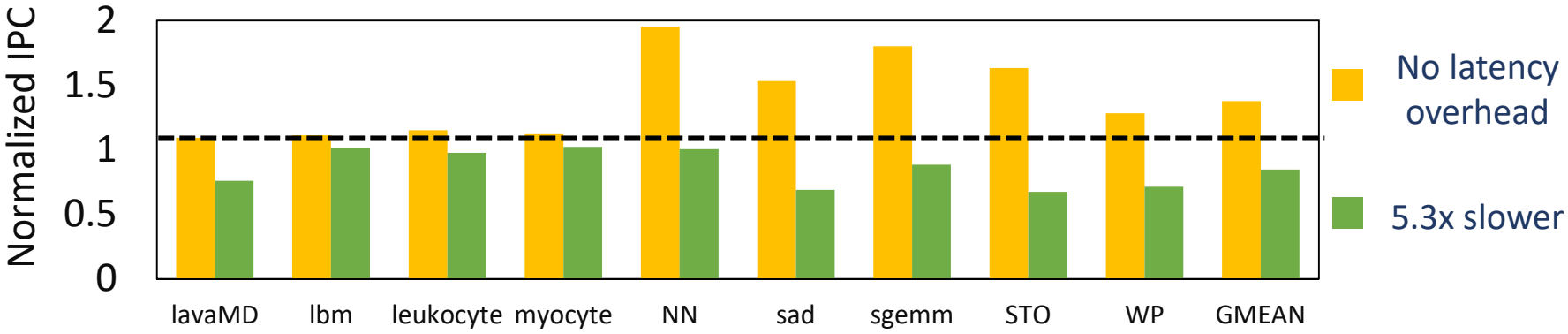
- Future slow memory accesses call for more threads
  - Multi-socket, multi-GPU, RDMA, NVM, etc.
- Compiler optimizations call for more registers per thread
  - Loop unrolling, thread coarsening, etc.

Need mechanisms to expand RF capacity (without large area/power overheads)

How to make register files larger?

- Emerging technologies [Jing'13][Mao'14][Wang'15][Abdel-Majid'17]
- Register file compression [Lee'15]
- Register file virtualization [Jion'15][Vijaykumar'16][Kloosterman'17]

- Common challenge: Latency overhead
  - **Example:** 8x larger register file with NTV TFET



Goal: Tolerate register file latencies

Contributions

- **Latency Tolerant Register File (LTRF)**
  - "2-level" main register file + register cache
  - Performs prefetch ops while executing other warps
  - Paves the way for several power/area optimizations
- **Compiler-driven Register Prefetching**
  - Break control flow graph into "prefetch subgraphs"
  - Prefetch registers at the beginning of each subgraph
  - Interval analysis to identify prefetch subgraphs

LTRF tolerates up to 6x slower register files

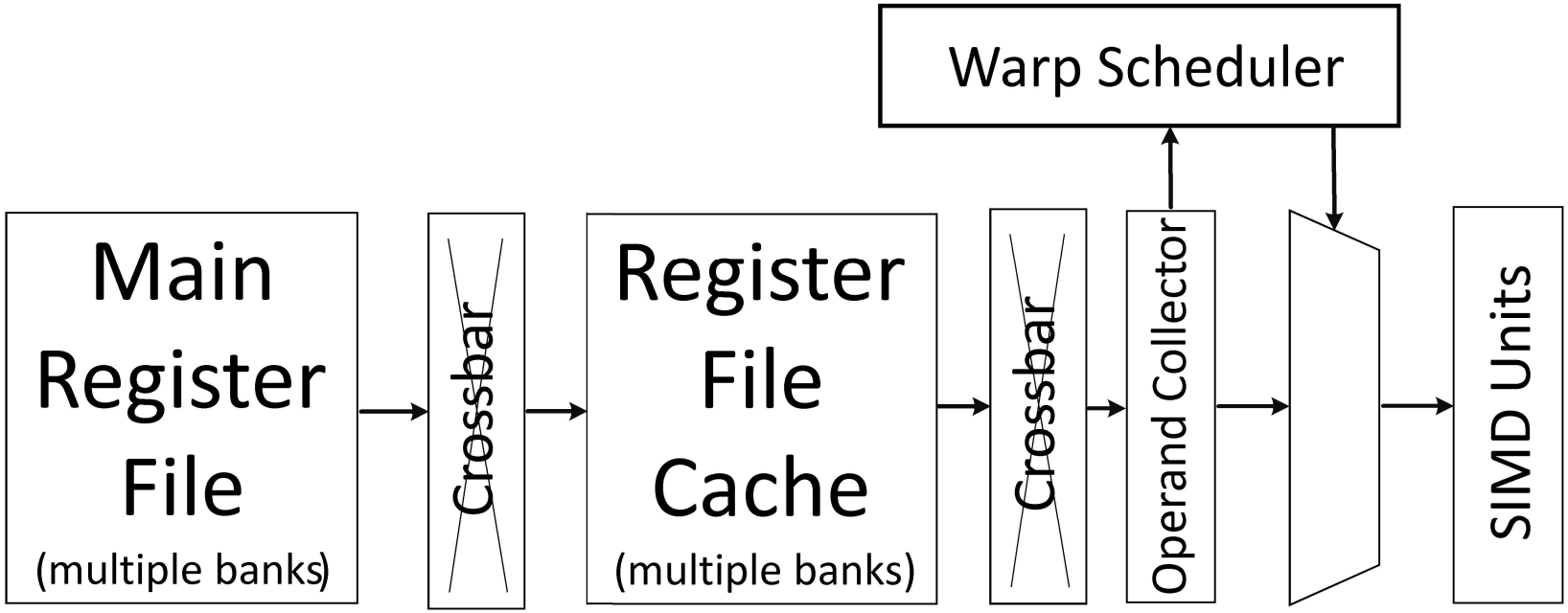
Example LTRF use case:
8× larger RF → 34.8% higher performance

Outline

- Background and challenges
- The case for compiler-driven register prefetching in GPUs
- LTRF architecture and compiler support
- Evaluation methodology
- Results

Register file caching [Gebhart' ISCA11]

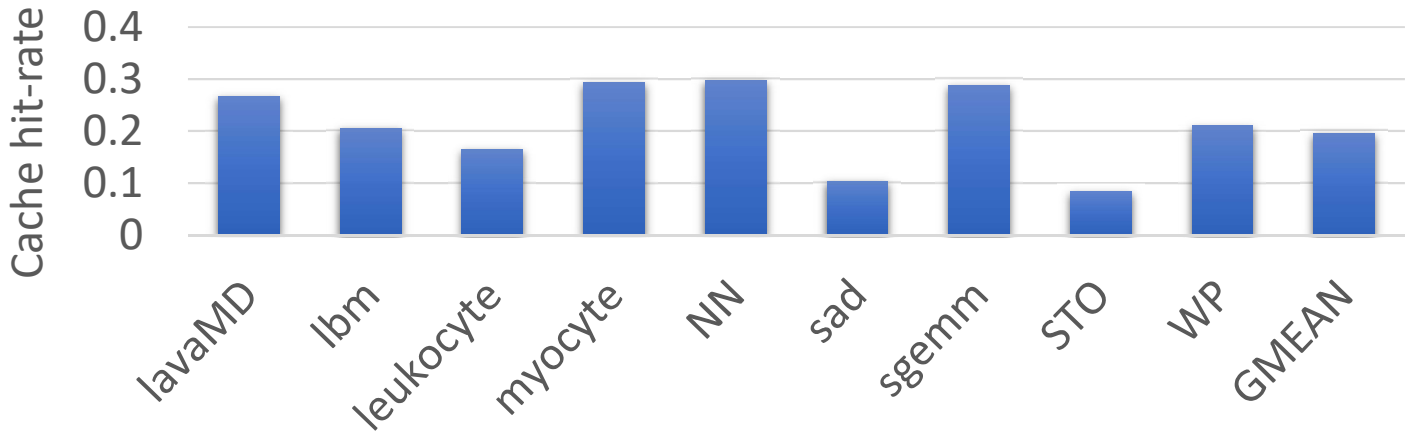
- Promising approach for latency tolerant register files



Unfortunately, classic demand fetch and replace yields low hit rate in register caches

Demand fetch/replace register caching

- 8-30% hit rate



- Why?
  - No spatial locality for registers
  - Values might be renamed to different registers
    - Scrambles temporal locality
  - Lots of threads → cache thrashing

Our solution: Precise register prefetching

Compiler-driven register prefetching

- Possible to have near-perfect register prefetchers
  - Register working sets known at compile time
    - No indirection or address translation
  - Prefetch latency may overlap with other warps

Compiler-driven register prefetching

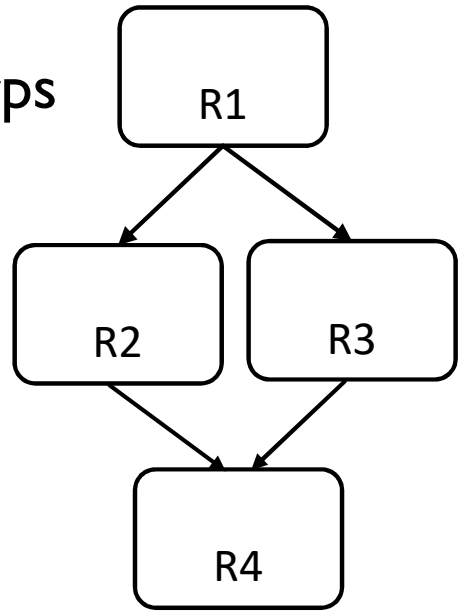
- Possible to have near-perfect register prefetchers
  - Register working sets known at compile time
    - No indirection or address translation
  - Prefetch latency may overlap with other warps

Key idea: **"prefetch subgraphs"**

- Prefetch register working sets into the cache at the beginning of each subgraph
- All register accesses in the prefetch subgraph hit in the register cache

Compiler-driven register prefetching

- Possible to have near-perfect register prefetchers
  - Register working sets known at compile time
    - No indirection or address translation
  - Prefetch latency may overlap with other warps

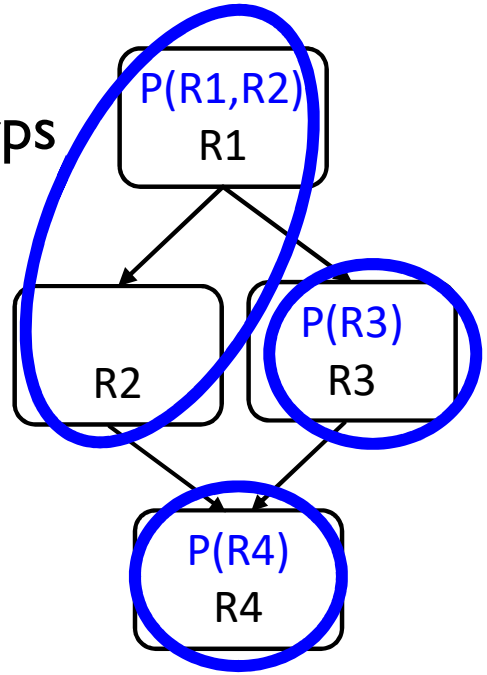


Key idea: **"prefetch subgraphs"**

- Prefetch register working sets into the cache at the beginning of each subgraph
- All register accesses in the prefetch subgraph hit in the register cache

Compiler-driven register prefetching

- Possible to have near-perfect register prefetchers
  - Register working sets known at compile time
    - No indirection or address translation
  - Prefetch latency may overlap with other warps



Key idea: **"prefetch subgraphs"**

- Prefetch register working sets into the cache at the beginning of each subgraph
- All register accesses in the prefetch subgraph hit in the register cache

What are best prefetch subgraphs?

Optimal prefetch subgraphs

Objectives

- Prefetch operations dominate register uses
- Minimum number of prefetch operations
- Fit entire loops
  - Maximize dynamic insts

Implications

- Single-entry subgraphs
- Largest possible subgraphs
- Capture backward branches

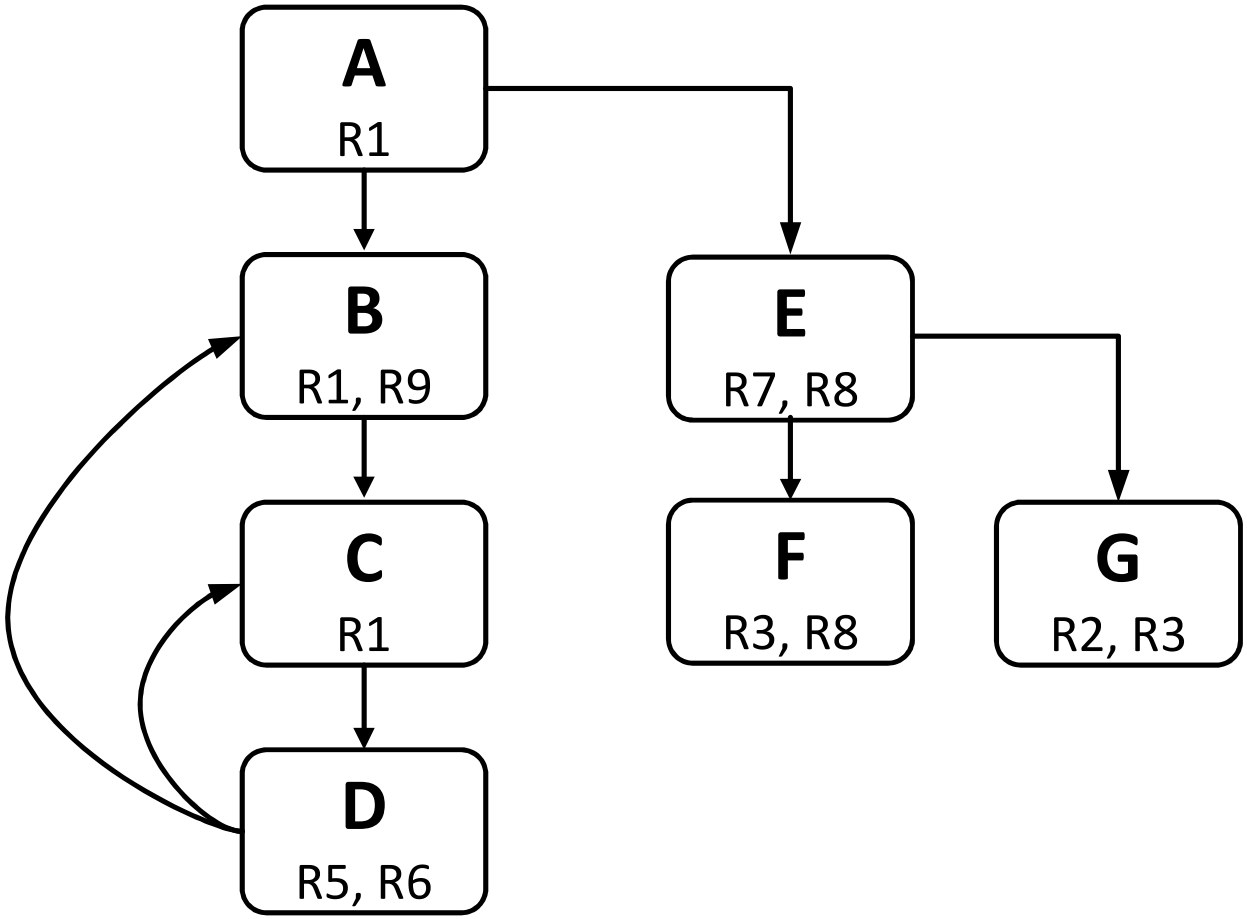
Register intervals

- **Intervals:** disjoint single-entry subgraphs of CFG
- **Register intervals** access at most *k* registers
  - Reserve *k* register slots for each warp to prevent eviction

1. Entry block is the header of the first interval
2. Greedily add child basic blocks iff:
   - Incoming edges only from within interval, AND
   - | registers accessed in interval | ≤ *k*
3. Remaining children become new headers
4. Repeat until graph is irreducible

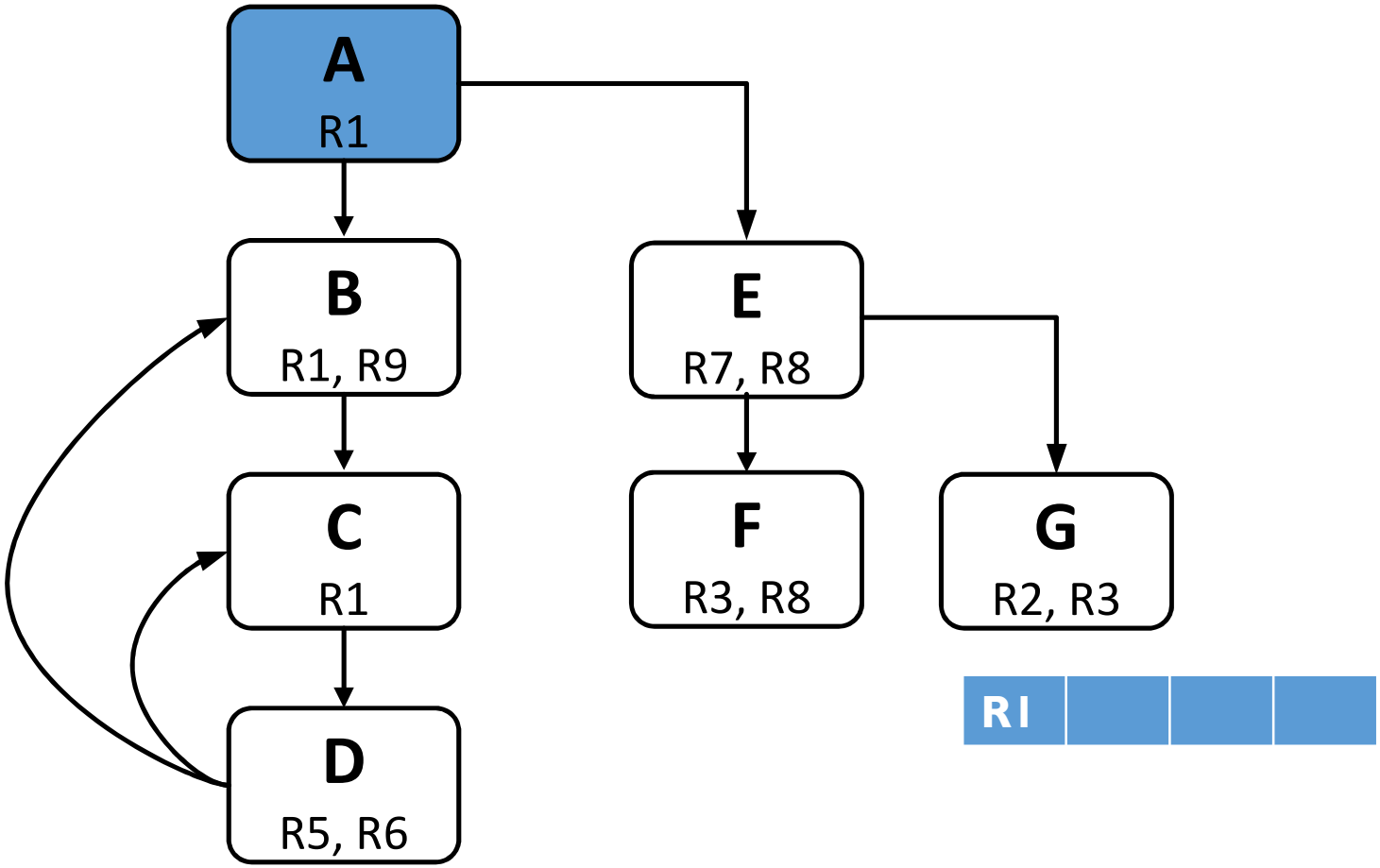
Prefetch register working sets at the beginning of register intervals

Register intervals in action



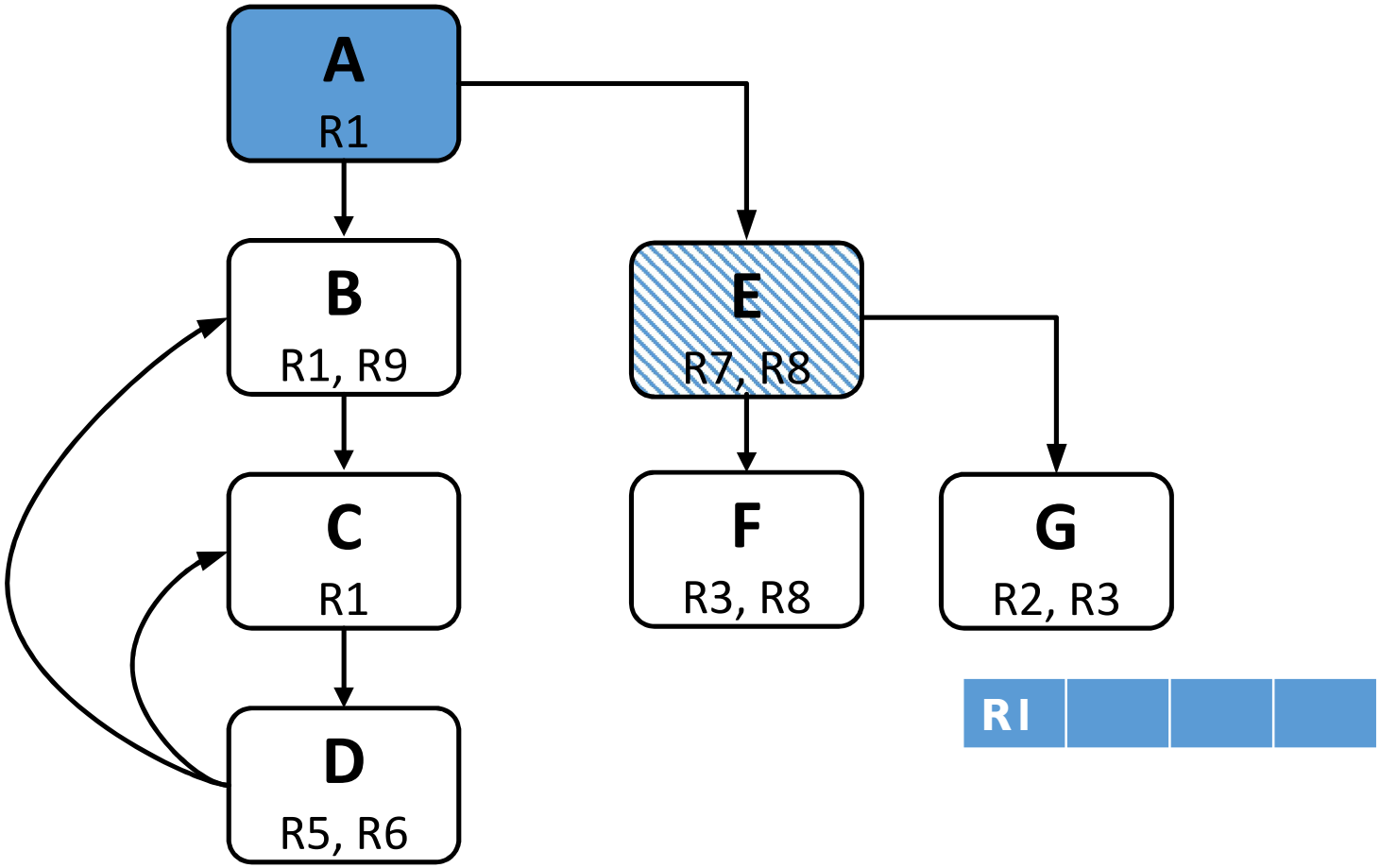
Register intervals in action

- A is the first interval header



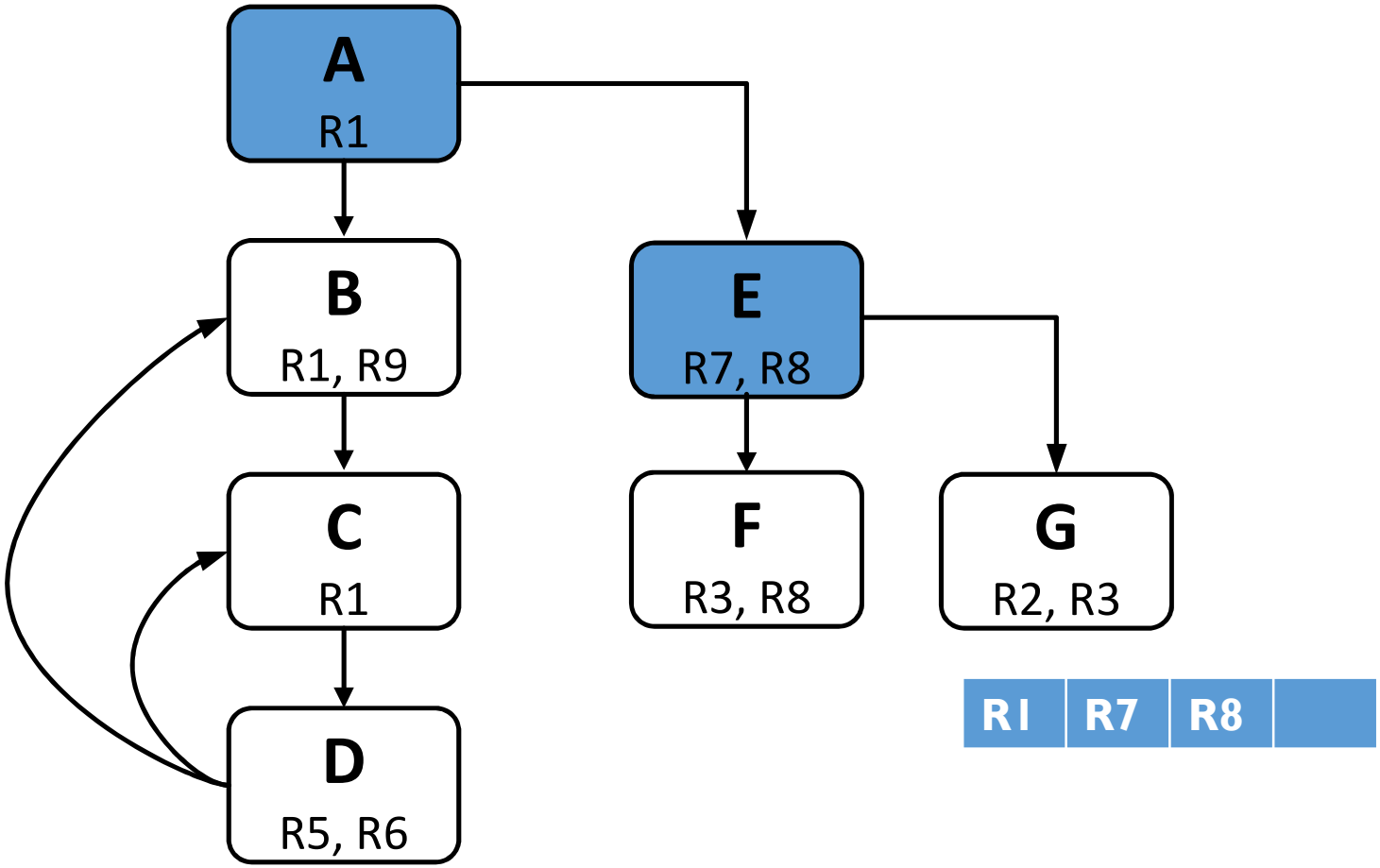
Register intervals in action

- E is the only candidate to merge



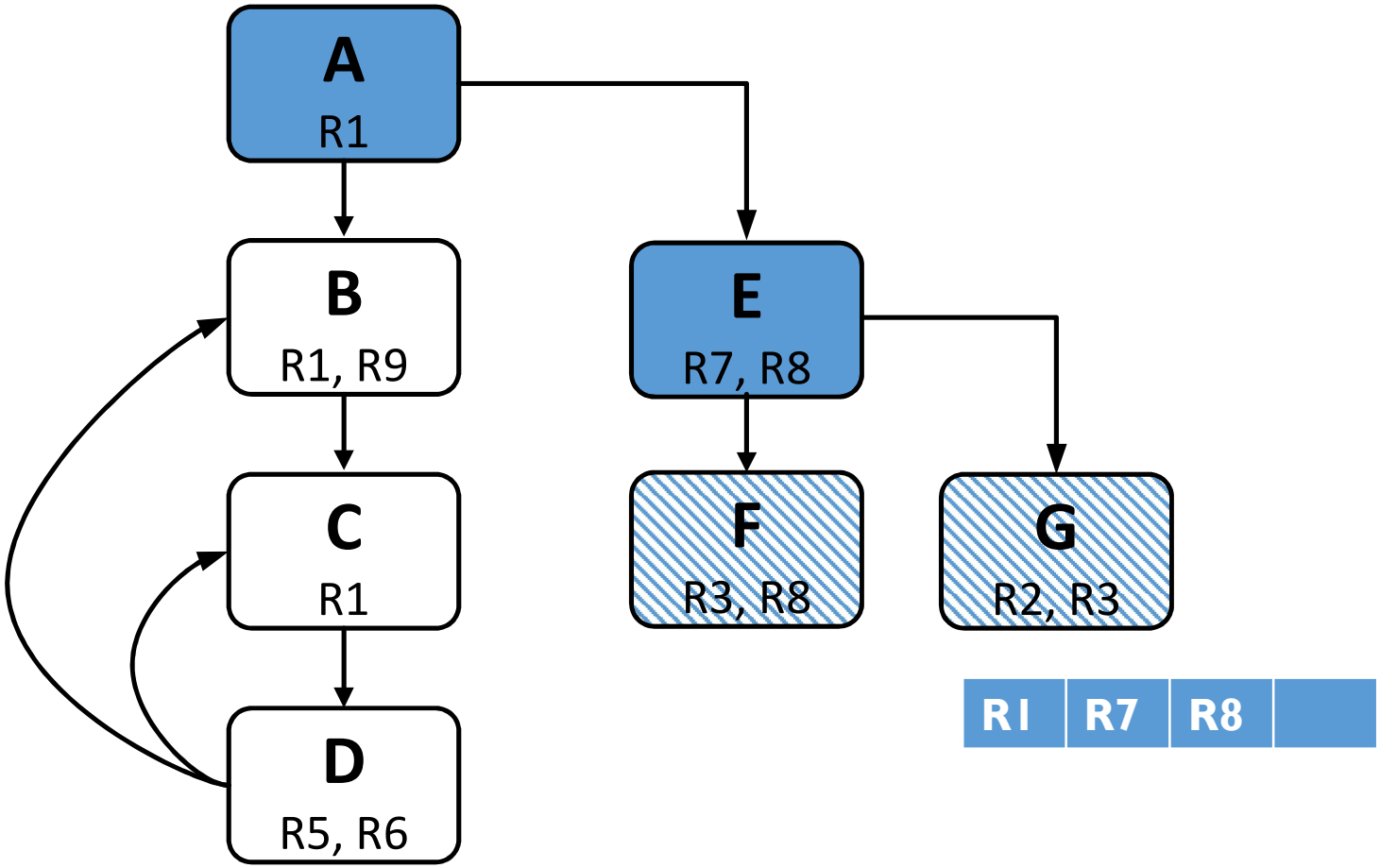
Register intervals in action

- E merges



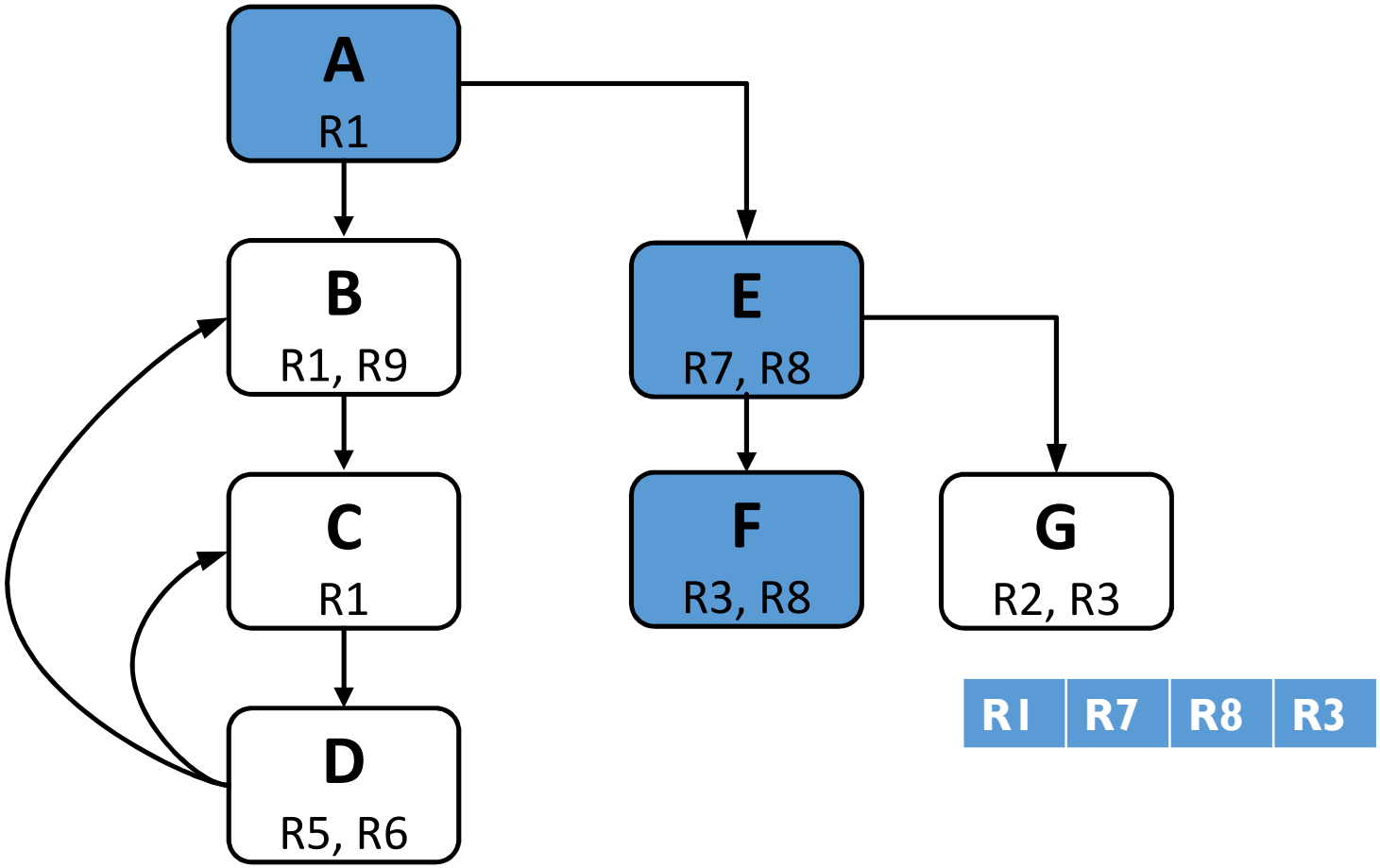
Register intervals in action

- F and G are potential candidates to merge



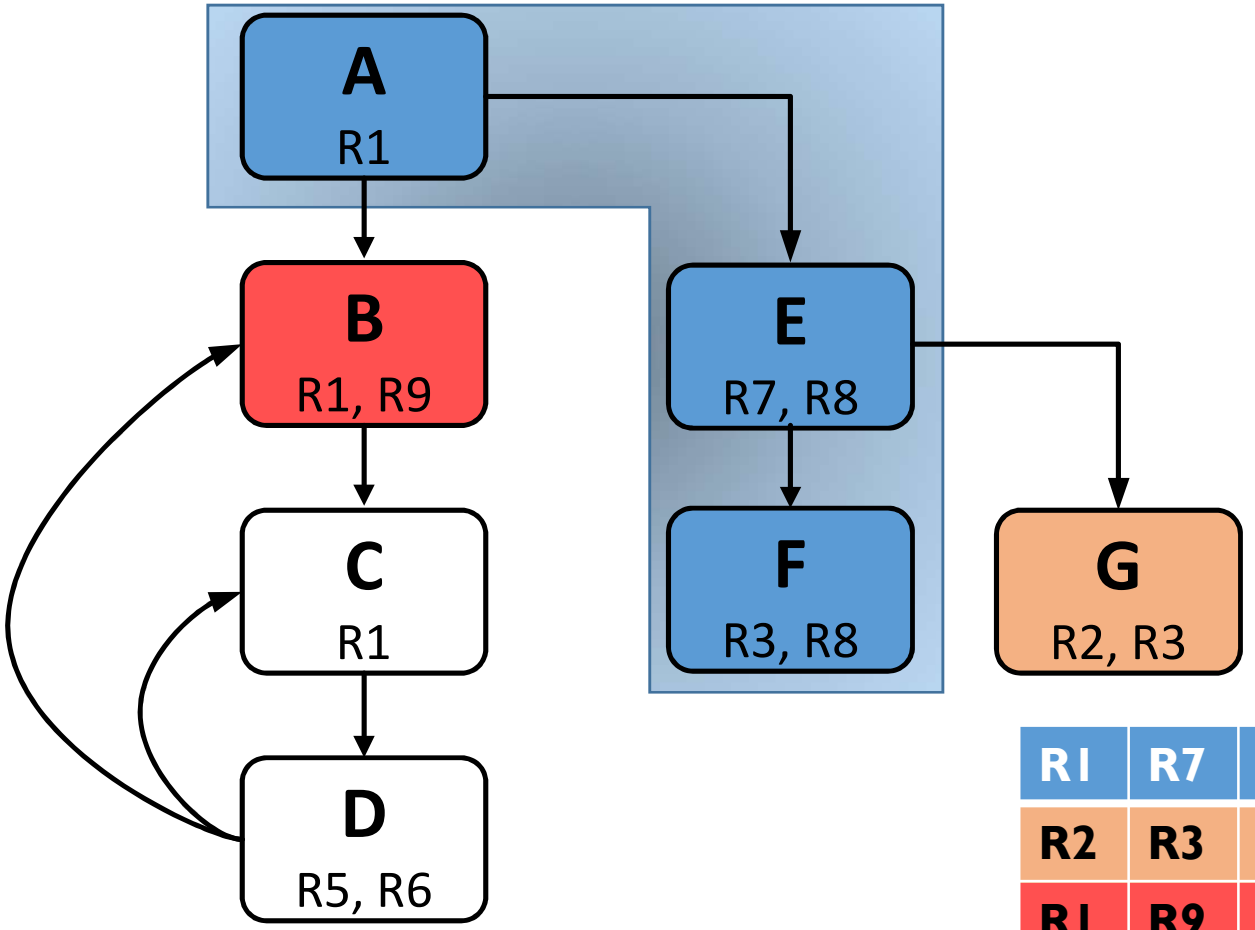
Register intervals in action

- F merges but G can't (register cache is full)



Register intervals in action

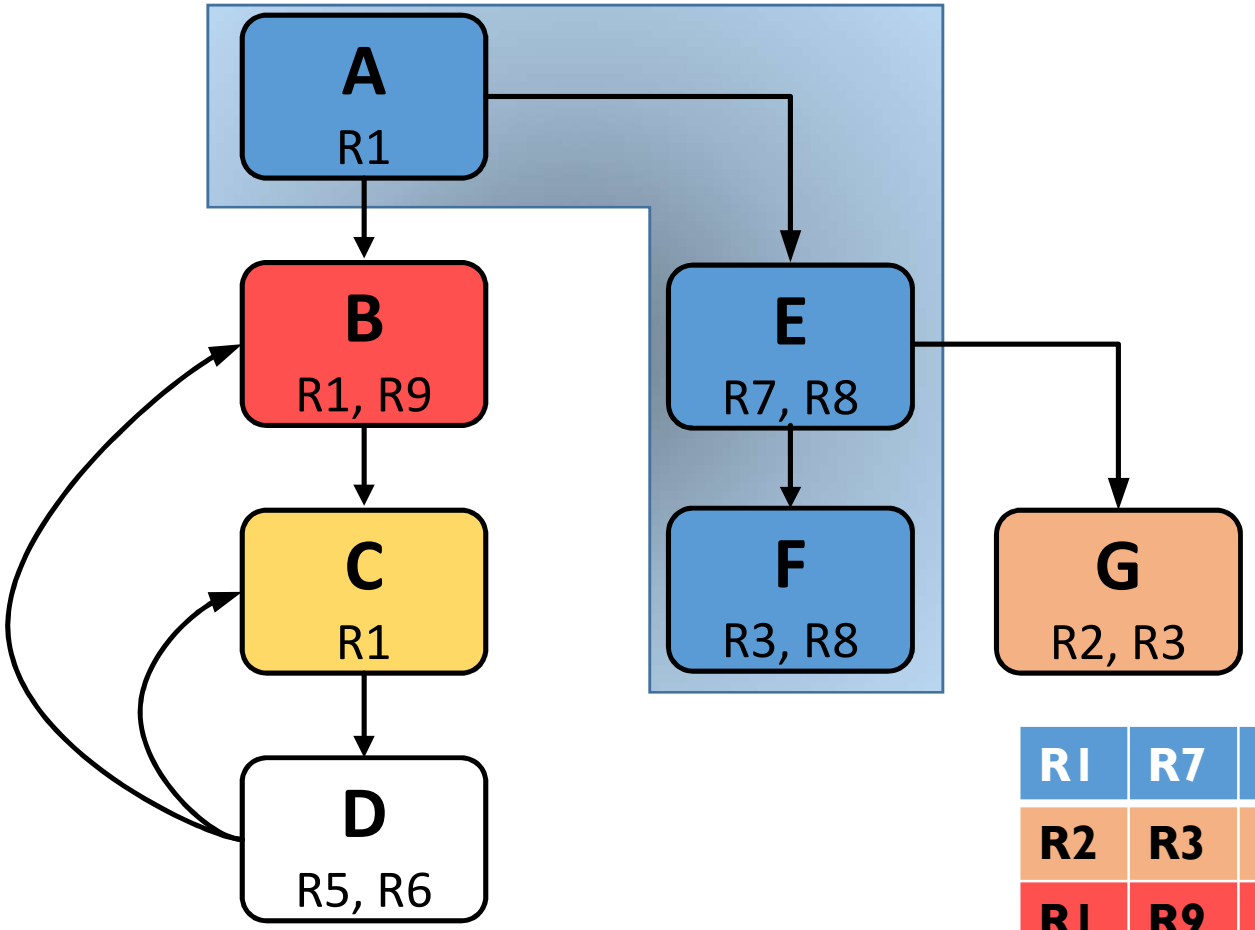
- Done with first interval --- B and G become headers



R1	R7	R8	R3
R2	R3		
R1	R9		

Register intervals in action

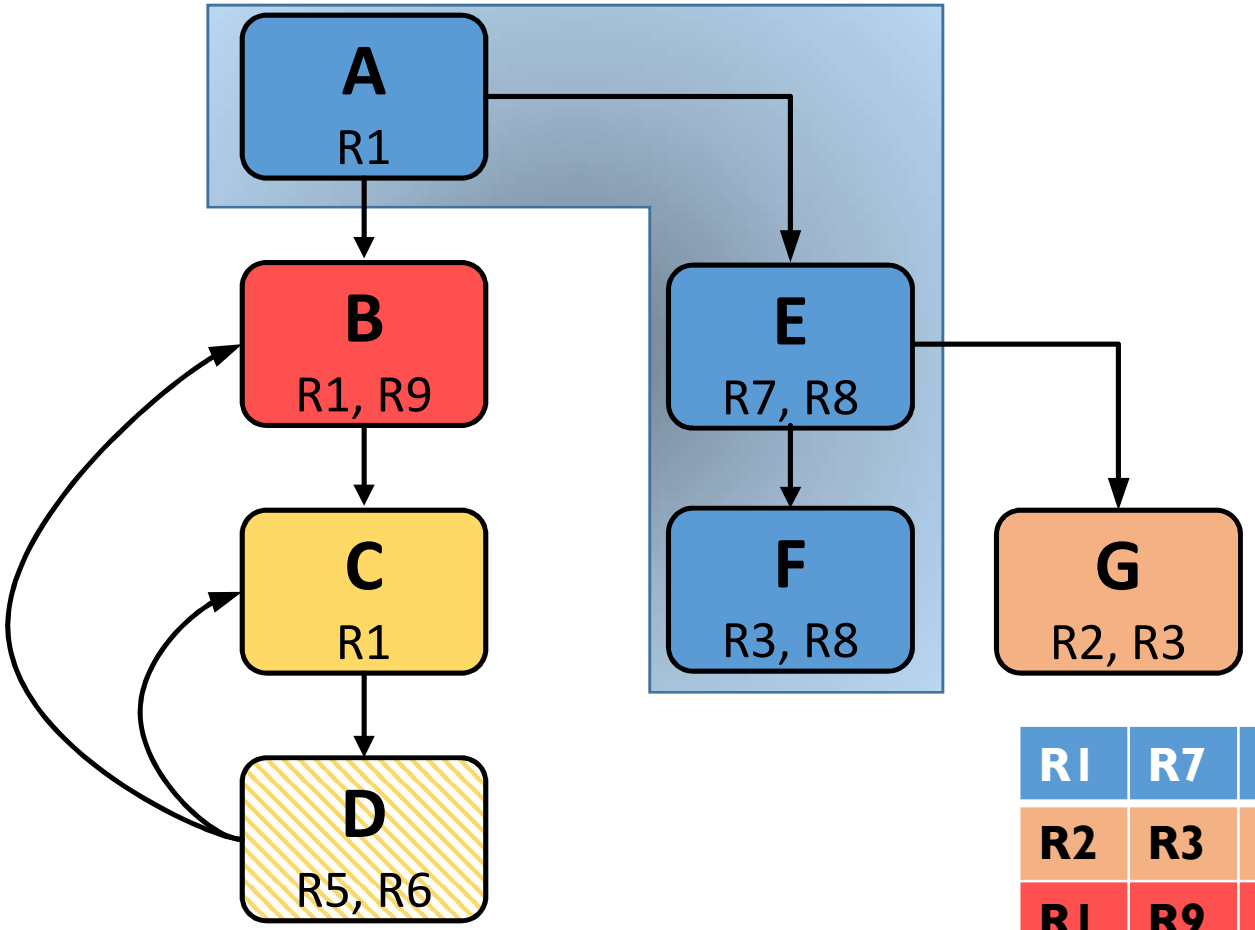
- No candidate to merge into B --- C becomes header



R1	R7	R8	R3
R2	R3		
R1	R9		
R1			

Register intervals in action

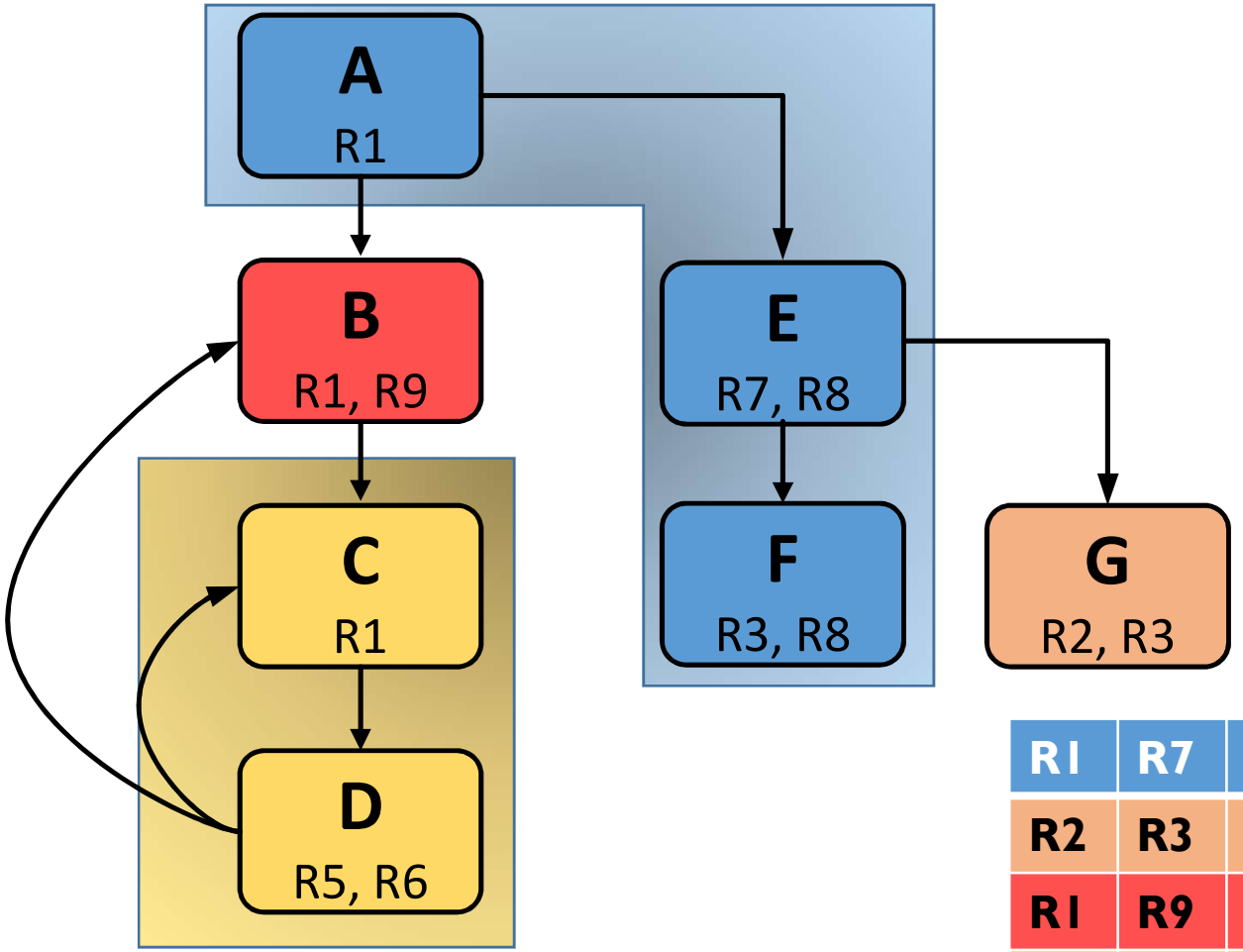
- D becomes candidate to merge into C



R1	R7	R8	R3
R2	R3		
R1	R9		
R1			

Register intervals in action

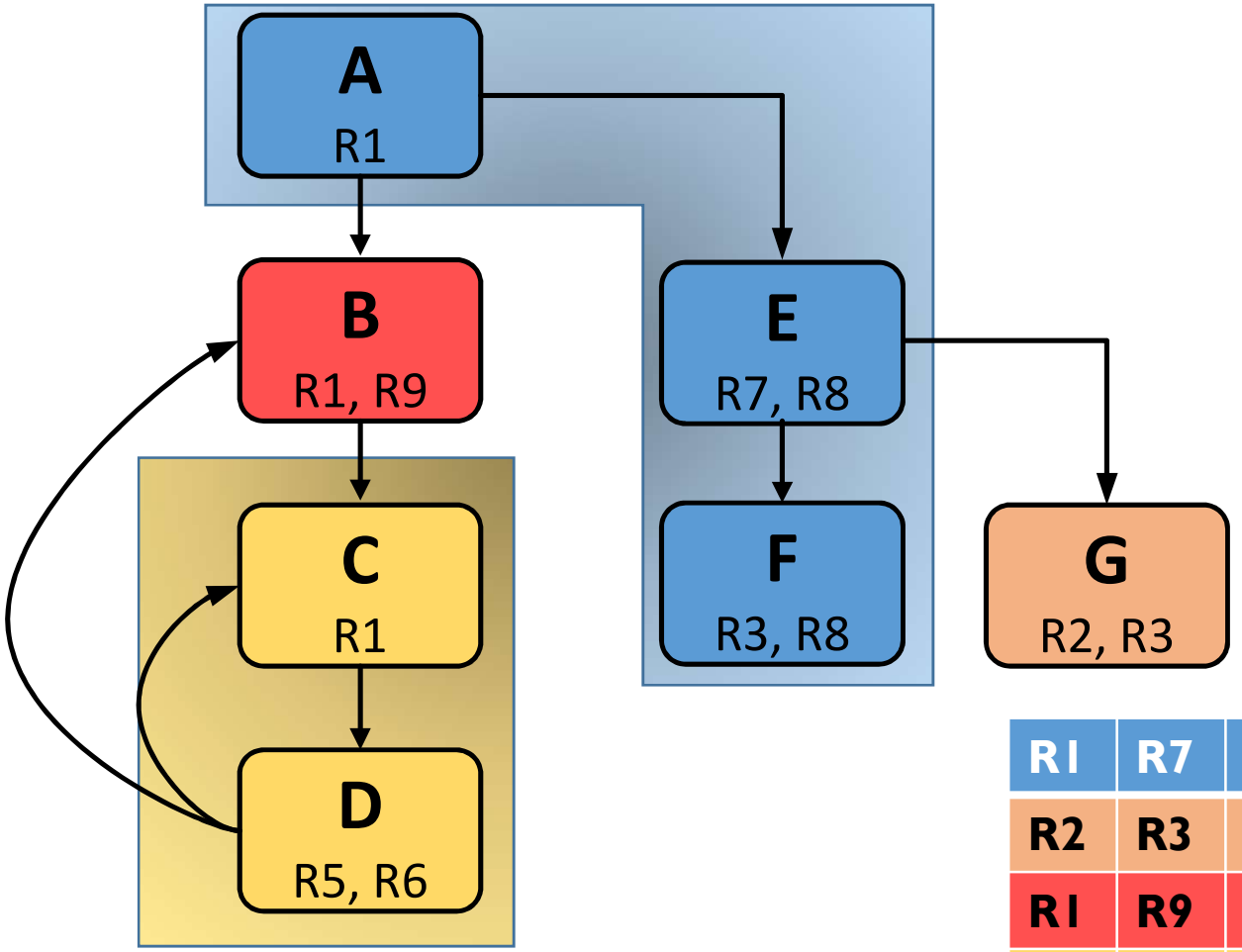
- D merges into C --- done with the first pass



R1	R7	R8	R3
R2	R3		
R1	R9		
R1	R5	R6	

Register intervals in action

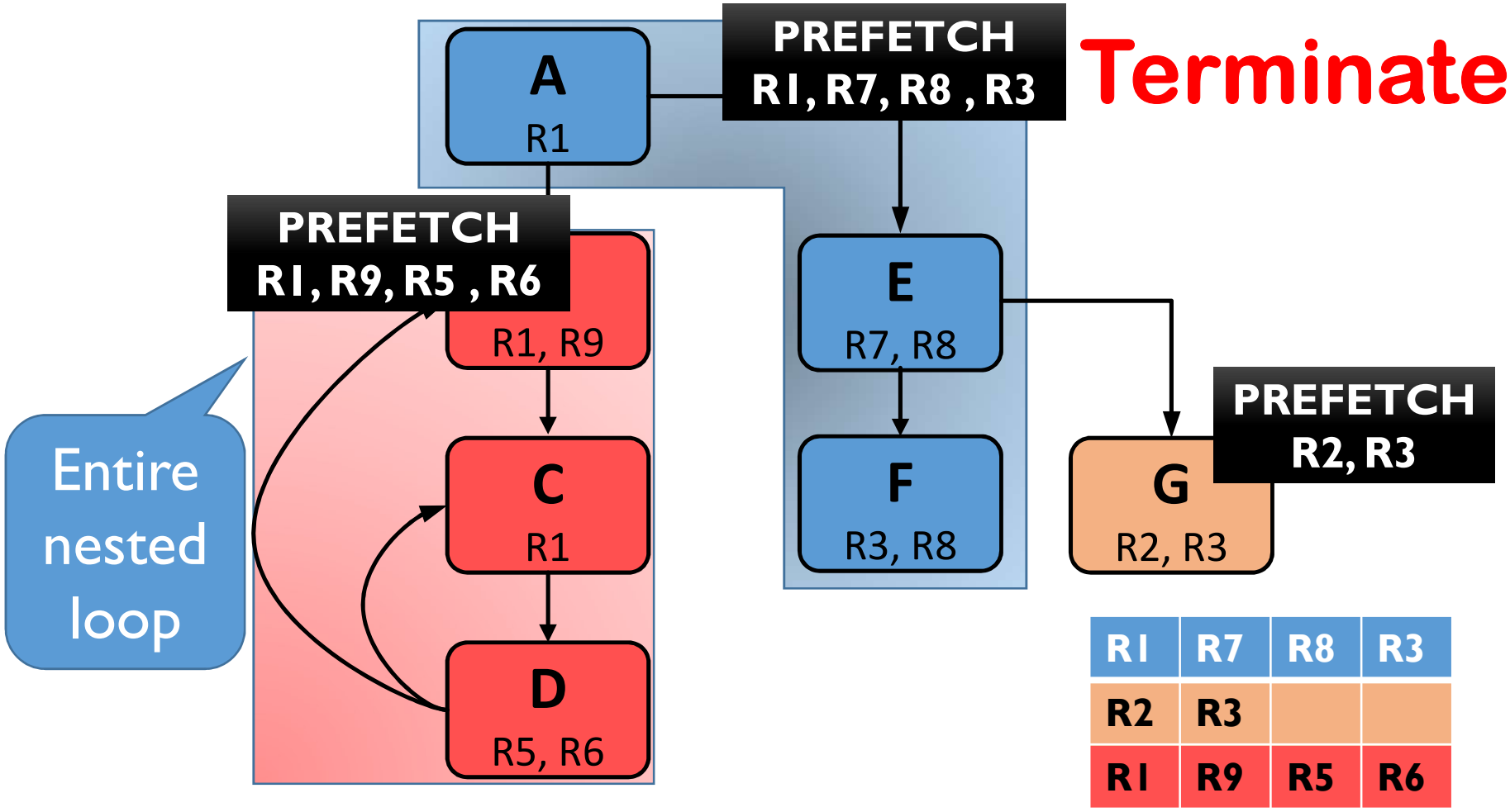
- **Second pass:** Yellow is able to merge into Red



R1	R7	R8	R3
R2	R3		
R1	R9		
R1	R5	R6	

Register intervals in action

- Done with second pass --- graph no further reducible



Register interval highlights

- Single-entry prefetch subgraphs
  - Prefetch operations dominate register uses
- Maximal-length subgraphs
  - Minimize prefetch overheads
- Minimal termination constraints
  - Encapsulate entire loops
  - Maximize dynamic instructions per interval
- Multi-pass construction algorithm based on classic interval analysis

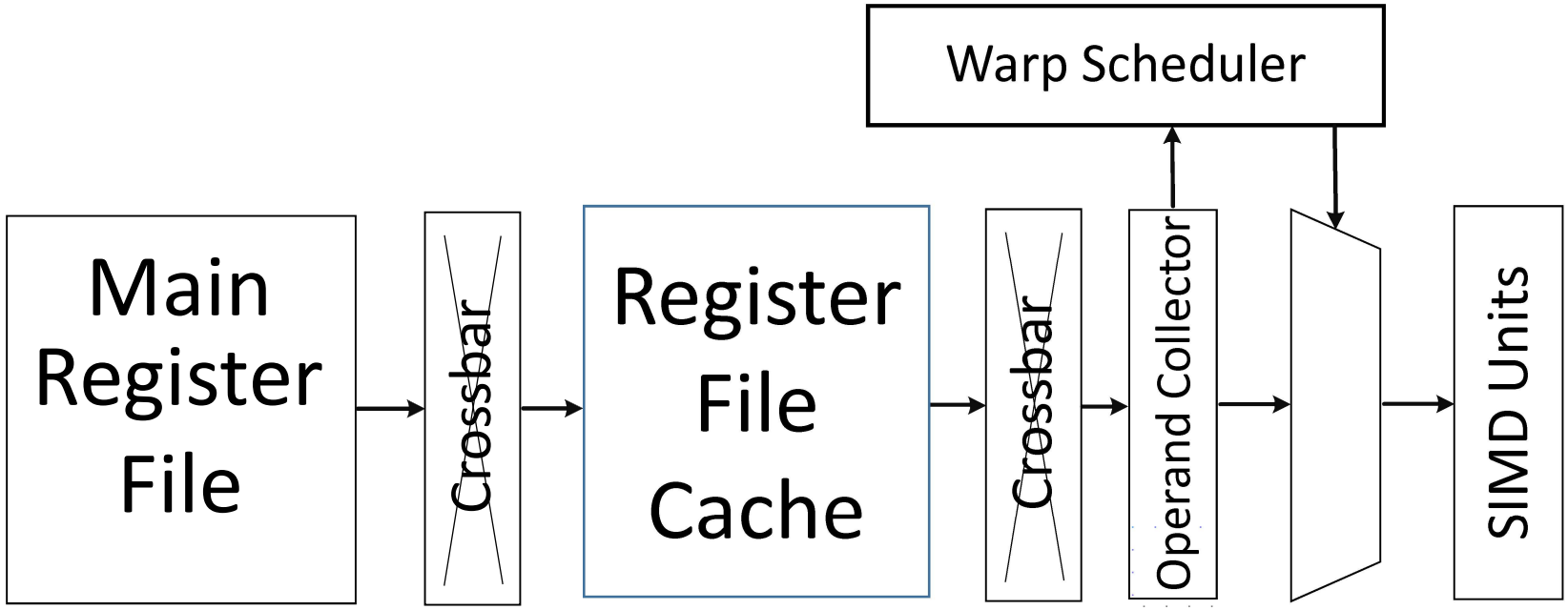
Need hardware mechanisms to provide fixed-size cache partitions for register intervals

Outline

- Background and challenges
- The case for compiler-driven register prefetching in GPUs
- **LTRF architecture and compiler support**
- Evaluation methodology
- Results

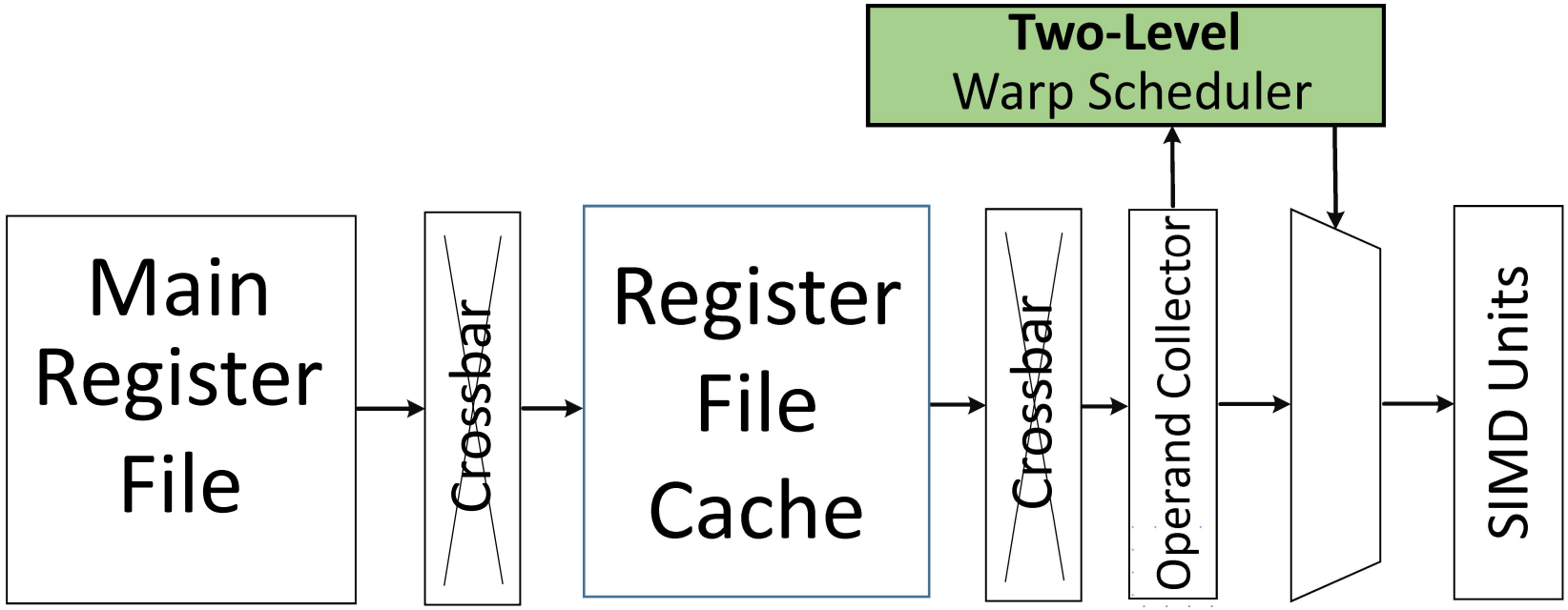
Latency Tolerant Register File (LTRF)

- "2-level" register file



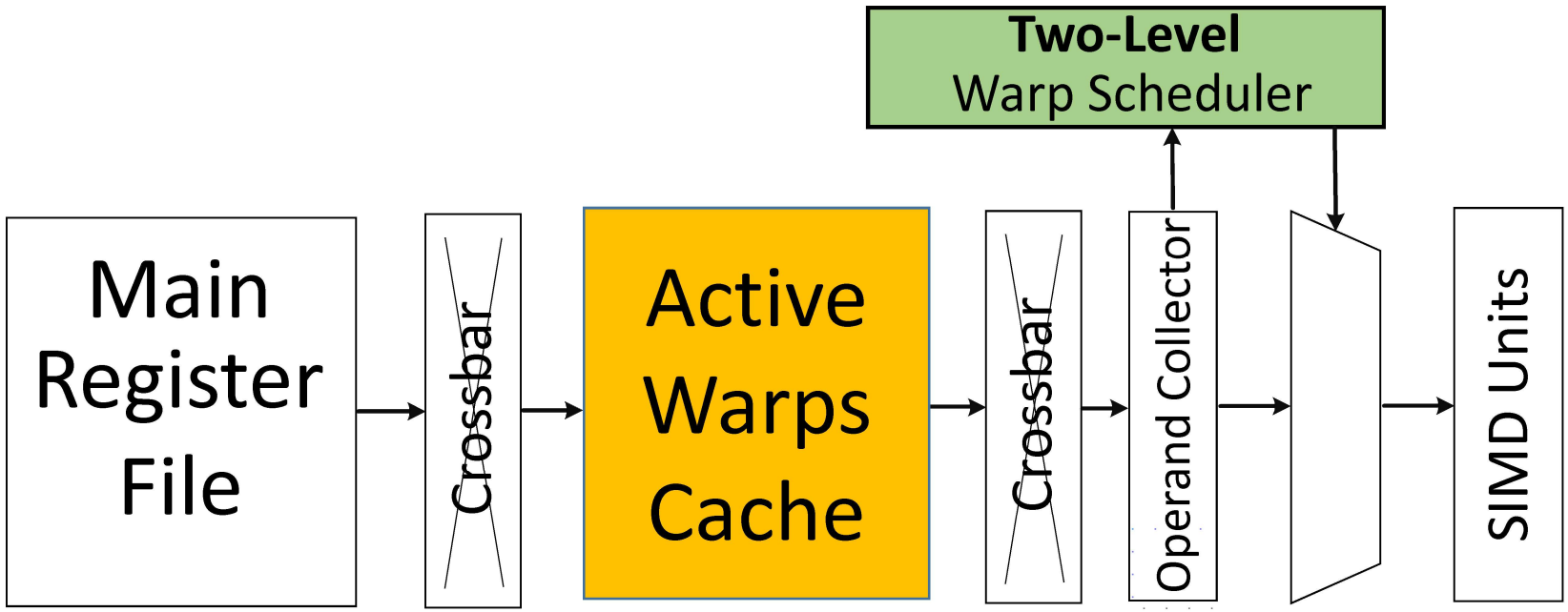
Latency Tolerant Register File (LTRF)

- "2-level" register file + warp scheduler



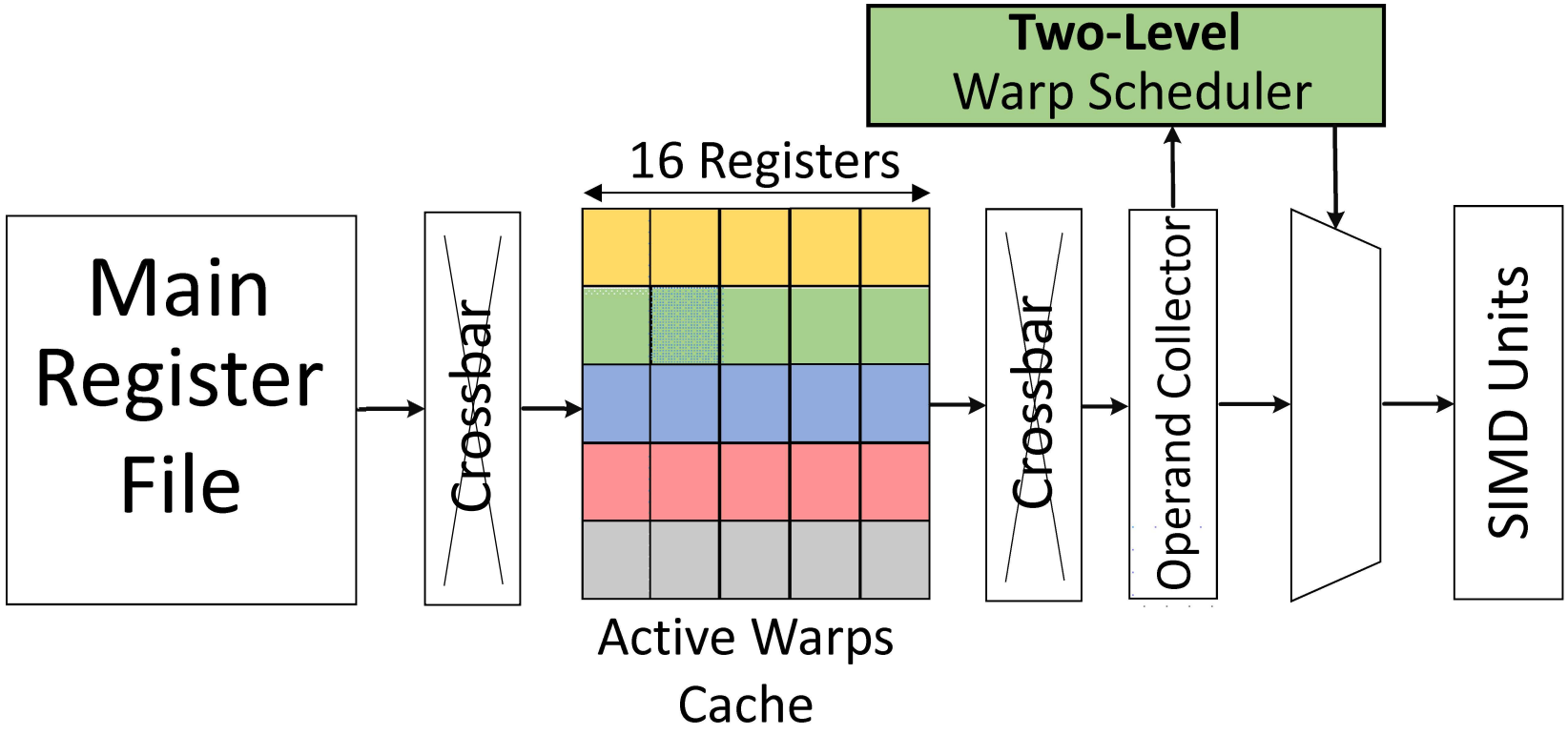
Latency Tolerant Register File (LTRF)

- "2-level" register file + warp scheduler
- Cache registers only for the **active** warps



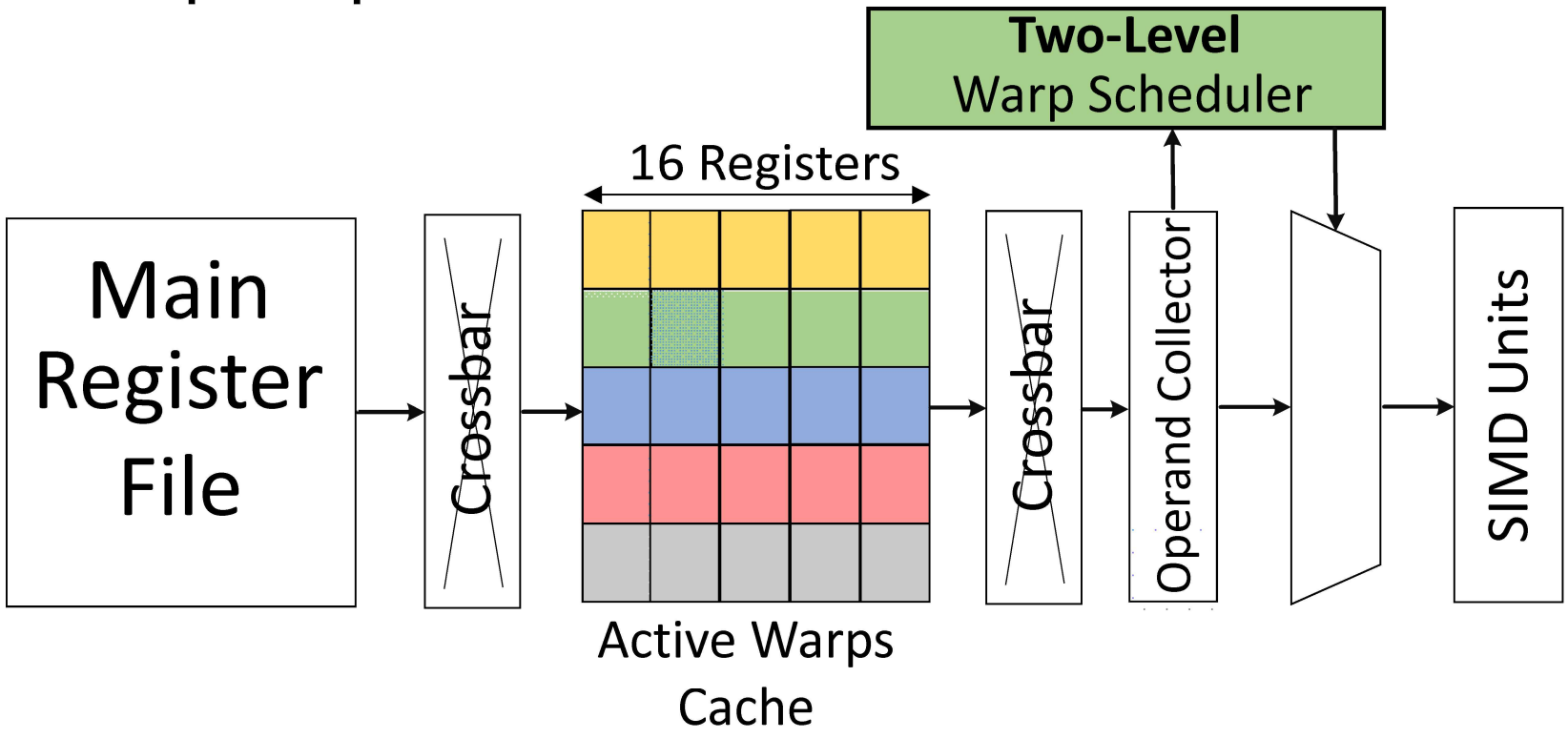
Latency Tolerant Register File (LTRF)

- "2-level" register file + warp scheduler
- Cache registers only for the **active** warps
- **Dedicated** register cache space for each warp



Latency Tolerant Register File (LTRF)

- "2-level" register file + warp scheduler
- Cache registers only for the **active** warps
- **Dedicated** register cache space for each warp
- Swap warps on PREFETCH



Outline

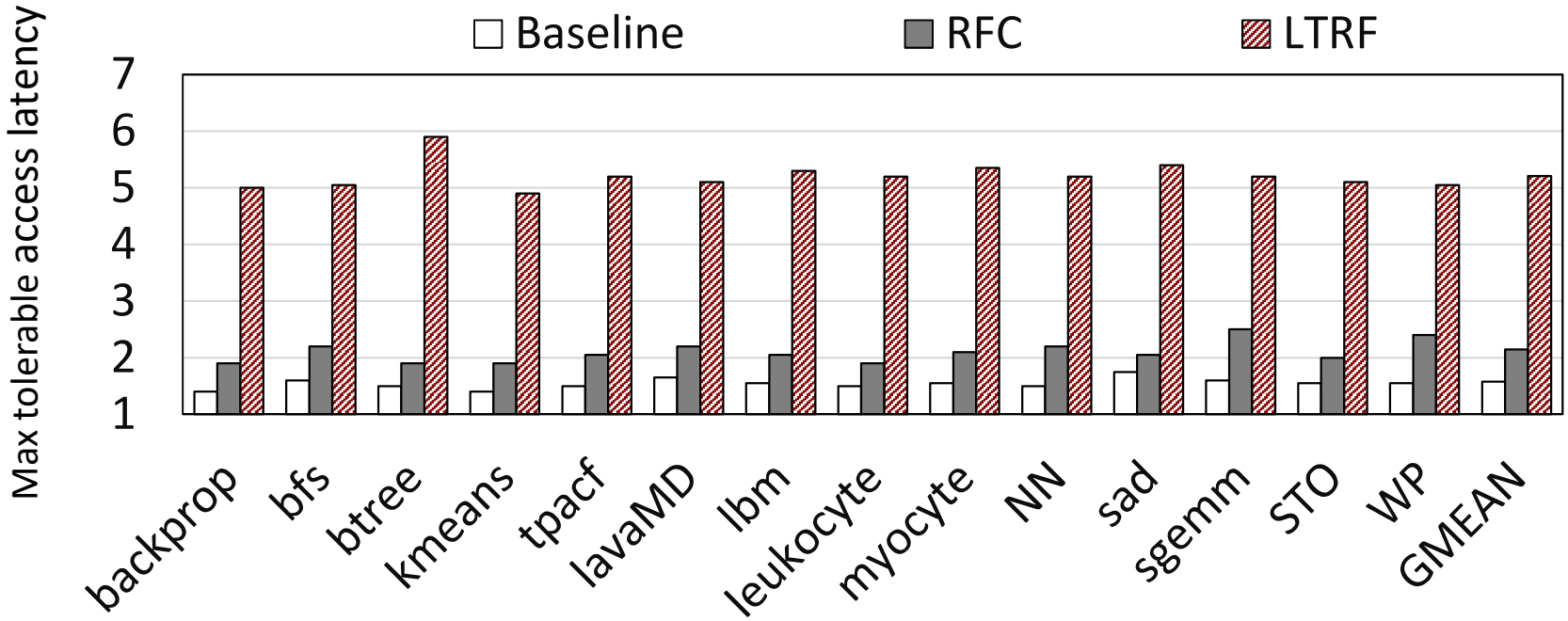
- Background and challenges
- The case for compiler-driven register prefetching in GPUs
- LTRF architecture and compiler support
- **Evaluation methodology**
- Results

Evaluation methodology

- **Simulator:** GPGPU-Sim modeling NVIDIA Maxwell
- **Workloads:** CUDA-SDK, Rodinia, and Parboil suites
- **Comparison points:**
  - Baseline: No register caching
  - RFC: Demand fetch register file caching (Gebhart' ISCA 2011)
  - **LTRF**
  - Ideal: Increased capacity with no latency overhead

Latency tolerance

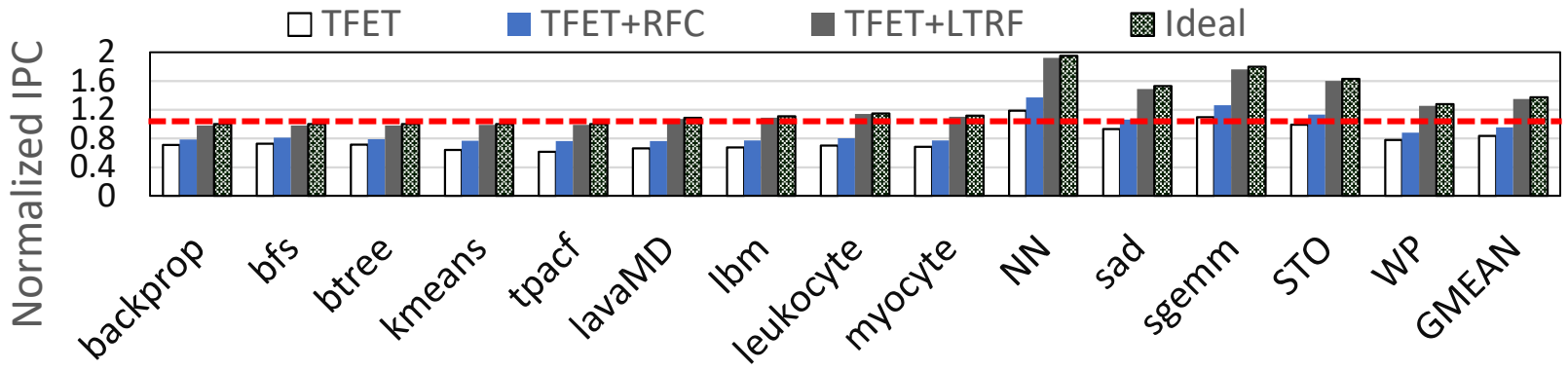
- Max tolerable RF latency with IPC slowdown <= 5%



LTRF tolerates the latencies of up to 6x slower register files

Performance improvement

- **Example use case:** Increase register file capacity from 256KB to 2MB using NTV TFET
  - Same power/area as the baseline register file (256 KB)
  - 2nd-level RF accesses 5.3X slower than baseline



LTRF+TFET improves performance by 34.8% within 2.3% of an ideal large register file

Also in the paper…

- **LTRF+:** minimize register movement between the register file and register cache using liveness analysis

- Register interval compared to other subraphs
  - Strands, superblocks, etc.

- Detailed analysis of hardware overheads
  - 16% more area
  - 21% less power

- Various LTRF use cases with different register file technologies and optimizations

Conclusion

- Register files are the main GPU scalability bottlenecks
  - They already consume 60% of total on-chip memory
  - Need more registers for highest performance
- Standalone register caching solutions yield low hit rates

- **Latency Tolerant Register File (LTRF)**
  - "2-level" main register file + register cache
  - Prefetch register working sets ahead of time
  - Performs prefetch ops while executing other warps
  - Interval analysis for near-optimal prefetching
  - Tolerates up to 6x slower main register files
  - Paves the way for several power/area optimizations

LTRF: Enabling High-Capacity Register Files for GPUs via Hardware/Software Cooperative Register Prefetching

Mohammad Sadrosadati
Seyed Borna Ehsani
Mario Drumond
Rachata Ausavarungnirun

Amirhossein Mirhosseini
Hamid Sarbazi-Azad
Babak Falsafi
Onur Mutlu