Shredder

GPU-Accelerated Incremental Storage and Computation

Pramod Bhatotia[§],

Rodrigo Rodrigues[§], Akshat Verma[¶]

[§]MPI-SWS, Germany

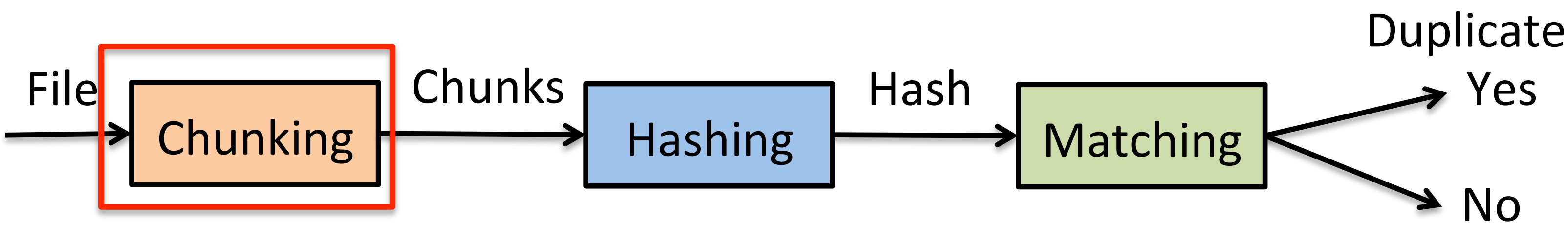
[¶]IBM Research-India

USENIX FAST 2012

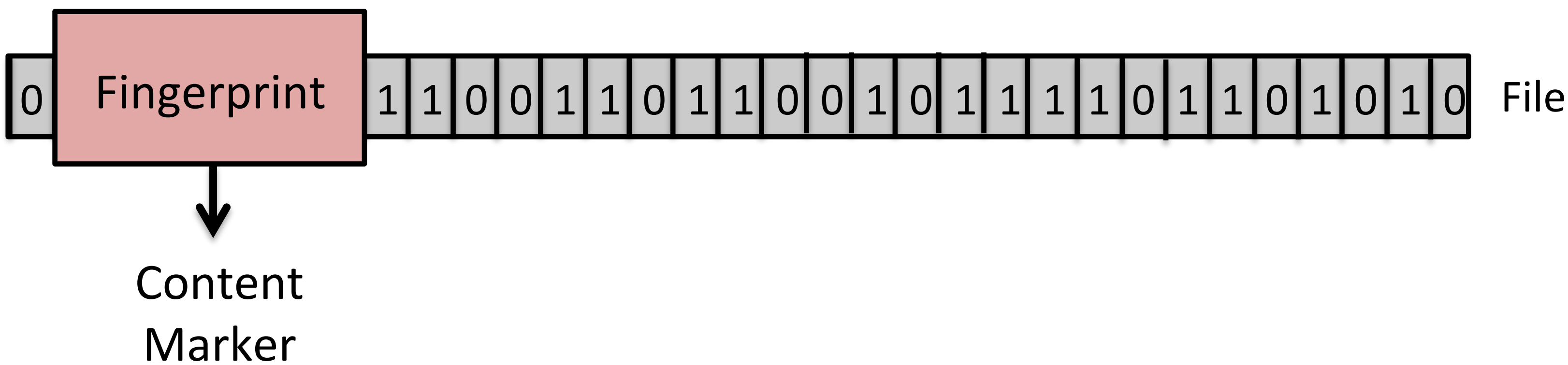
Handling the data deluge

- Data stored in data centers is growing at a fast pace
- Challenge: How to store and process this data ?
  - Key technique: Redundancy elimination
- Applications of redundancy elimination
  - Incremental storage: data de-duplication
  - Incremental computation: selective re-execution

Redundancy elimination is expensive



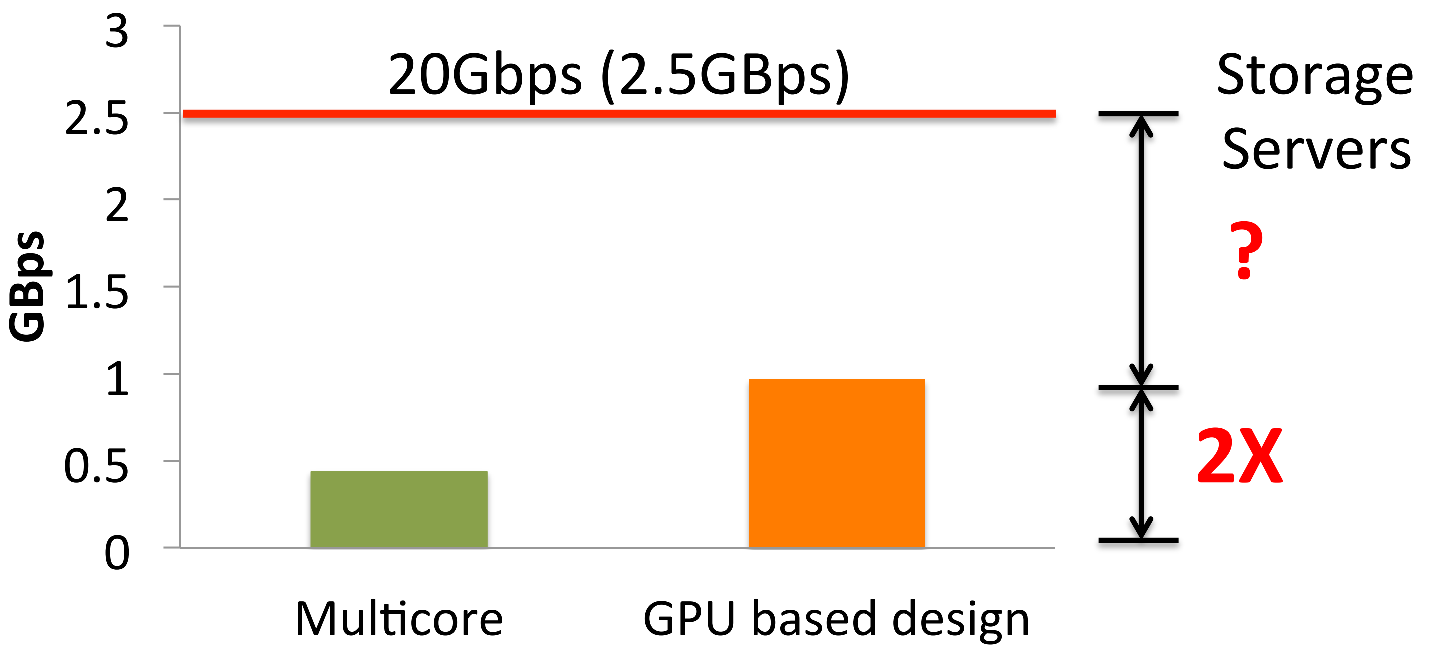
Content-based chunking [SOSP'01]



For large-scale data, chunking easily becomes a bottleneck

Accelerate chunking using GPUs

GPUs have been successfully applied to compute-intensive tasks

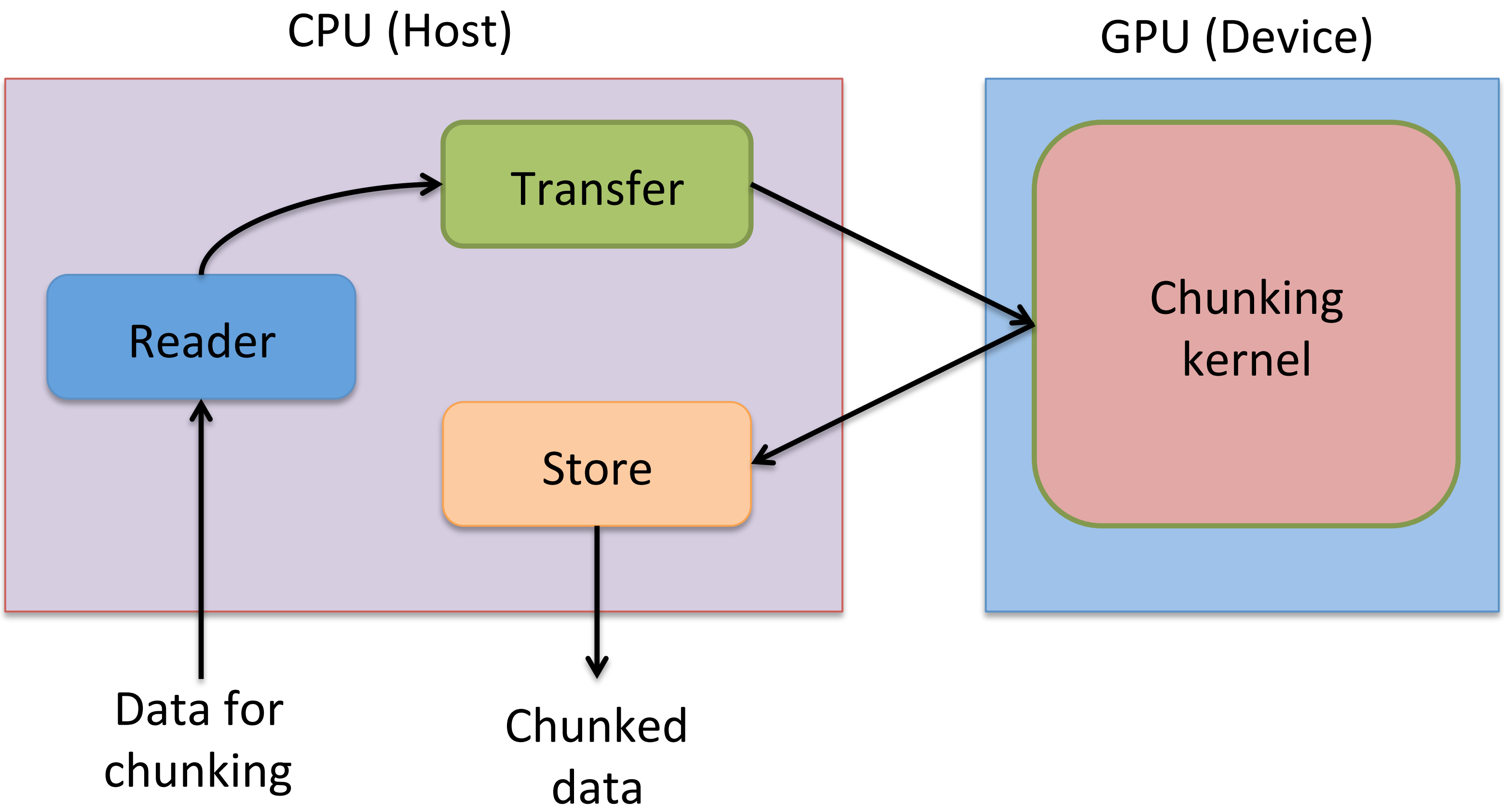


Using GPUs for data-intensive tasks presents new challenges

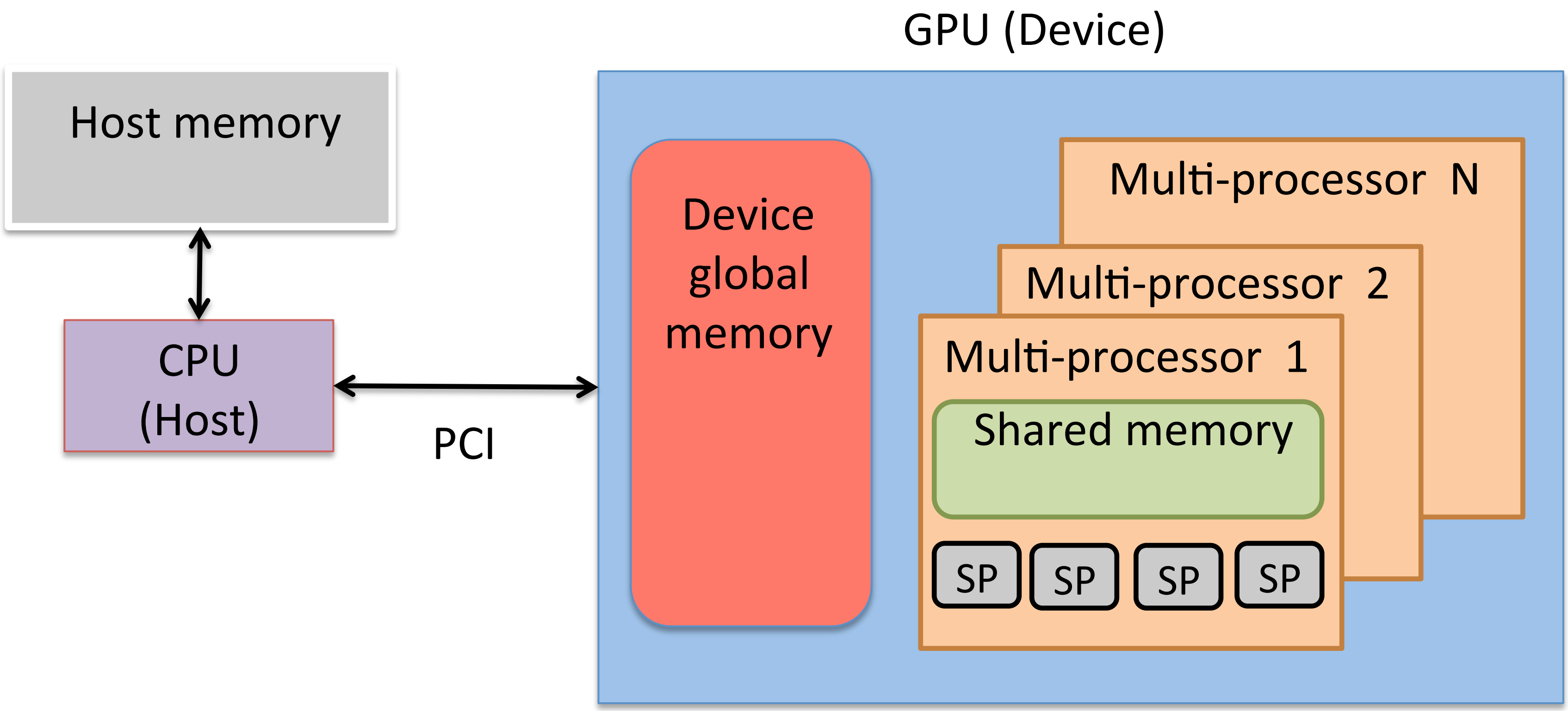
Rest of the talk

- Shredder design
  - Basic design
  - Background: GPU architecture & programming model
  - Challenges and optimizations
- Evaluation
- Case studies
  - Computation: Incremental MapReduce
  - Storage: Cloud backup

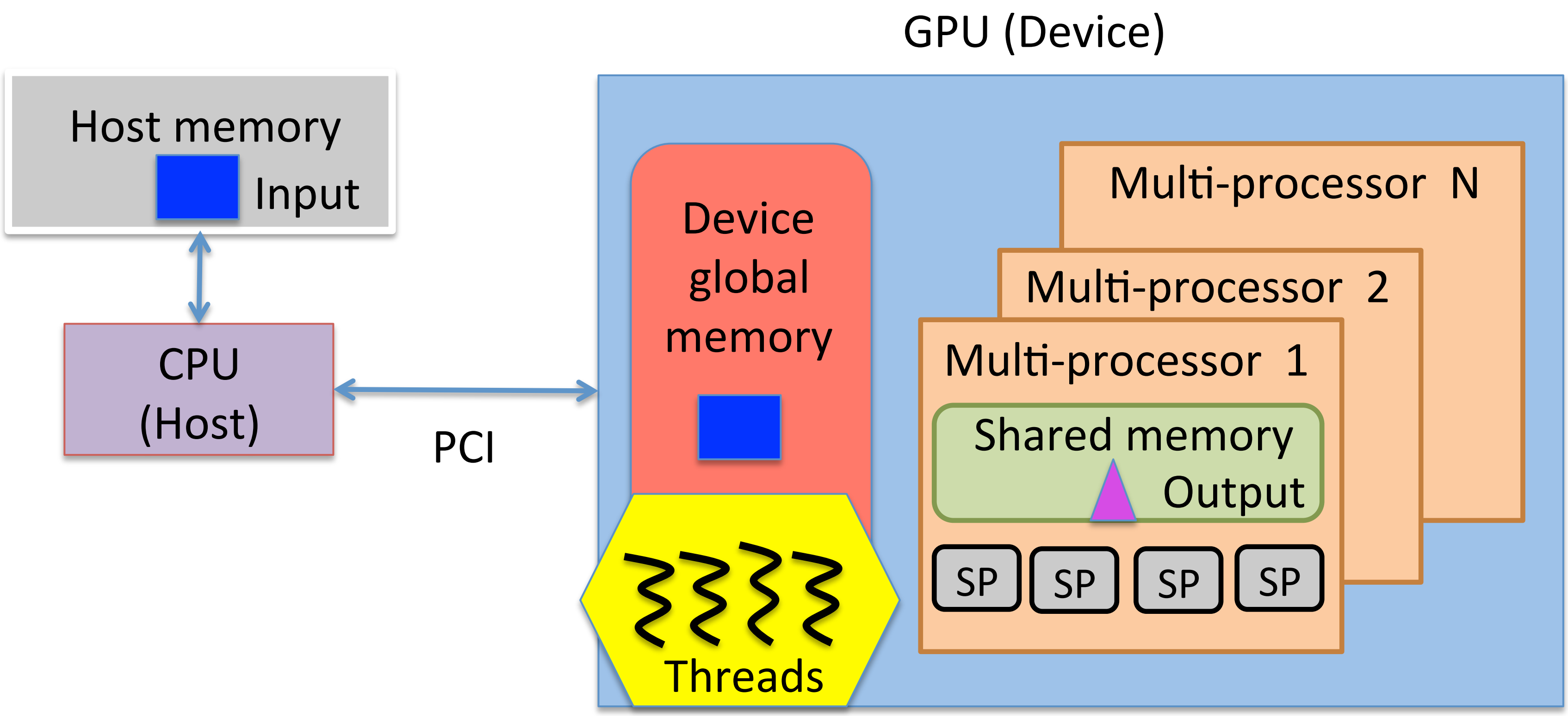
Shredder basic design



GPU architecture



GPU programming model

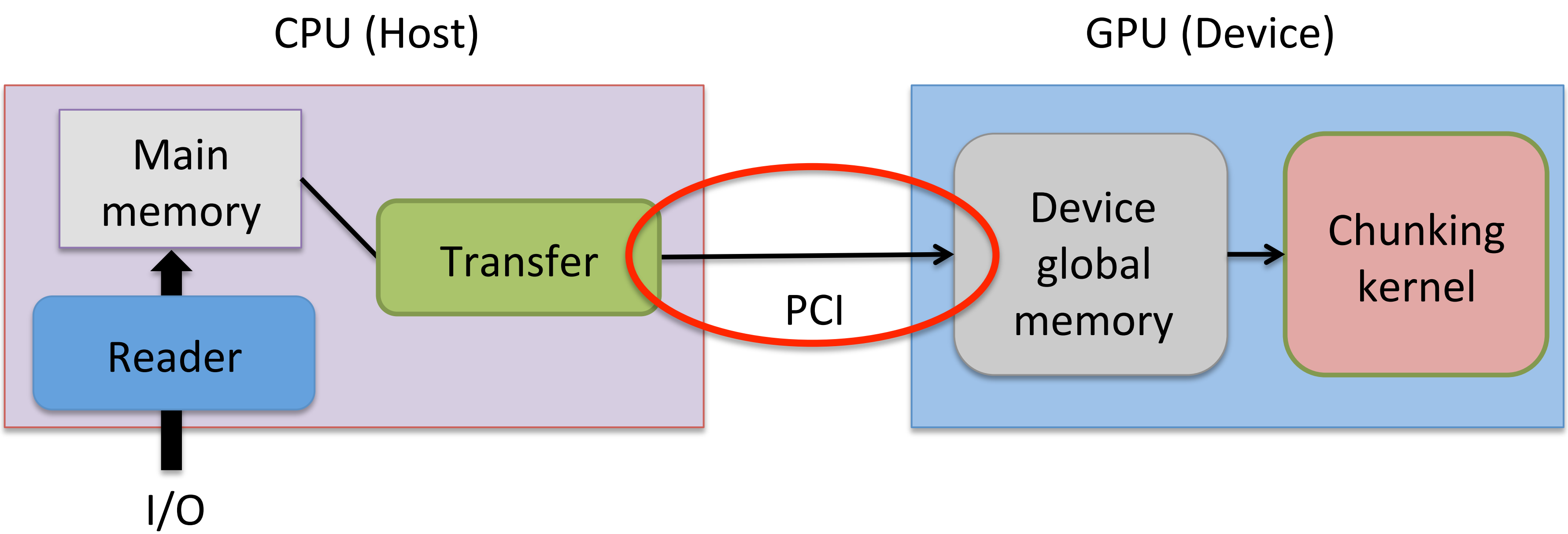


Scalability challenges

1. Host-device communication bottlenecks
2. Device memory conflicts
3. Host bottlenecks (See paper for details)

Challenge # 1

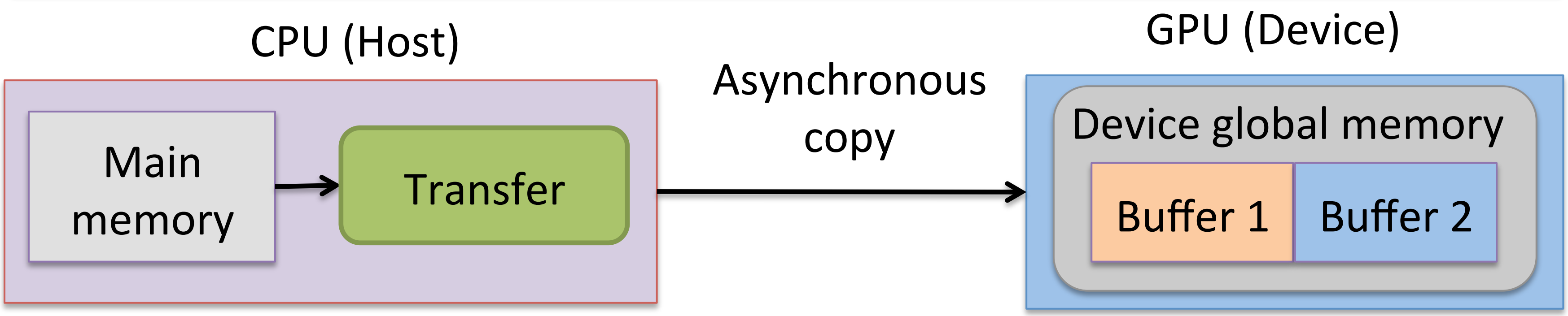
Host-device communication bottleneck



Synchronous data transfer and kernel execution

- Cost of data transfer is comparable to kernel execution
- For large-scale data it involves many data transfers

Asynchronous execution

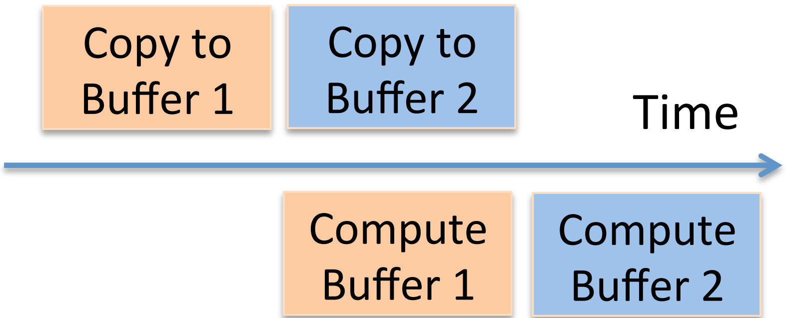


Pros:

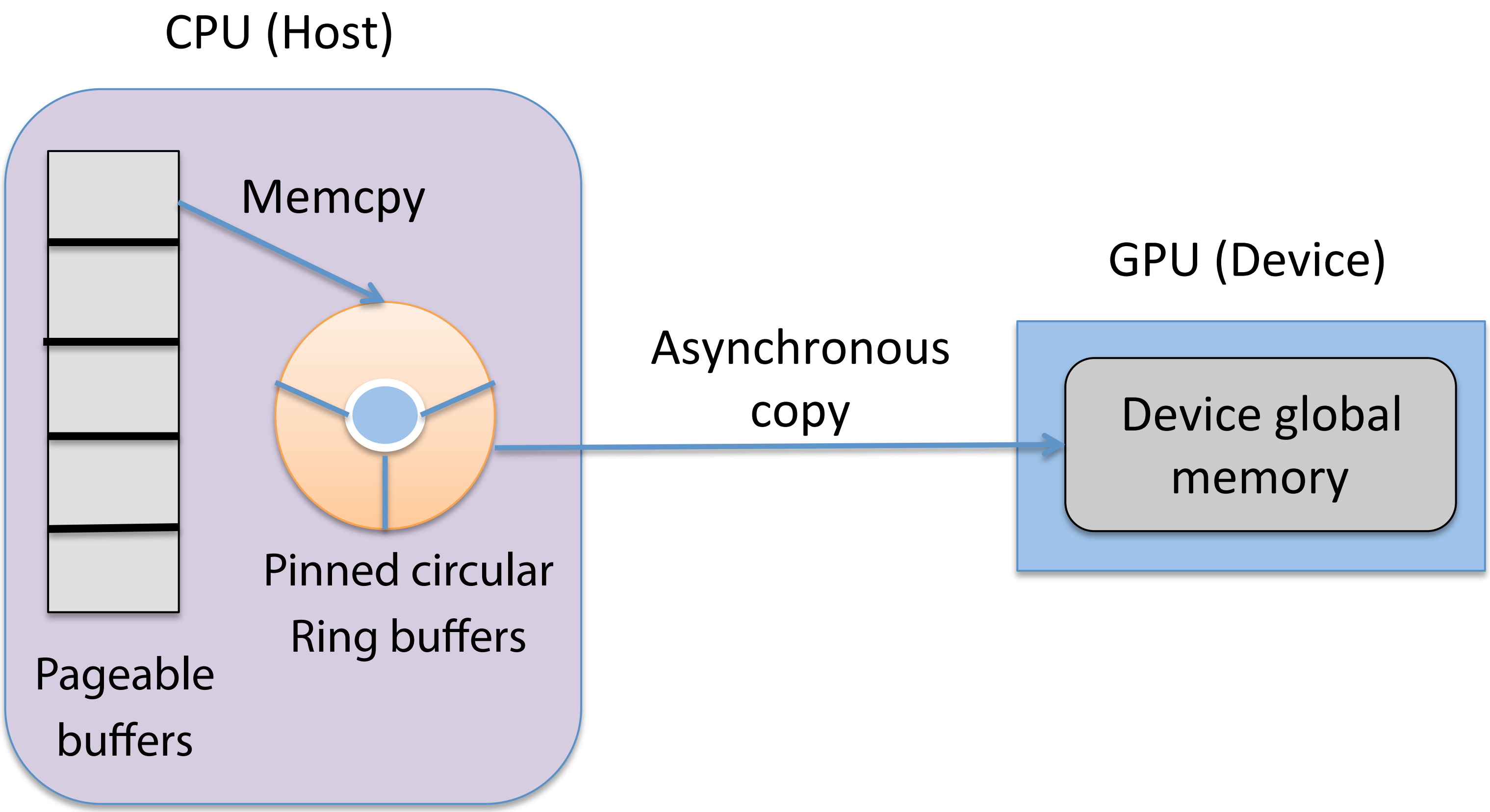
+ Overlaps communication with computation
+ Generalizes to multi-buffering

Cons:

- Requires page-pinning of buffers at host side

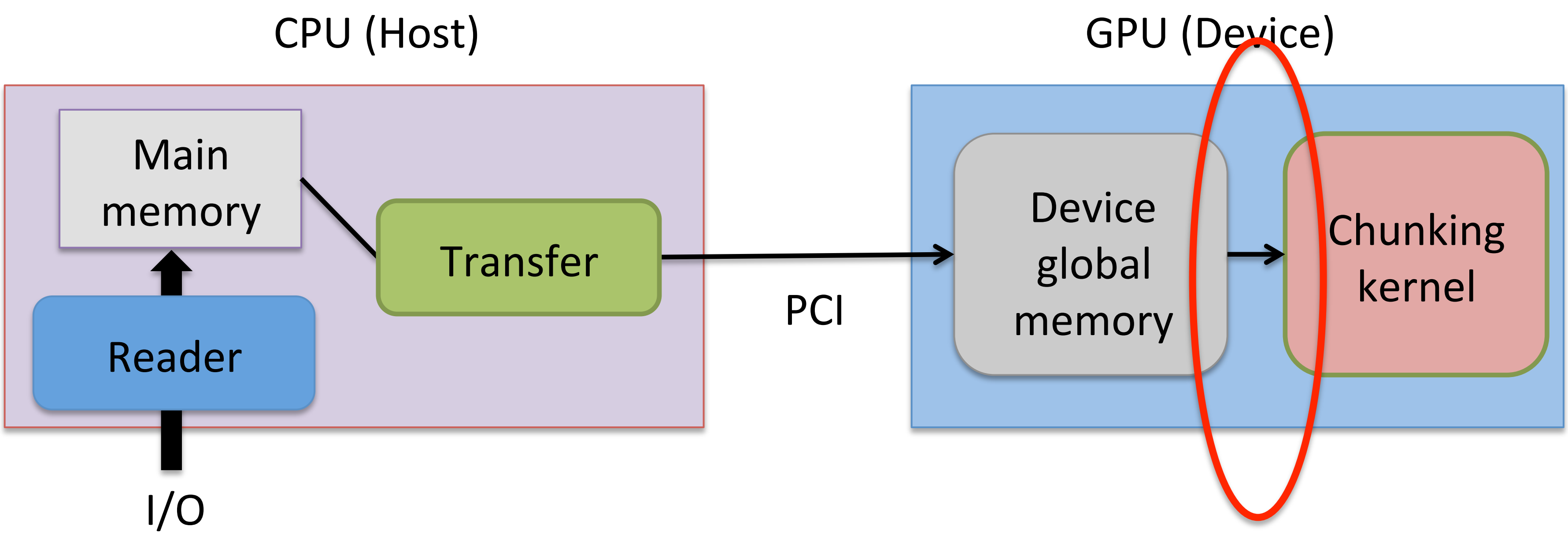


Circular ring pinned memory buffers

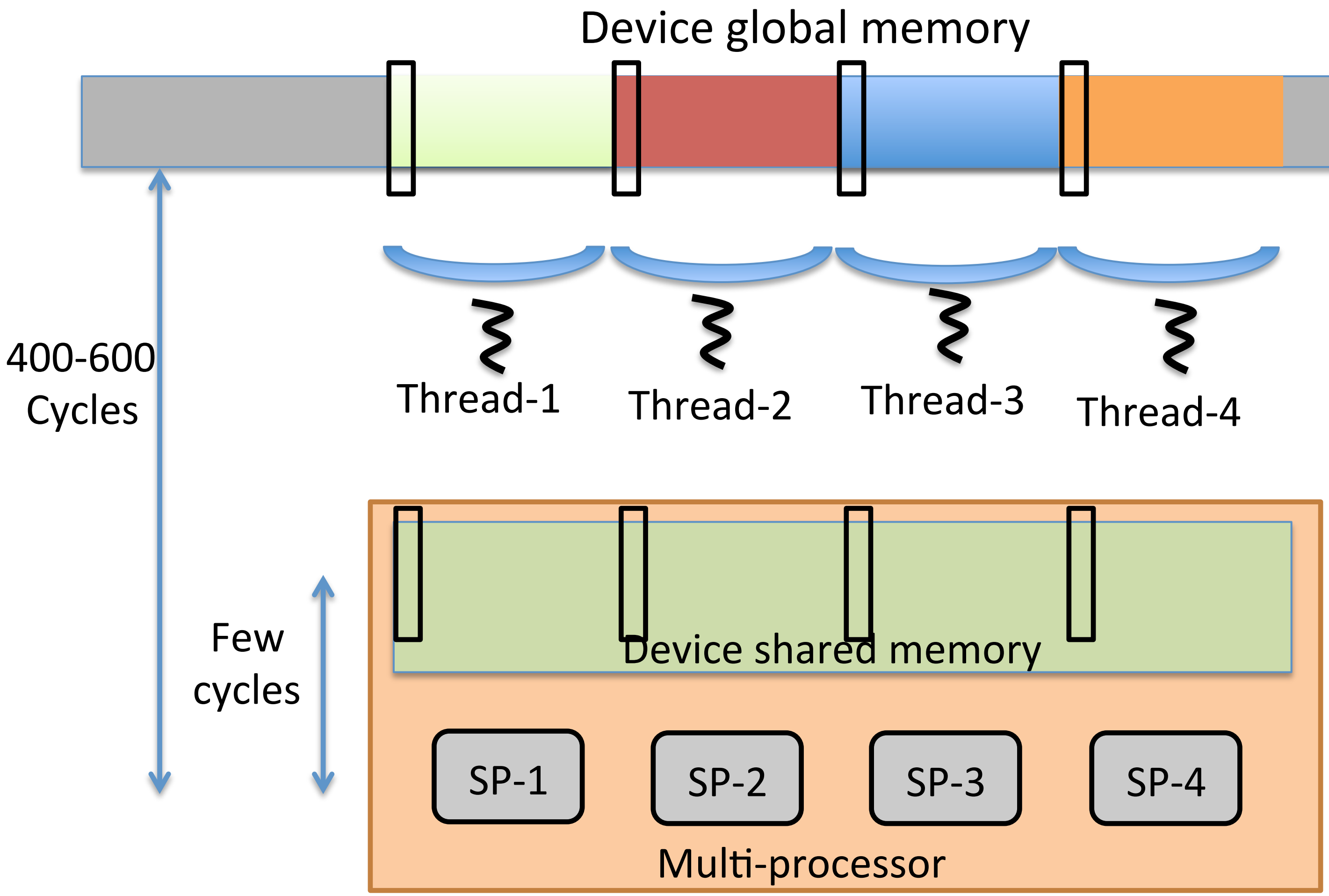


Challenge # 2

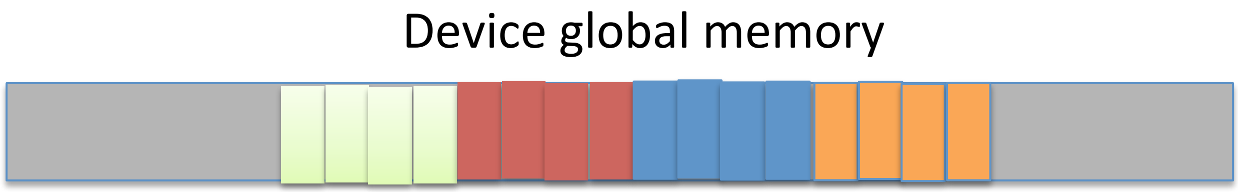
Device memory conflicts



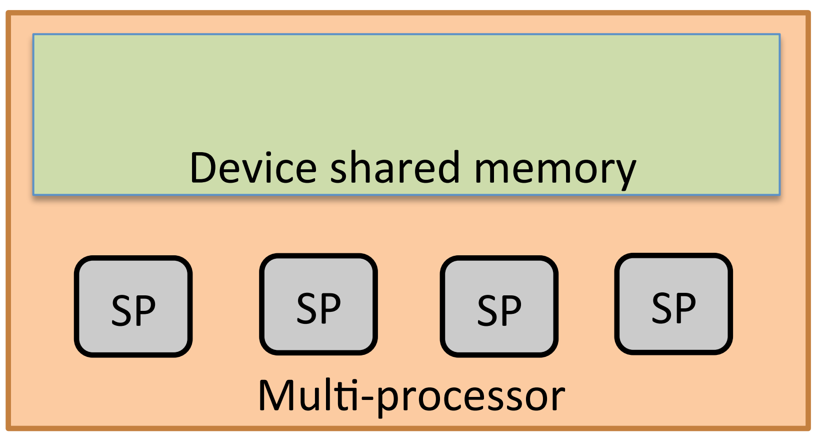
Accessing device memory



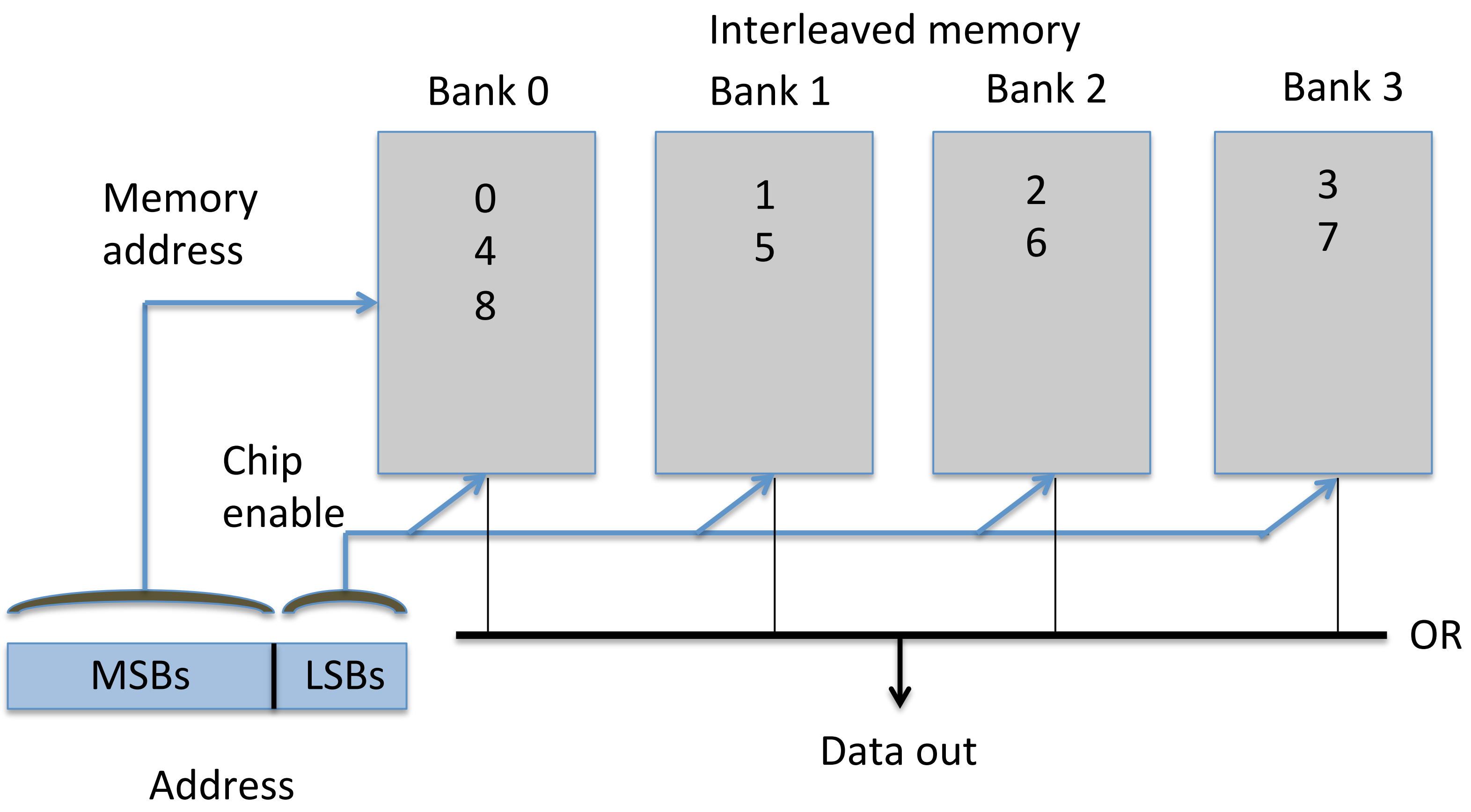
Memory bank conflicts



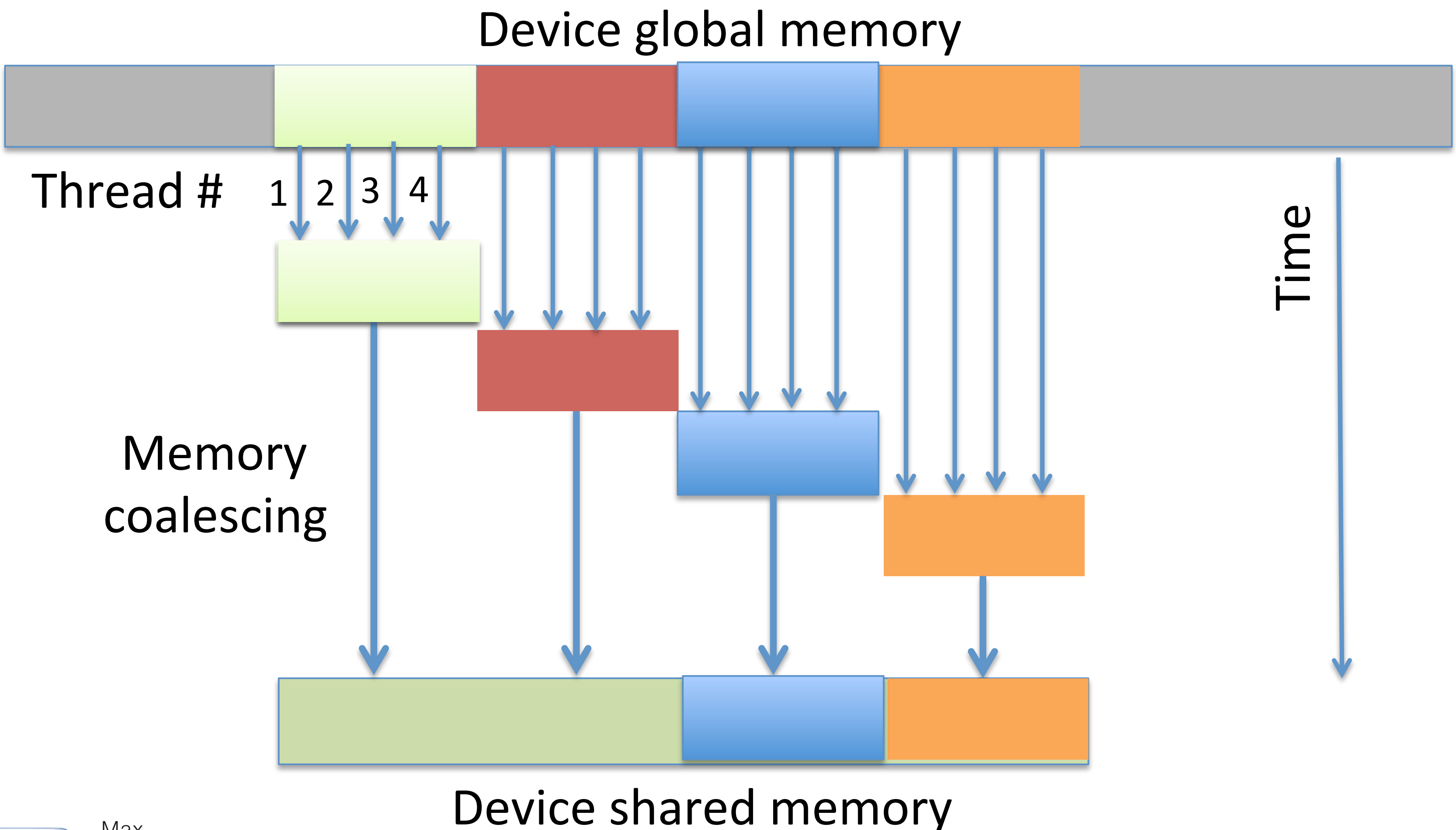
Un-coordinated accesses to global memory lead to a large number of memory bank conflicts



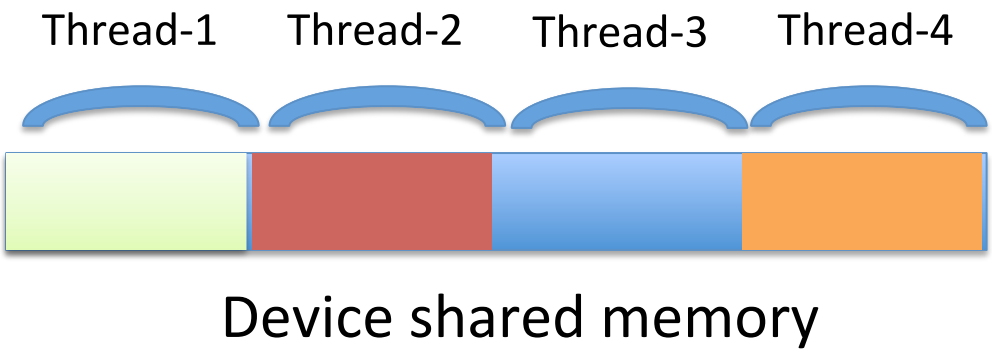
Accessing memory banks



Memory coalescing



Processing the data



Outline

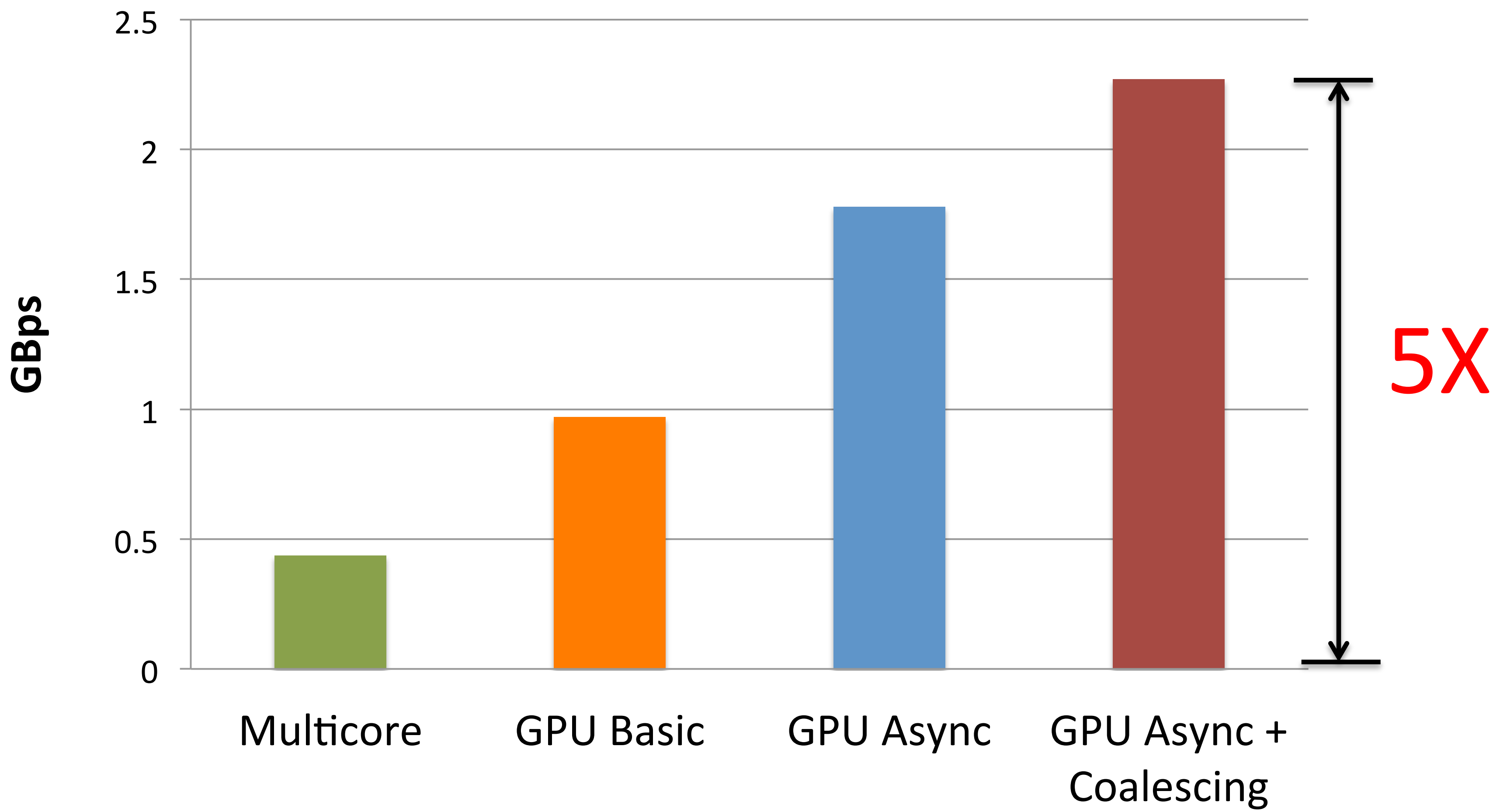
- ~~Shredder design~~
- Evaluation
- Case-studies

Evaluating Shredder

- Goal: Determine how Shredder works in practice
  - How effective are the optimizations? (See paper for details)
  - How does it compare with multicores?

- Implementation
  - Host driver in C++ and GPU in CUDA
  - GPU: NVidia Tesla C2050 cards
  - Host machine: Intel Xeon with12 cores

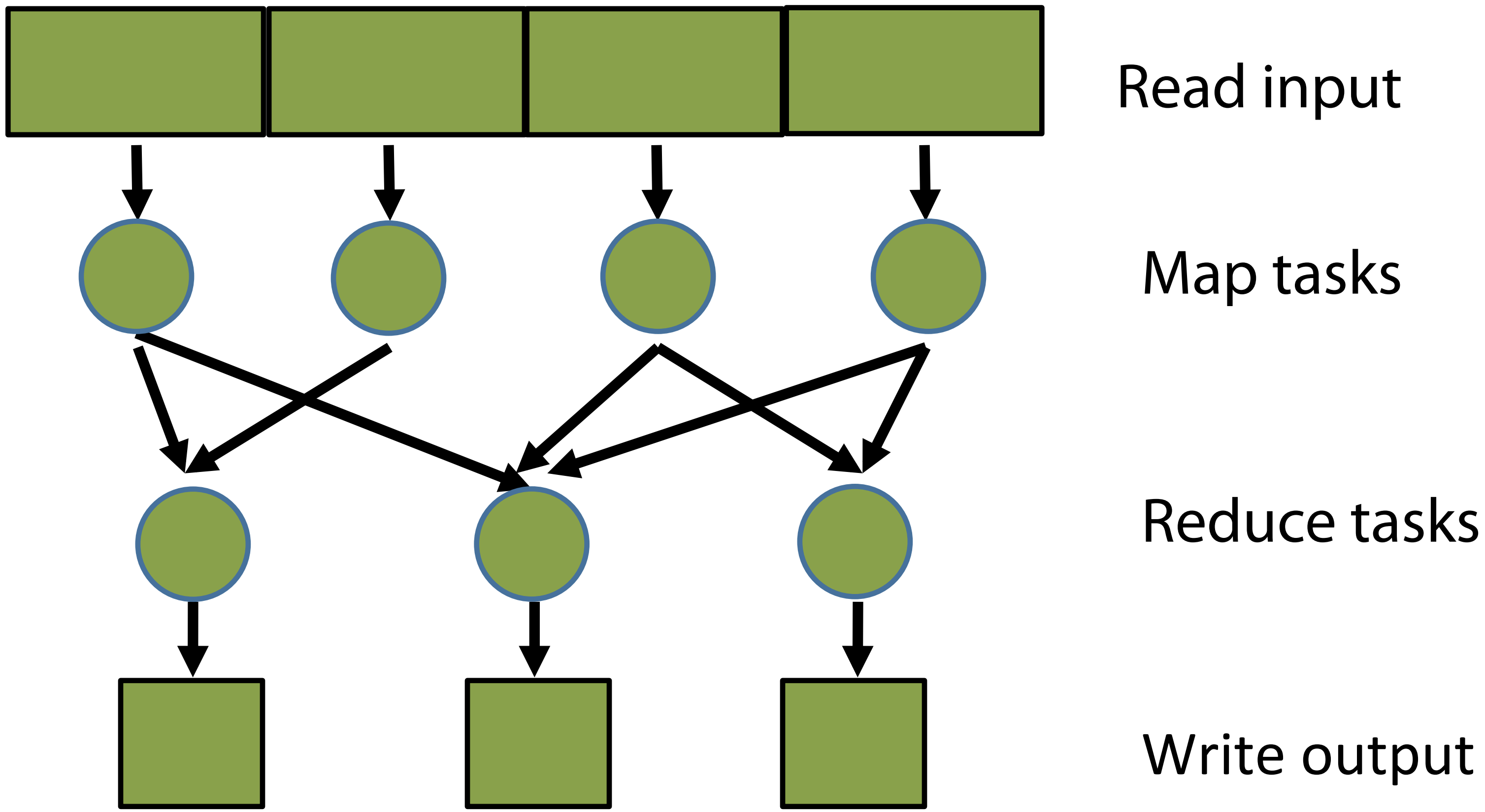
Shredder vs. Multicores



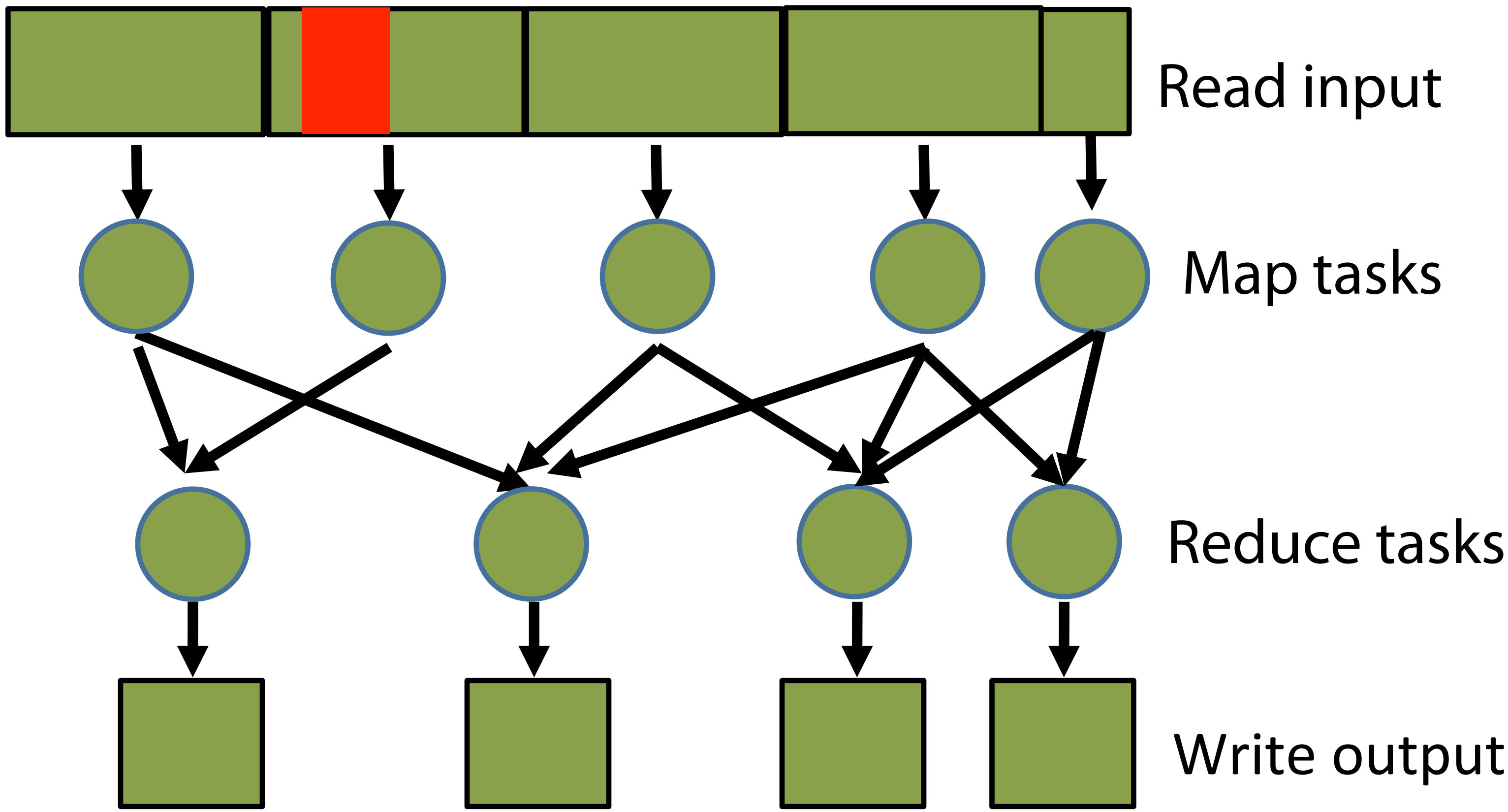
Outline

- ~~Shredder design~~
- ~~Evaluation~~
- Case studies
  - Computation: Incremental MapReduce
  - Storage: Cloud backup (See paper for details)

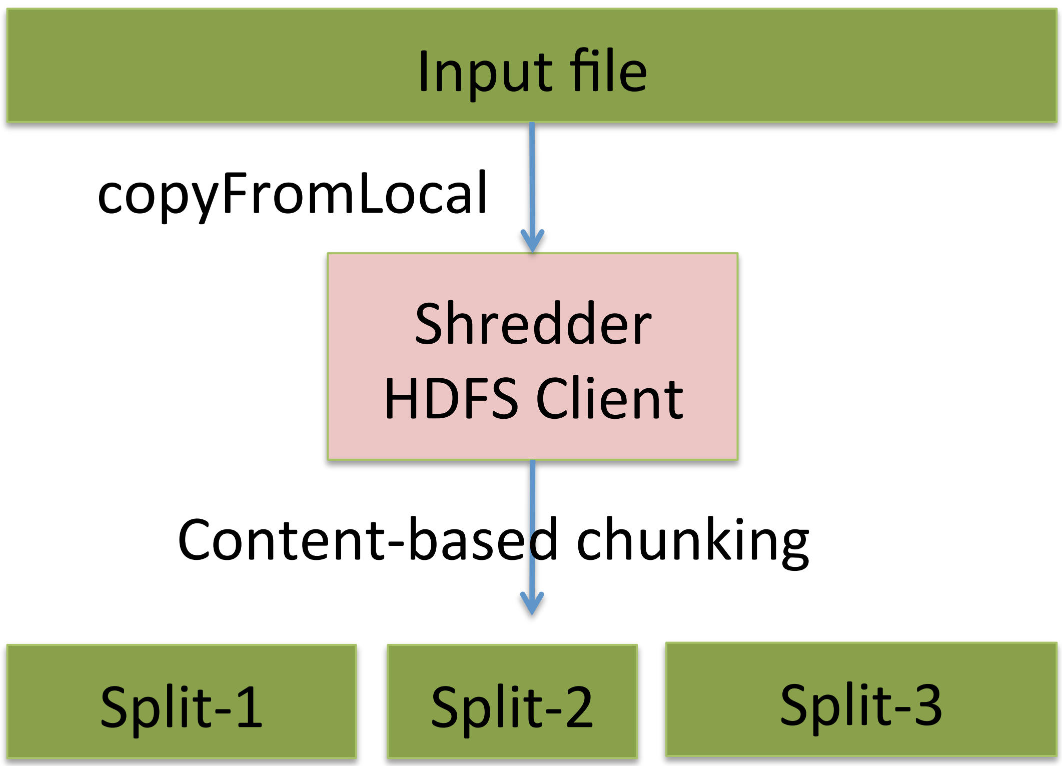
Incremental MapReduce



Unstable input partitions



GPU accelerated Inc-HDFS



Related work

- GPU-accelerated systems
  - Storage: Gibraltar [ICPP'10], HashGPU [HPDC'10]
  - SSLShader[NSDI'11], PacketShader[SIGCOMM'10], …

- Incremental computations
  - Incoop[SOCC'11], Nectar[OSDI'10], Percolator[OSDI'10],…

Conclusions

- GPU-accelerated framework for redundancy elimination
    - Exploits massively parallel GPUs in a cost-effective manner
- Shredder design incorporates novel optimizations
    - More data-intensive than previous usage of GPUs
- Shredder can be seamlessly integrated with storage systems
    - To accelerate incremental storage and computation

Thank You!