LDetector: A low overhead data race detector for GPU programs

1

PENGCHENG LI
CHEN DING
XIAOYU HU
TOLGA SOYATA

UNIVERSITY OF ROCHESTER

Introduction & Contribution

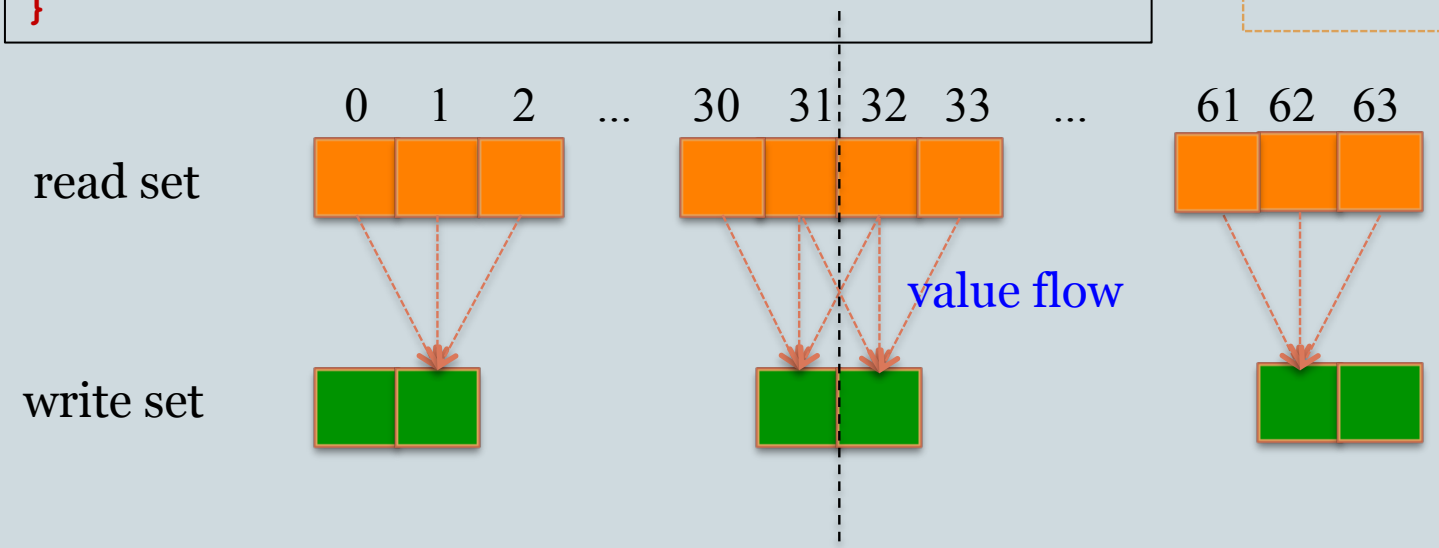
- Data races in GPU
  - Impact correctness as bugs
  - Impact determinism
  - Inter-warp data races and intra-warp data races

- Contribution
  - A two-pass approach to detect data races in GPU
    - As a debugging tool
    - Simple, but effective and low overhead
    - Zero per memory access monitoring
  - Seek other optimizations to reduce two-pass cost

A data race example


```
__global__ void Jacobi(int * data)
{
     extern __shared__ int a[]; // size 64
     int tid = threadIdx.x ;
     ...
     if (tid < BLOCK_SIZE-1 && tid >0 )
         A[tid] = (A[tid-1] + A[tid] +
A[tid+1]) / 3;
     __syncthreads();
}
```

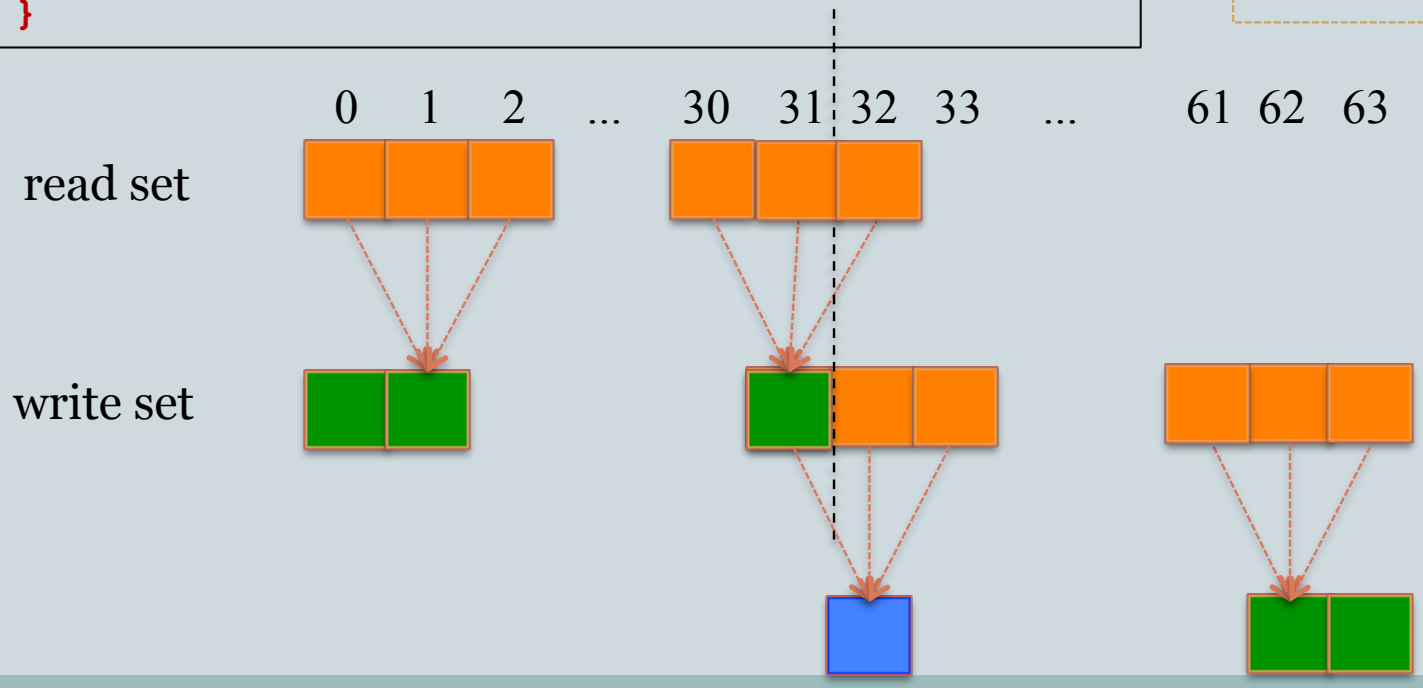
Array size: 64
BLOCK_SIZE: 64
WARP_SIZE: 32
# of Warps: 2
# of threads: 64



A data race example

```
__global__ void Jacobi(int * data)
{
     extern __shared__ int a[]; // size 64
     int tid = threadIdx.x ;
     ...
     if (tid < BLOCK_SIZE-1 && tid >0 )
         A[tid] = (A[tid-1] + A[tid] +
A[tid+1]) / 3;
     __syncthreads();
}
```

Array size: 64
BLOCK_SIZE: 64
WARP_SIZE: 32
# of Warps: 2
# of threads: 64



A data race example

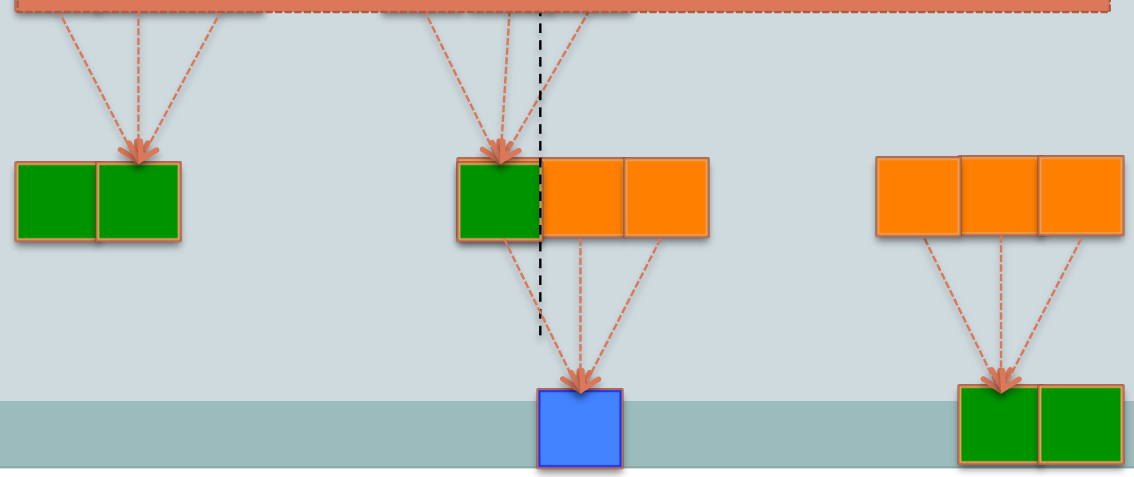

```
__global__ void Jacobi(int * data)
{
      extern __shared__ int a[]; // size 64
      int tid
      ...
      if (tid
          A[t
A[tid+1]) /
      __synct
}
```

Array size: 64
BLOCK_SIZE: 64
_SIZE: 32
arps: 2
reads: 64

- Application scenario:
  - Inter-warp data races
  - Shared array (GPU internal) oriented
  - No atomics in code regions

read set

write set



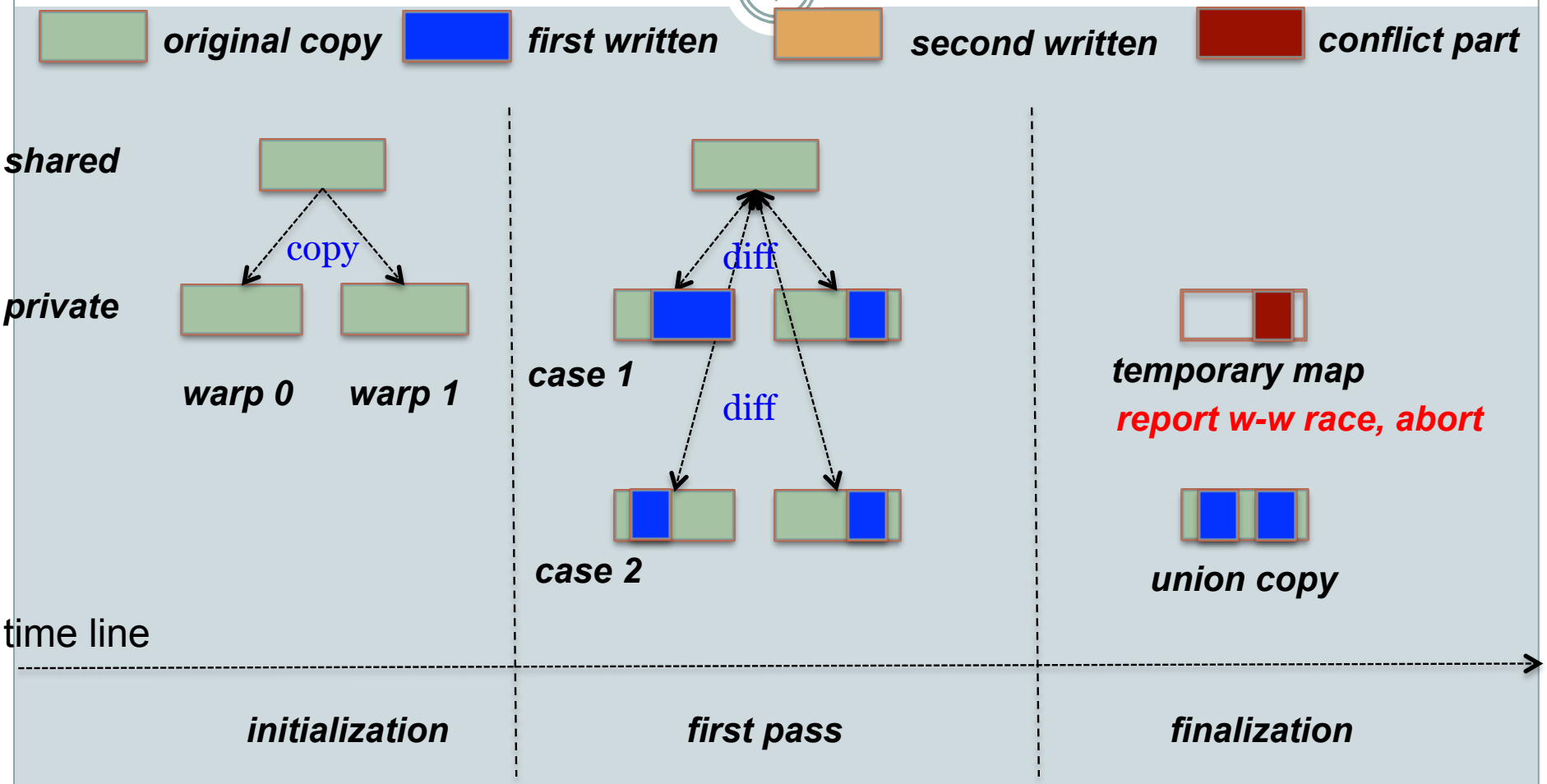
Shared array oriented data race detection

- Synchronized code block

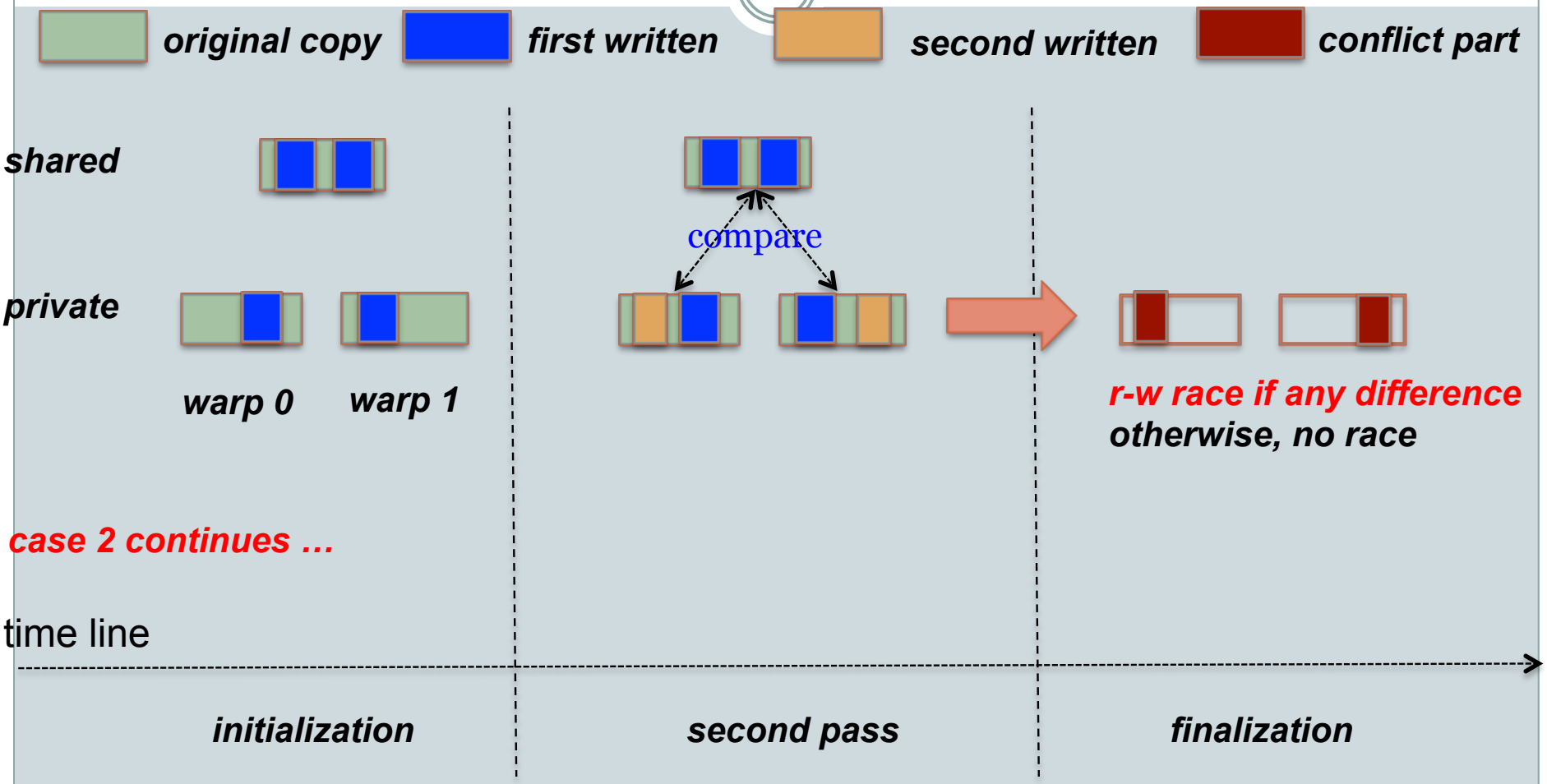
  A code region between two adjacent synchronization statements is called a synchronized block. Inter-warp data races on a shared array happen only in the same synchronized block.

- API: __syncthreads()
- A two-pass approach
  - 1st pass : write-write data race detection
  - 2nd pass: read-write data race detection

First pass: write-write race detection



Second pass: read-write race detection

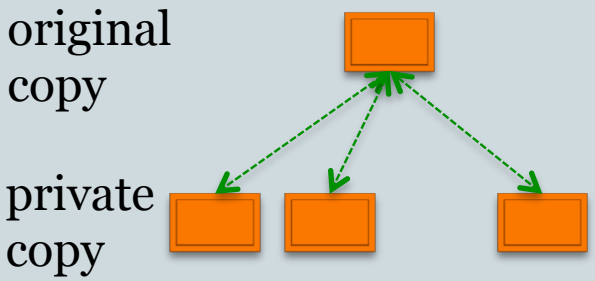


Correctness

- Correctness of write-write race detection
  - Value-based check, region overlaps mean write-write conflicts
- Correctness of read-write race detection
  - On condition that no write-write races exist
  - If outputs differ, must be from read-write races
- Convergence

A efficient lightweight runtime and compiler supports

- *Four operations: Diff, union, combine, compare*
  - Massive parallelism means high efficiency for bulk operations
  - Take diff as example

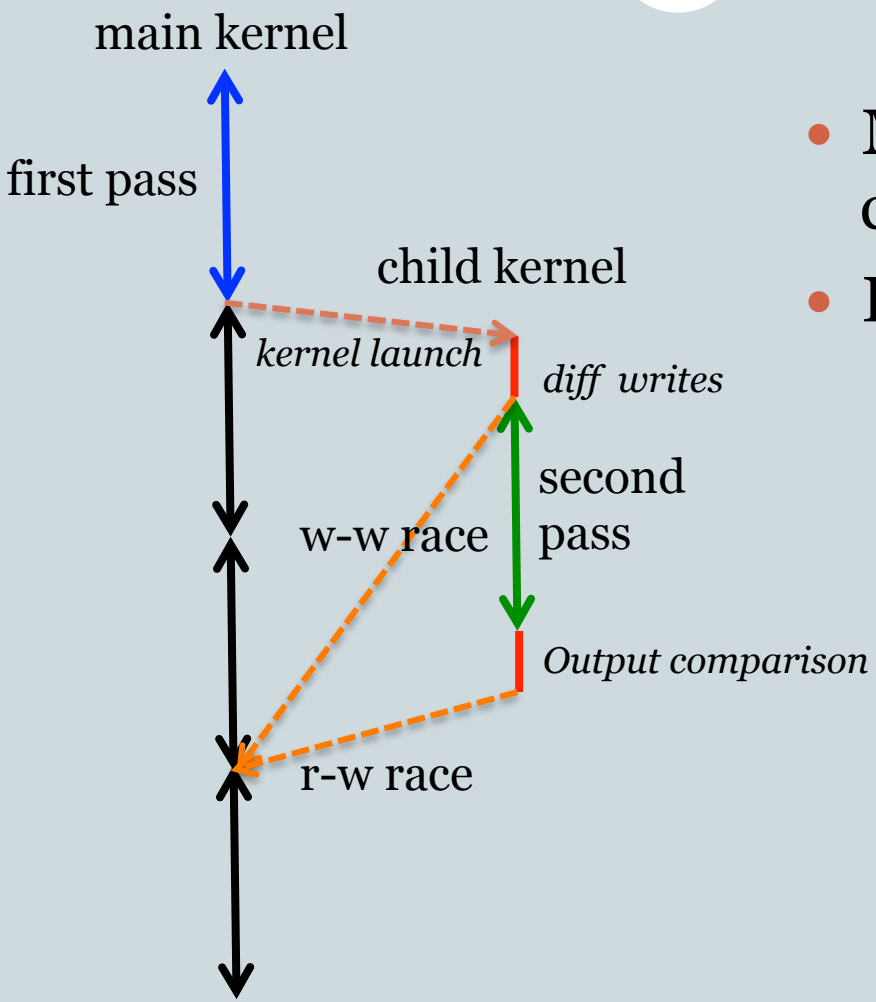


- Two-level parallelism
  - Warp-level parallelism (32 warps)
  - Intra-warp thread-level parallelism (32 threads)
- Well scalability

  - Byte-wise operations

- Data duplication
  - Data structure promotion in compiler

Using speculation to reduce two-pass overhead



- Main kernel speculatively continues execution
- Executions overlap

Evaluation

- Platform settings
  - Host machine: Xeon E5-2643 with 3.3GHz, 16GB memory
  - Device: Nvidia Tesla K20c GPU
    - 1 Kepler GK110 architecture, 2496 cores, 208GB/sec memory bandwidth, 5GB memory
    - Dynamic parallelism
- Two benchmarks on data mining
  - EM
  - Co-clustering
- Compared with GRace[Zheng et al. PPoPP'11]

Performance overhead

Comparisons	EM	Co-clustering
native	60.27(1x)	28.91(1x)
LDetector	250.15(4.15x)	99.46(3.44x)
LDetector-spec	177.42(2.94x)	107.04(3.70x)
GRace-stmt	104445.9(1733x)	1643781.88(56858x)
GRace-addr	18866.83(313x)	7236.8(250x)

- measured time in milliseconds (slowdown)

Memory overhead

Comparisons	EM	Co-clustering
LDetector	16K, 2.6M	16K, 2M
LDetector-spec	0K, 2.7M	0K, 2.2M
GRace-stmt	1.1K, 1000M*	1.1K, 1000M*
GRace-addr	1.1K, 18M	1.1K, 9M

- shared memory usage, global memory usage
- * Instrument as many memory accesses when memory overhead gets up to 1GB

Future work

- Conclusion
  - A two-pass approach to detect data races in shared memory
  - Zero per memory access monitoring
  - A high efficient lightweight runtime
  - Speculative parallelization optimization

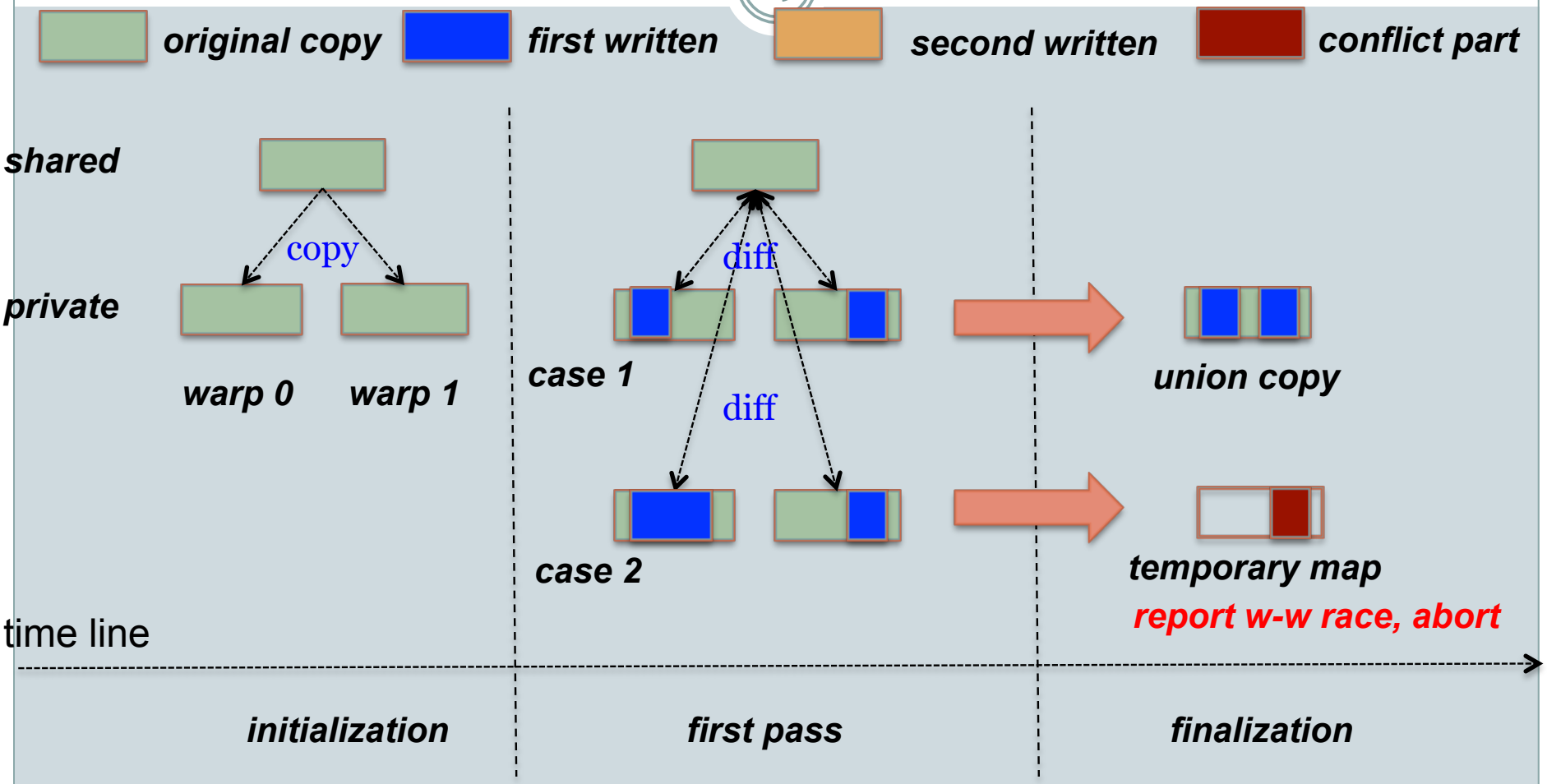
- Future work
  - Support atomics, locks
  - Solutions on data races in global memory
    - Memory consumption explosion

- Thanks !
- Q & A

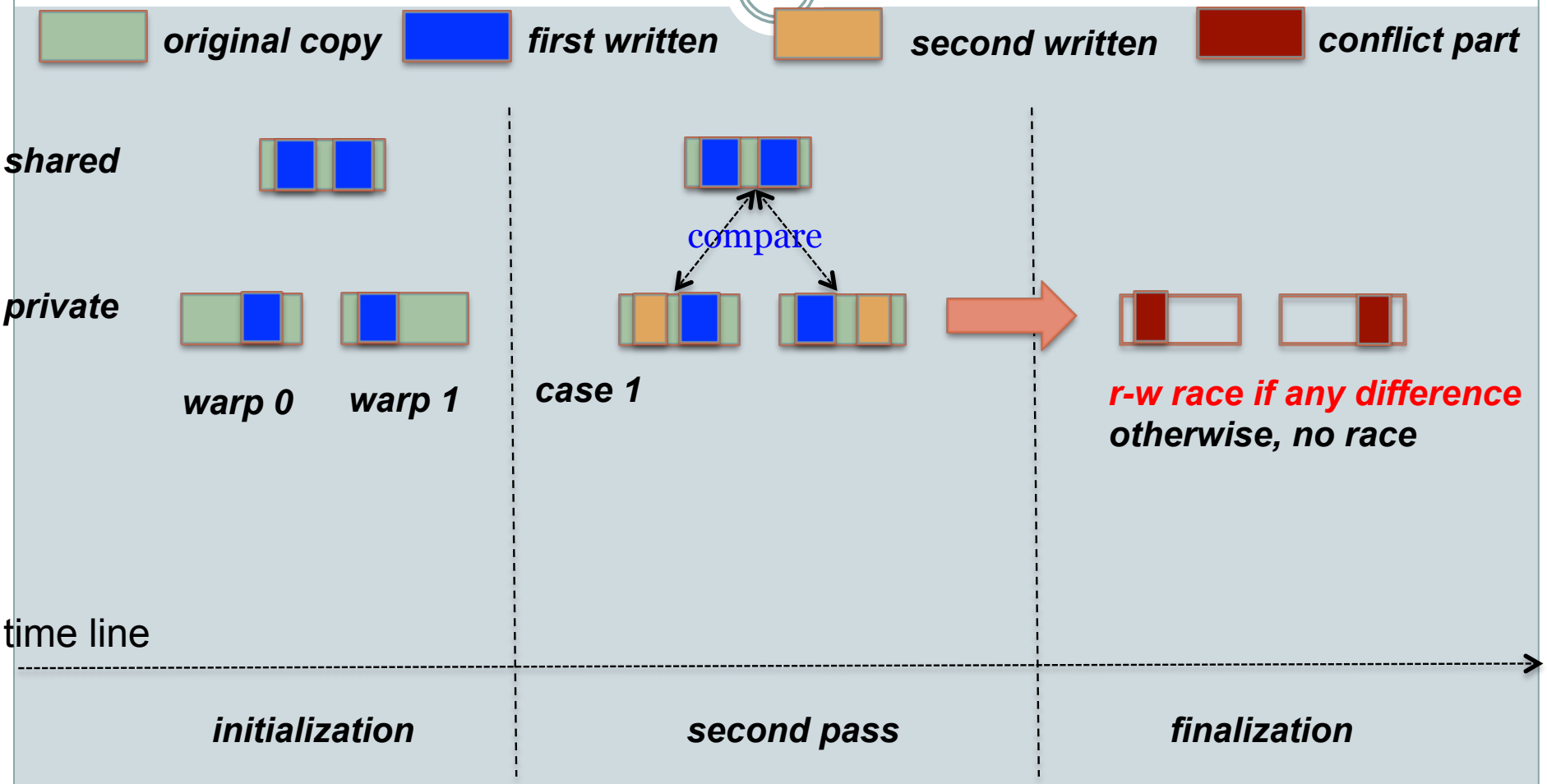
Data structure expansion

- Data privatization techniques

First pass: write-write race detection



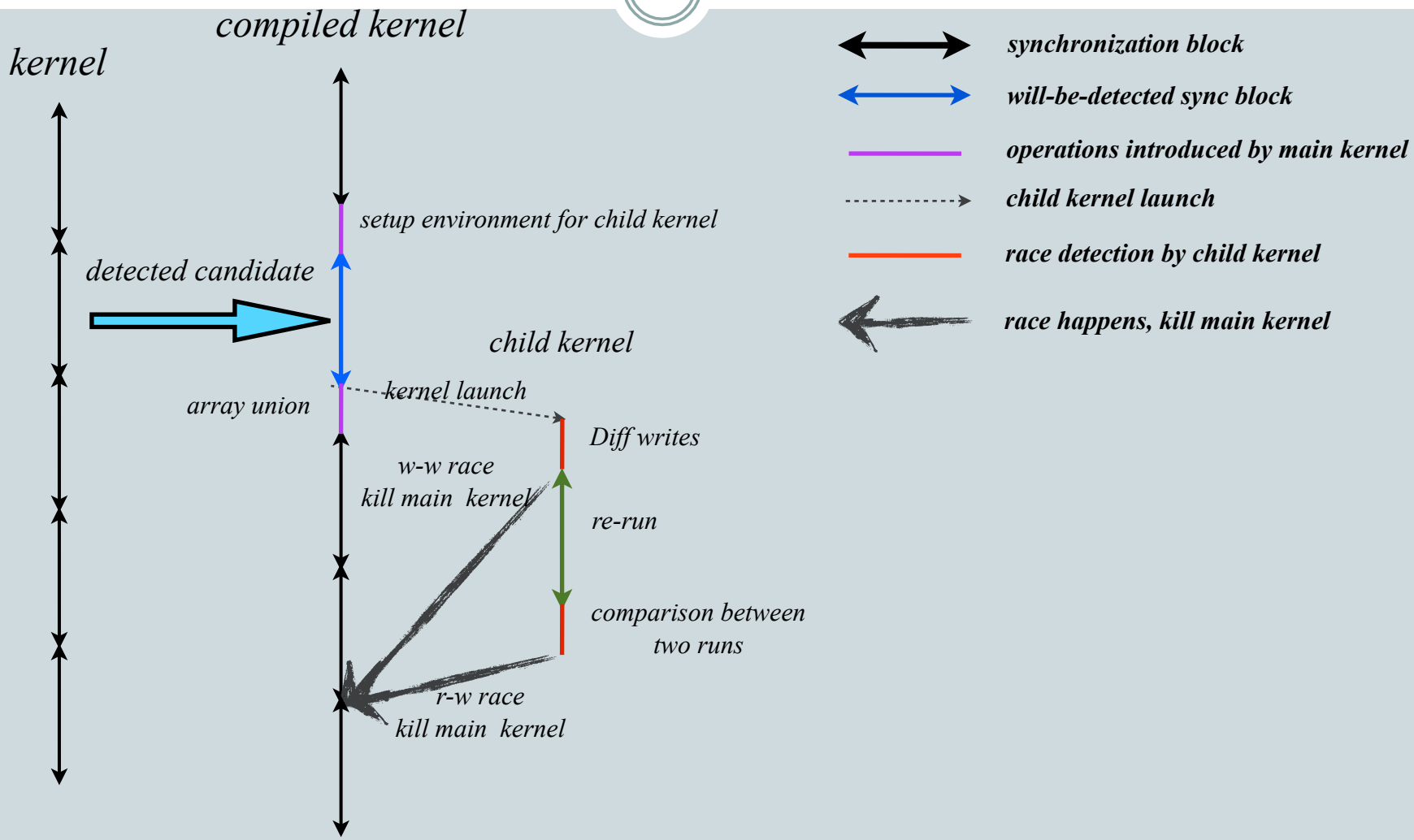
Second pass: read-write race detection



Our contribution

- Target:
  - Shared array oriented
  - With no atomics in code regions
  - Inter-warp data races

Using speculation to reduce two-pass overhead



A two-pass approach