SPIN:

Seamless Operating System Integration of Peer-to-Peer DMA Between SSDs and GPUs

Shai Bergman | Tanya Brokhman | Tzachi Cohen | Mark Silberstein

ACSL - Technion

Summary

- You are here

What do we do?

Enable efficient file I/O for GPUs

Why?

Support diverse I/O workloads involving GPUs

How?

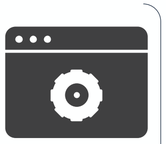
Make P2P a first class citizen within the file I/O stack

Results

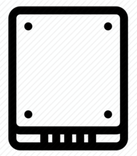
Better throughput
Standard file API
cross-GPU portability

Background

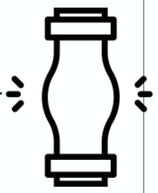
Fast data transfers



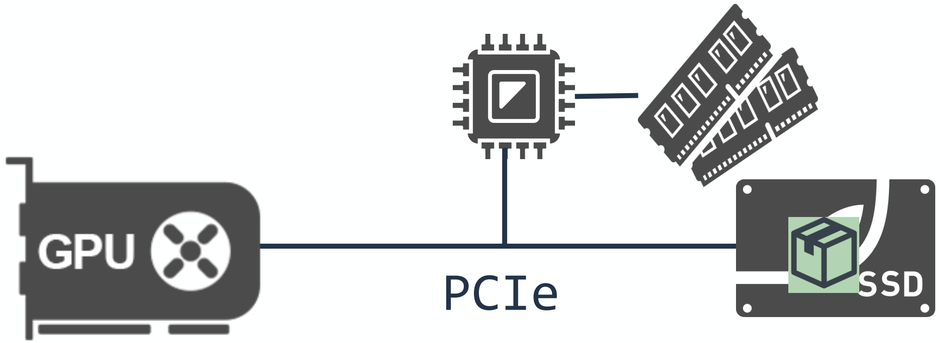
Data resides in SSD



Bounded by extra copy



CPU mediated data transfers introduce extra latency with lower throughput



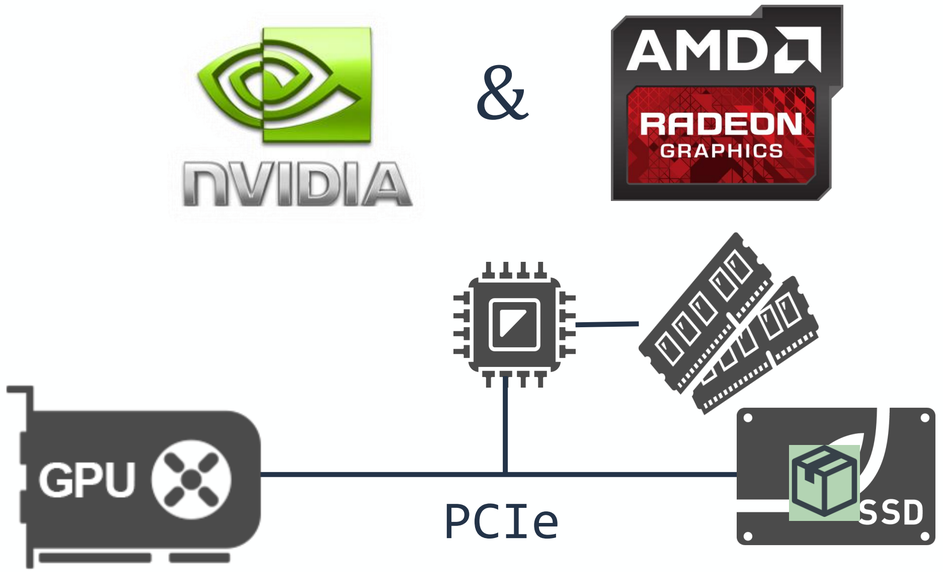
CPUIO - CPU mediated transfer

- You are here

Background

- You are here

GPU vendors support P2P



Eliminates redundant copies

Not involved in data path

Better power efficiency

Higher throughput & lower latency

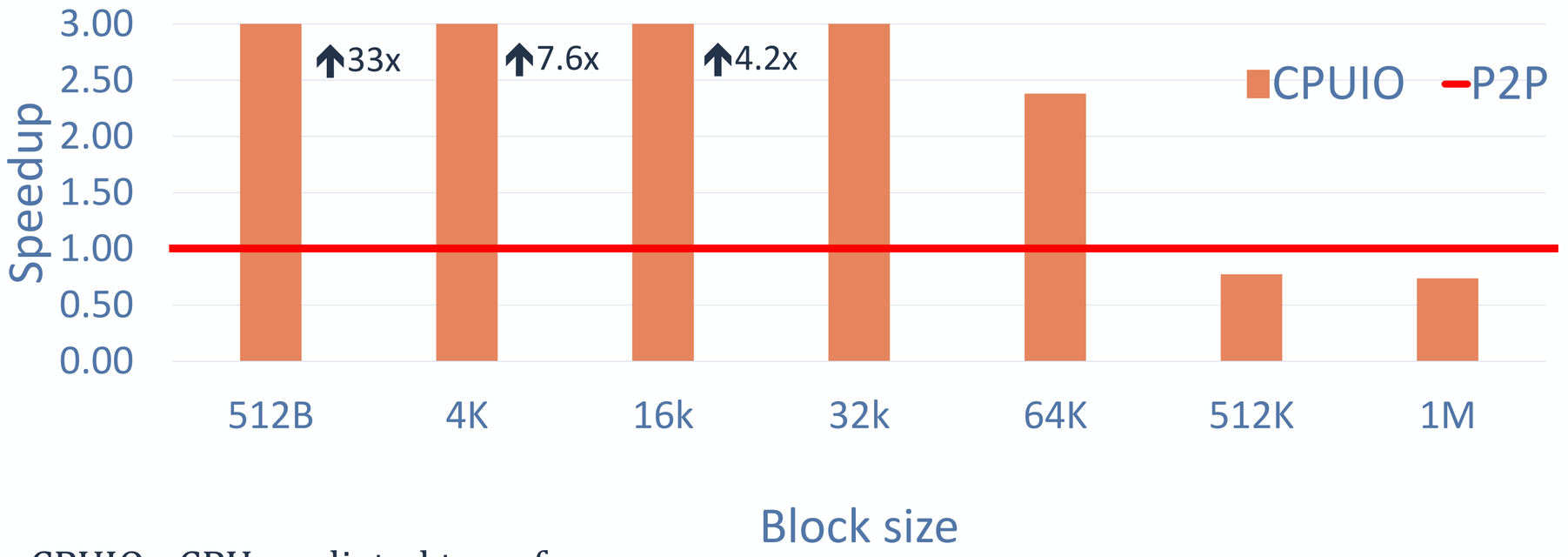
[1] J. Zhang, D. Donofrio, J. Shalf, M. T. Kandemir, and M. Jung, “NVMMU: A Non-volatile Memory Management Unit for Heterogeneous GPU-SSD Architectures,”
[2] H.-W. Tseng, Y. Liu, M. Gahagan, J. Li, Y. Jin, and S. Swanson, “Gullfoss: Accelerating and Simplifying Data Movement Among Heterogeneous Computing and Storage Resources,”
[3] M. Shihab, K. Taht, and M. Jung, “GPUDrive: Reconsidering Storage Accesses for GPU Acceleration,”
[4] H.-W. Tseng, Q. Zhao, Y. Zhou, M. Gahagan, and S. Swanson, “Morpheus: creating application objects efficiently for heterogeneous computing,”
[5]“Project Donard.” https://github.com/sbates130272/donard, 2015.

Observations

P2P is great!*

* But wait – only for certain data access patterns

Sequential reads (e.g. grep)



CPUIO - CPU mediated transfer

Short sequential reads: P2P ~33x **Slower** than CPUIO?

**data is not preloaded to the page cache

● You are here

Observations

You are here

What went wrong?
P2P bypasses the kernel!

No Page Cache Integration

No read ahead | Cannot utilize P$ for data reuse

Hard to utilize

Non-standard API | No misaligned accesses | LVM/MDADM incompatible

No file consistency

Can read stale data | Requires explicit flushes to SSD

Objective

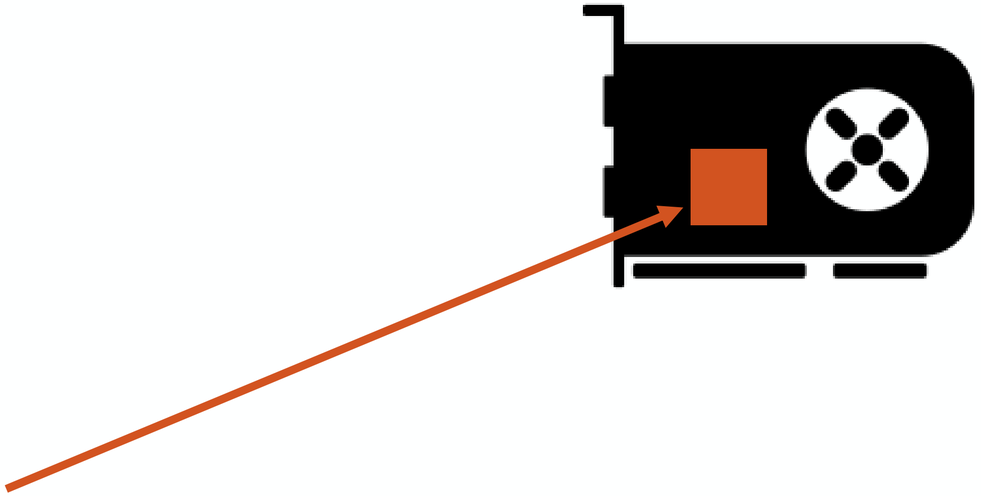
You are here

What do we want?
Regular file I/O to GPU memory

```
int fd;
...

//open file
...

pread64(fd,gpu_buffer,20*1024,0);
...
```



SPIN: Contributions

Standard File API

- Underlying block device support (RAID, LVM)
- Activate P2P when beneficial

Data Consistency + POSIX file semantics

Keep POSIX file semantics + data consistency, even when CPU + GPU work on the same file

Combine Page Cache and P2P

Interleave system memory and SSD when possible

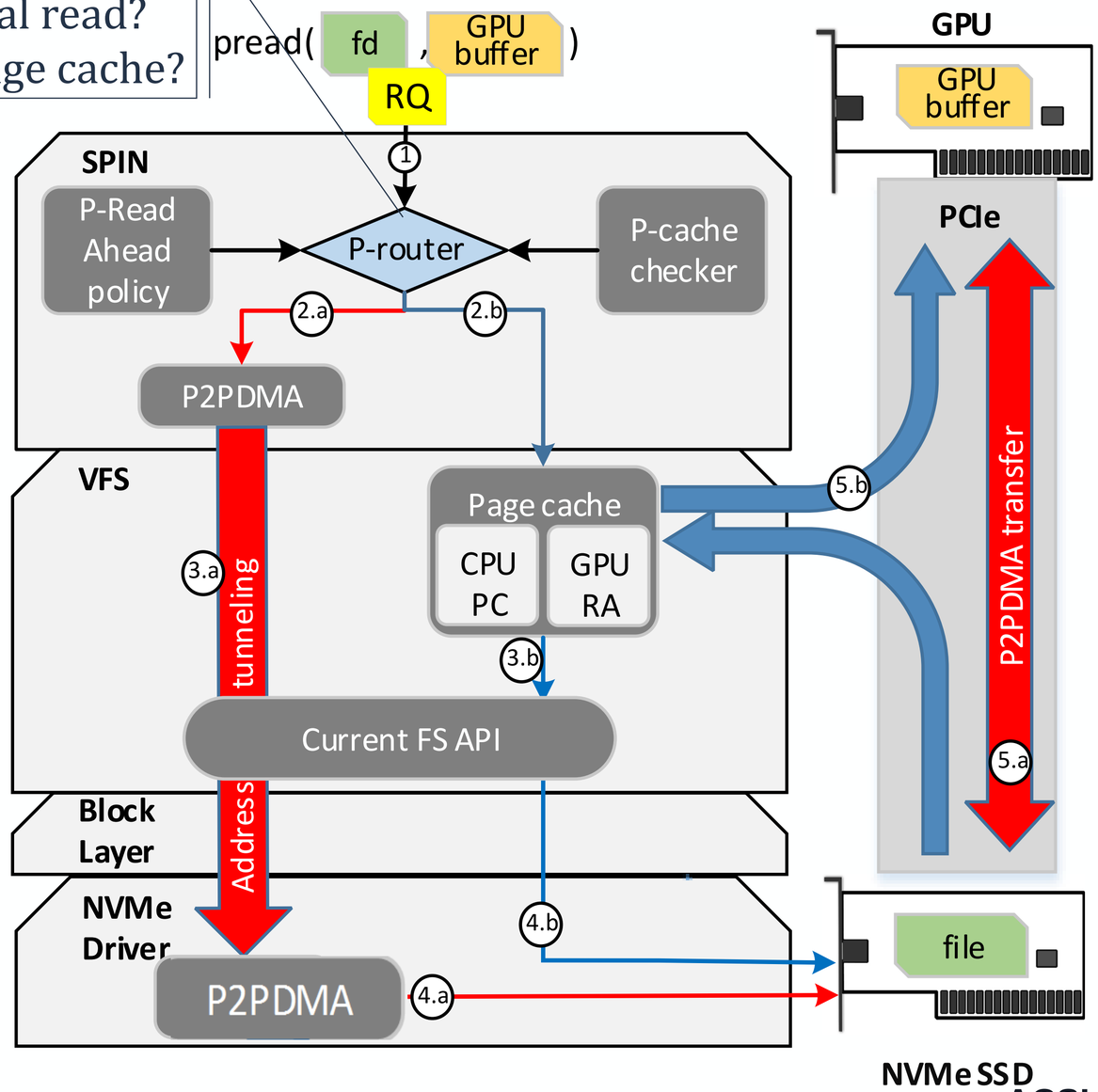
GPU Read Ahead

Activate read ahead mechanism when determined beneficial. Nested page cache within CPU memory for GPU Use

• You are here

SPIN: High level overview

From Compatible SSD?
Destined to GPU?
Part of a sequential read?
Data resides in page cache?

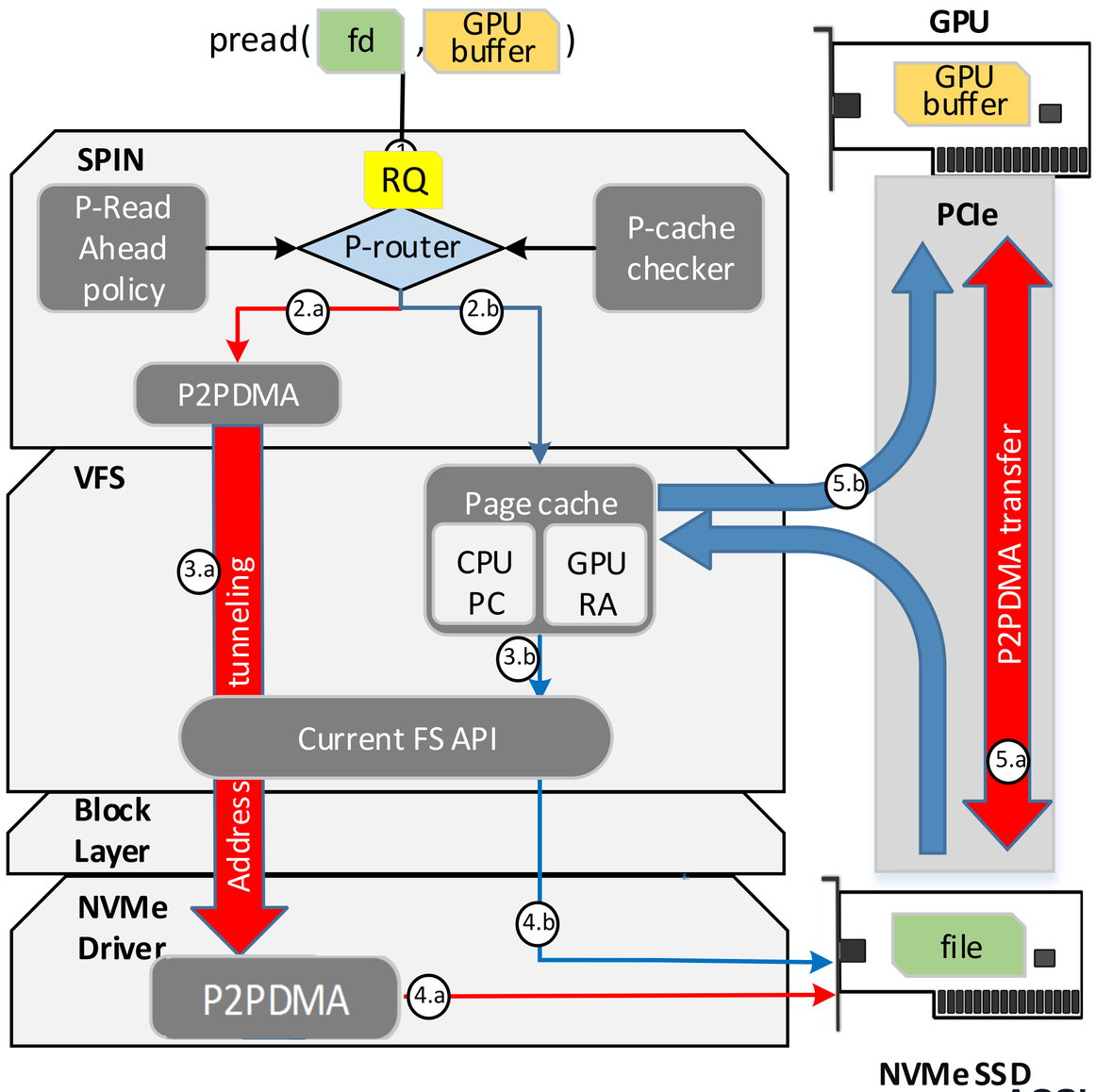


- You are here

SPIN: High level overview

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

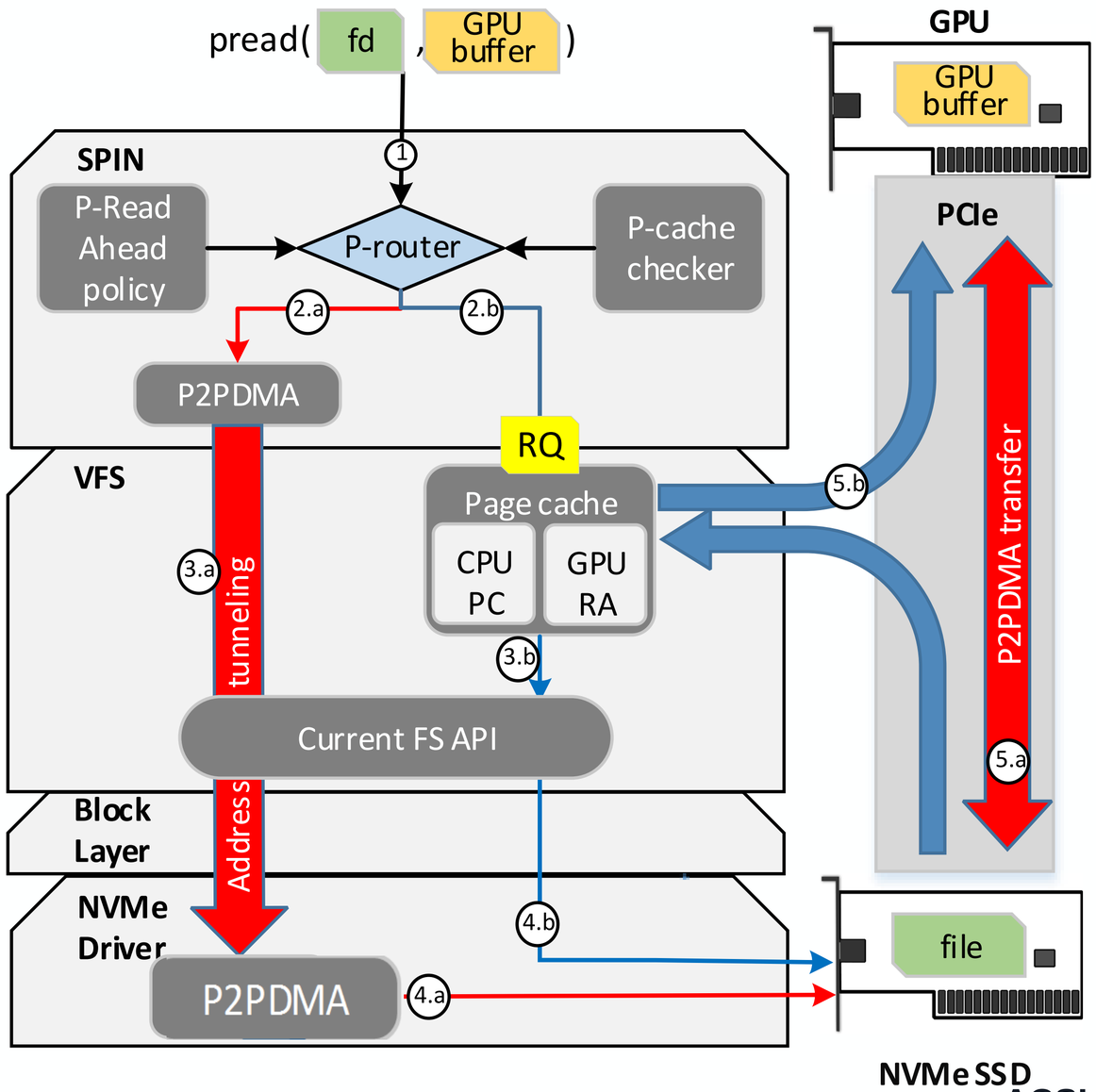
- You are here



SPIN: High level overview

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

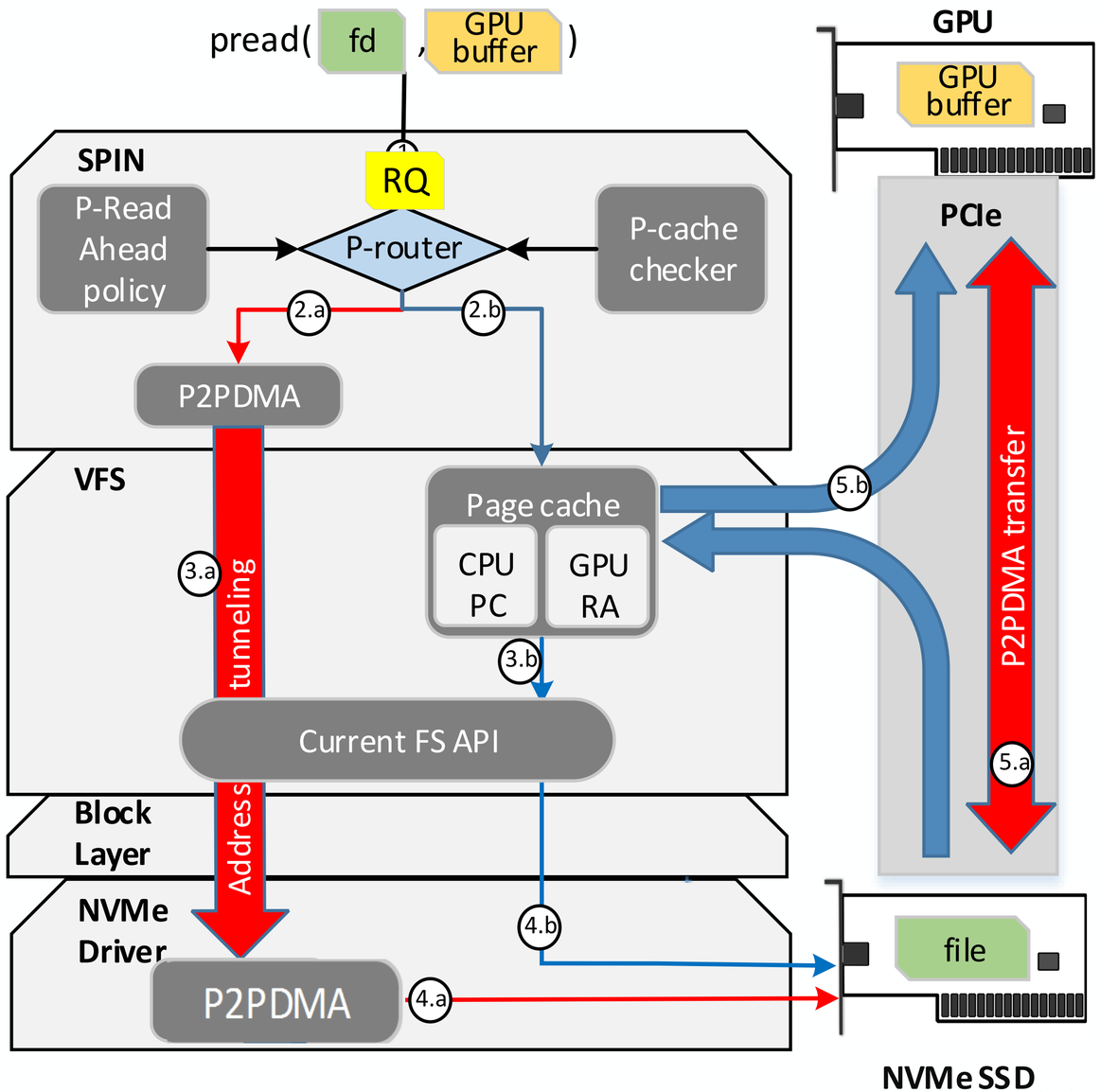
- You are here



SPIN: High level overview

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

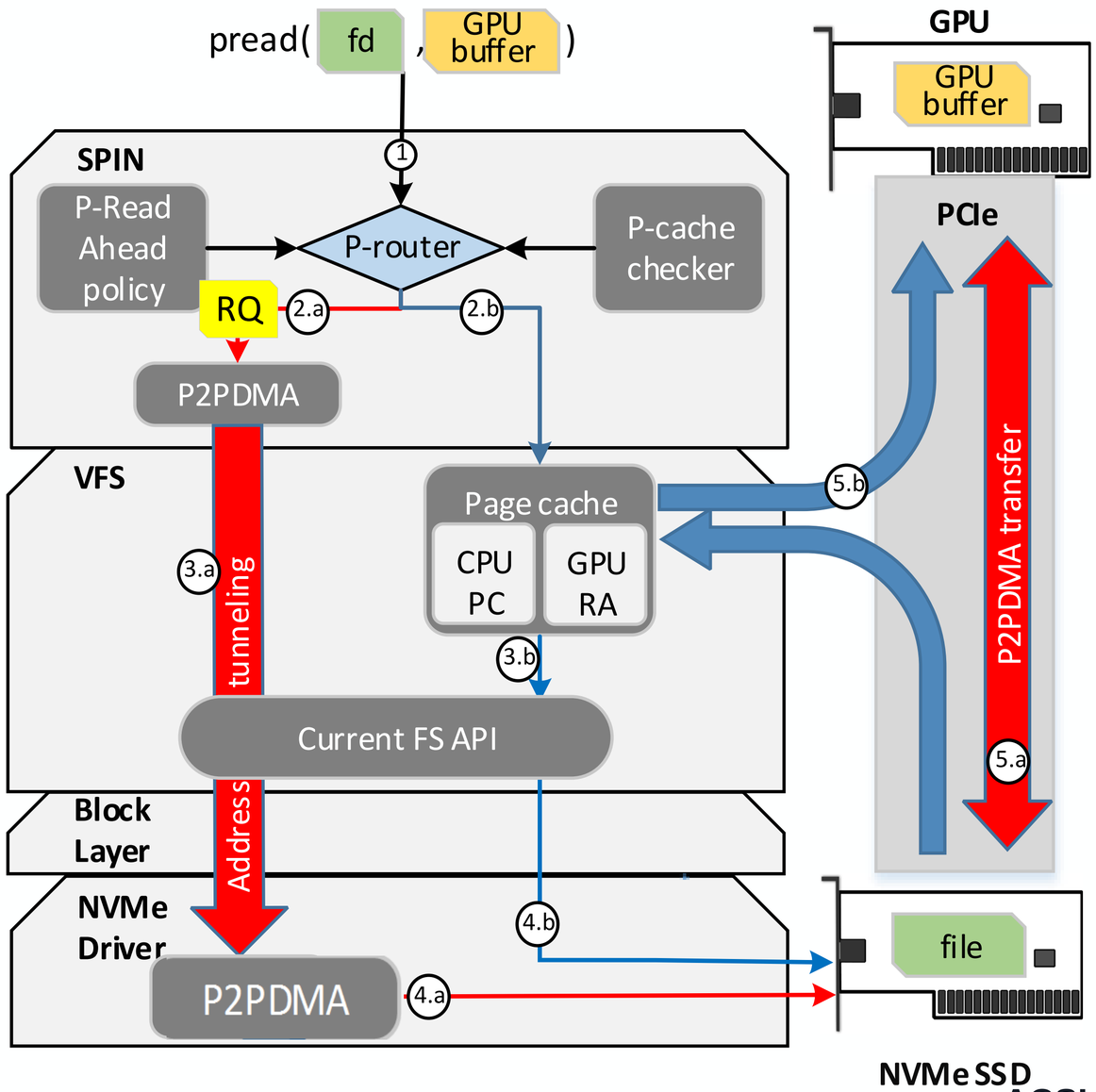
- You are here



SPIN: High level overview

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

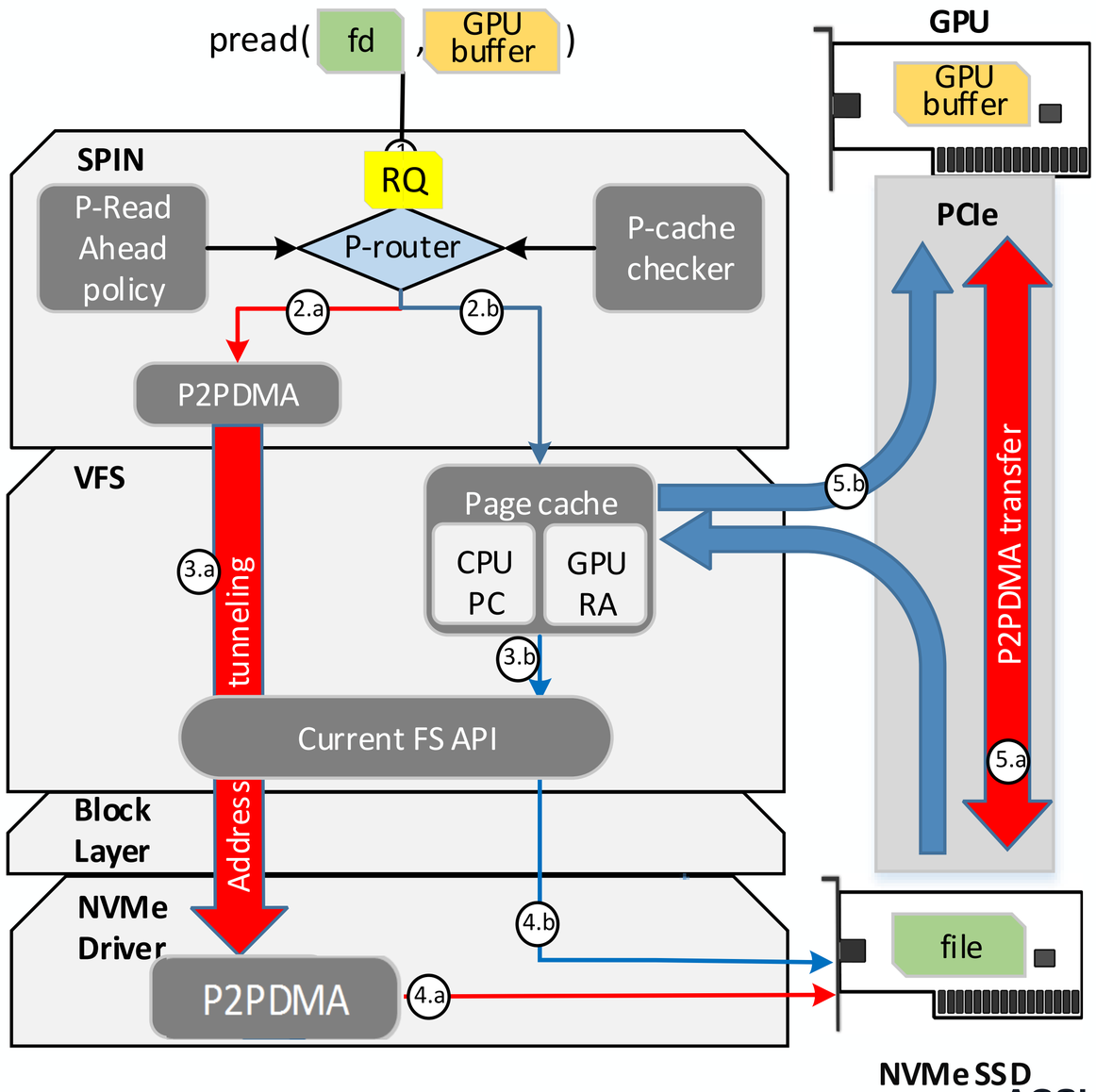
- You are here



SPIN: High level overview

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

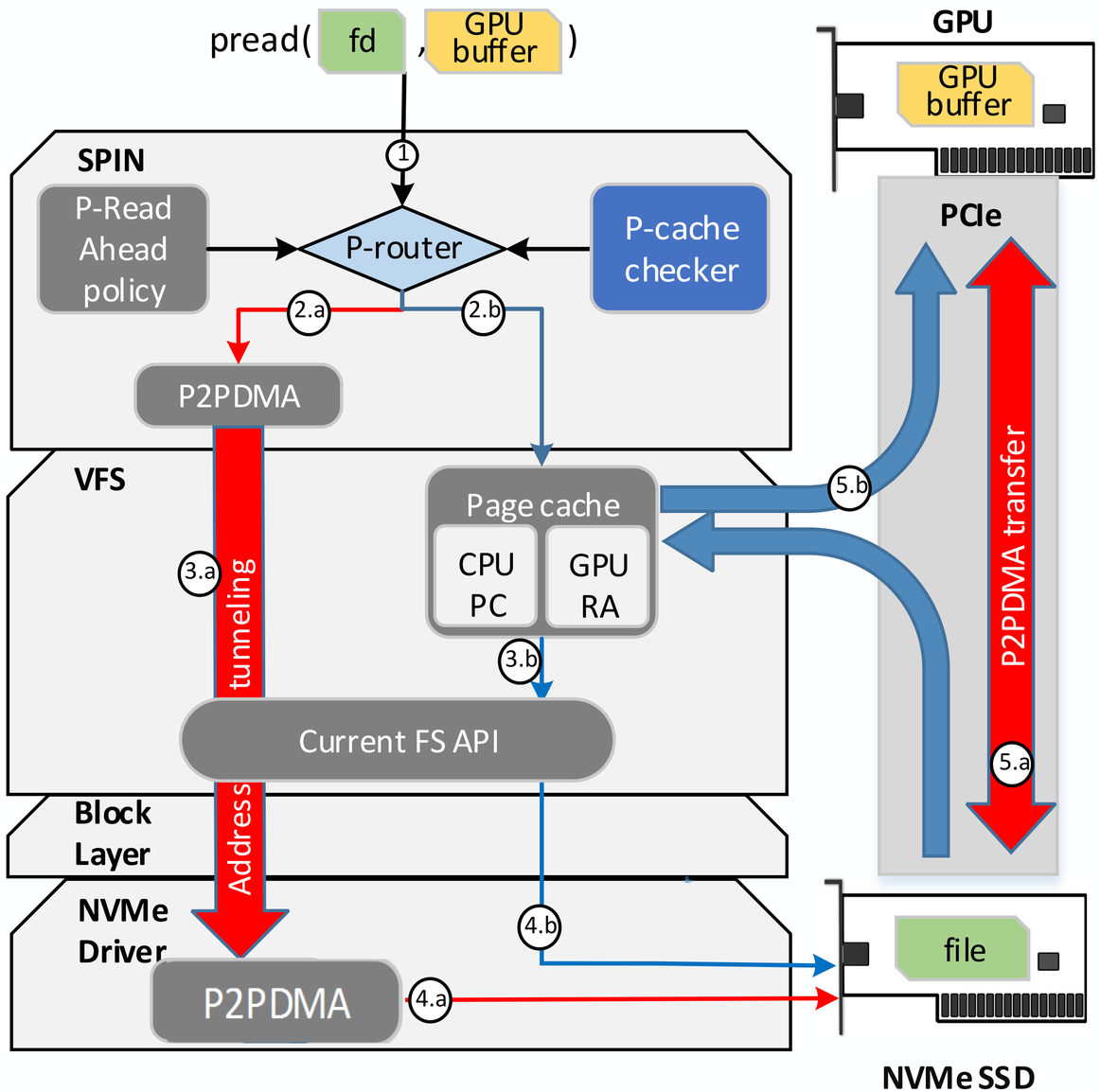
- You are here



SPIN: High level overview

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

- You are here

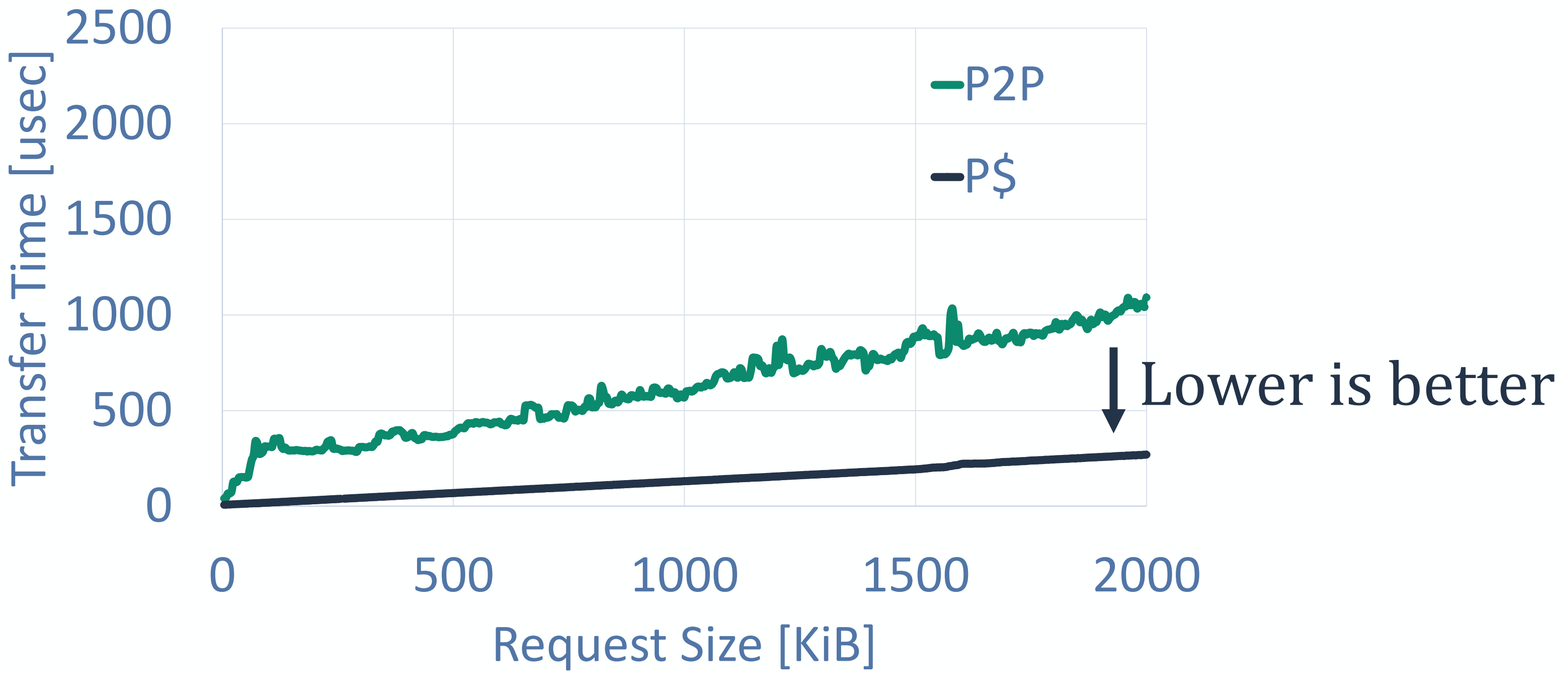


SPIN: Combine page cache and P2P

Sometimes the requested data resides in the P$

e.g due to previous usage of the data by CPU

Transfer time from P$ and SSD P2P vs request size



Transferring data from P$ is faster!

You are here

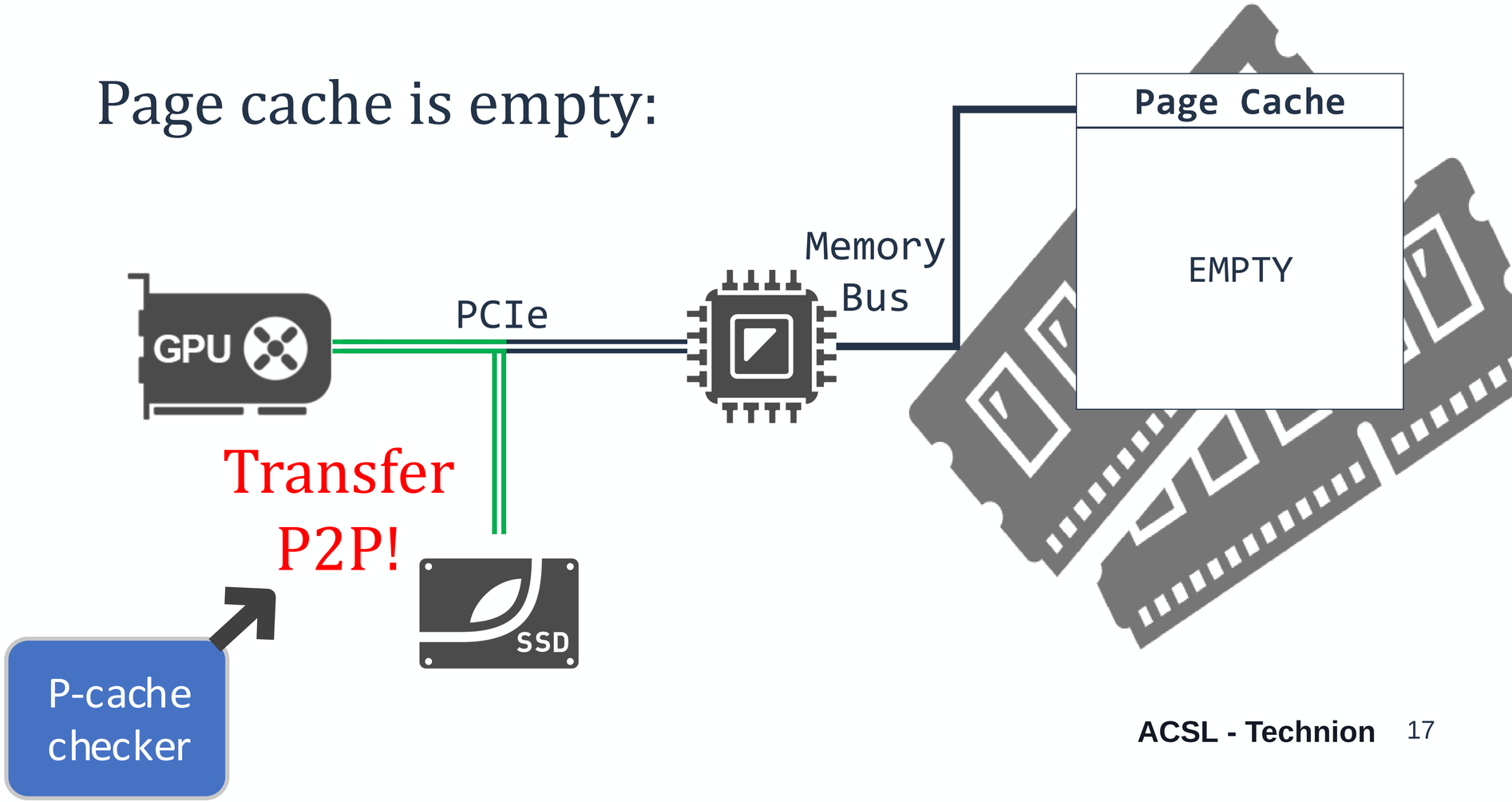
SPIN: Combine page cache and P2P

- You are here

```
pread64(fd,gpu_dest,5*4096,0);  //5 pages of 4KiB
```

Page #0	Page #1	Page #2	Page #3	Page #4

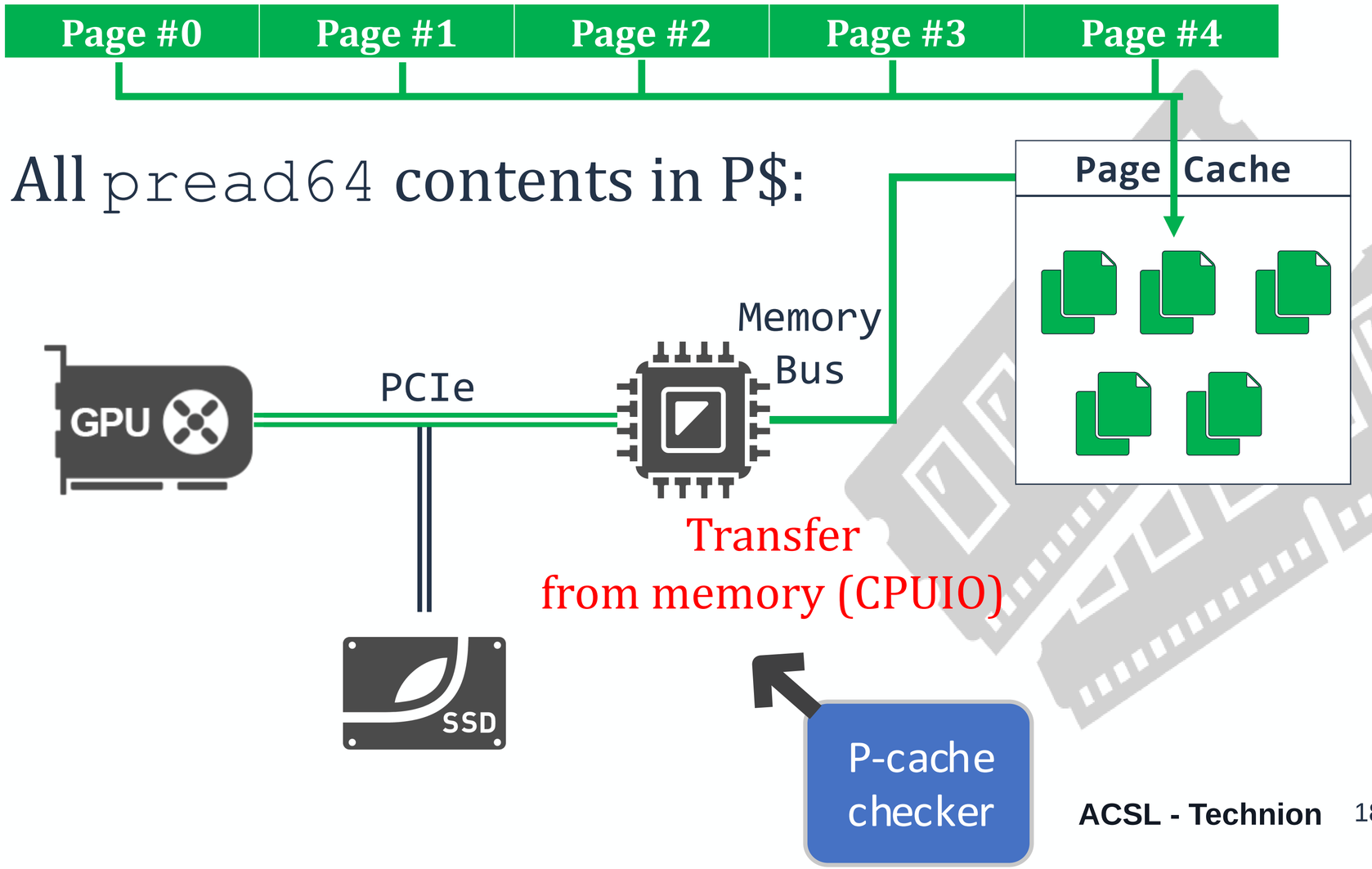
Page cache is empty:



SPIN: Combine page cache and P2P

- You are here

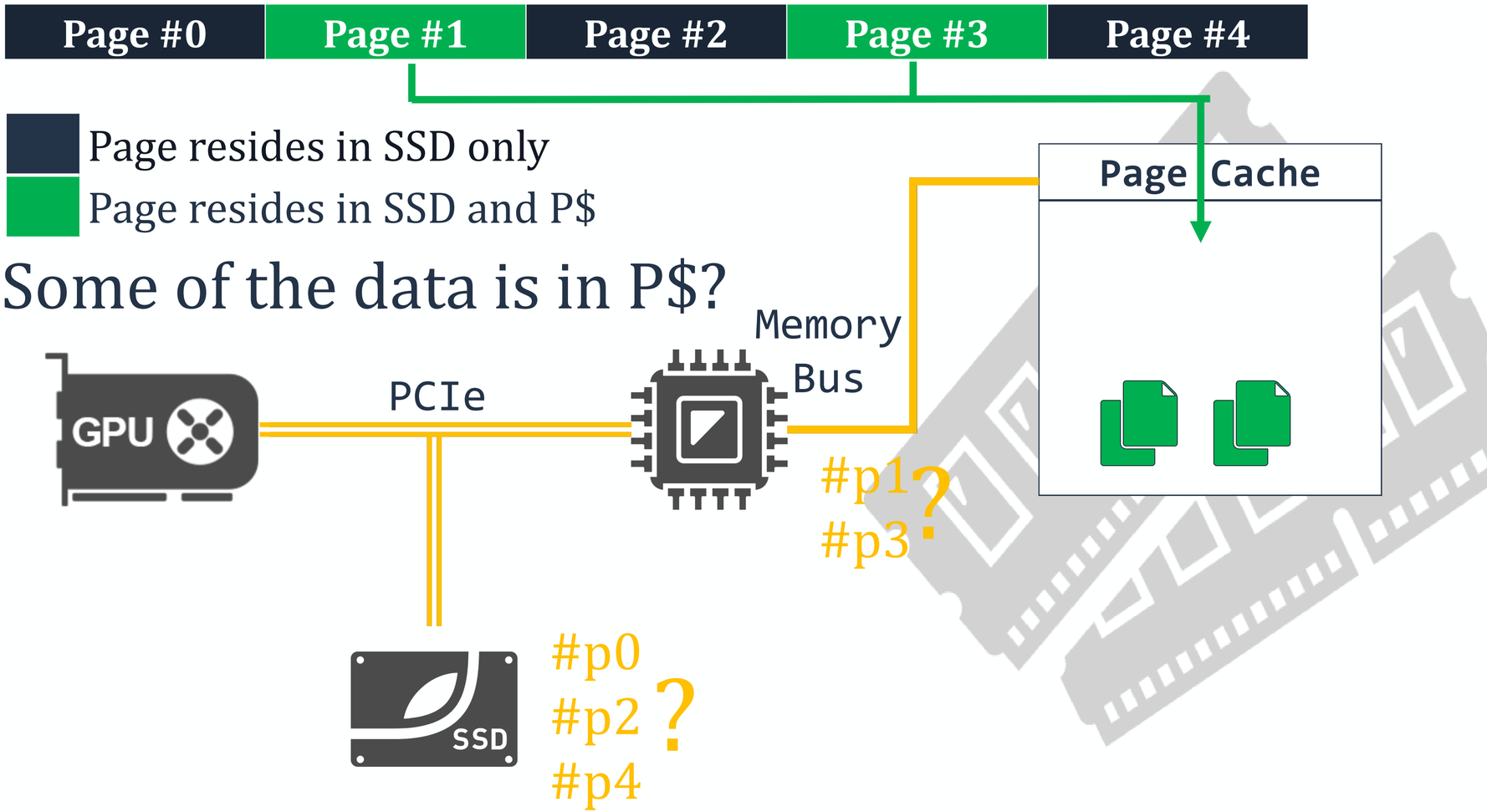
```
pread64(fd,gpu_destk,5*4096,0); //5 pages of 4KiB
```



SPIN: Combine page cache and P2P

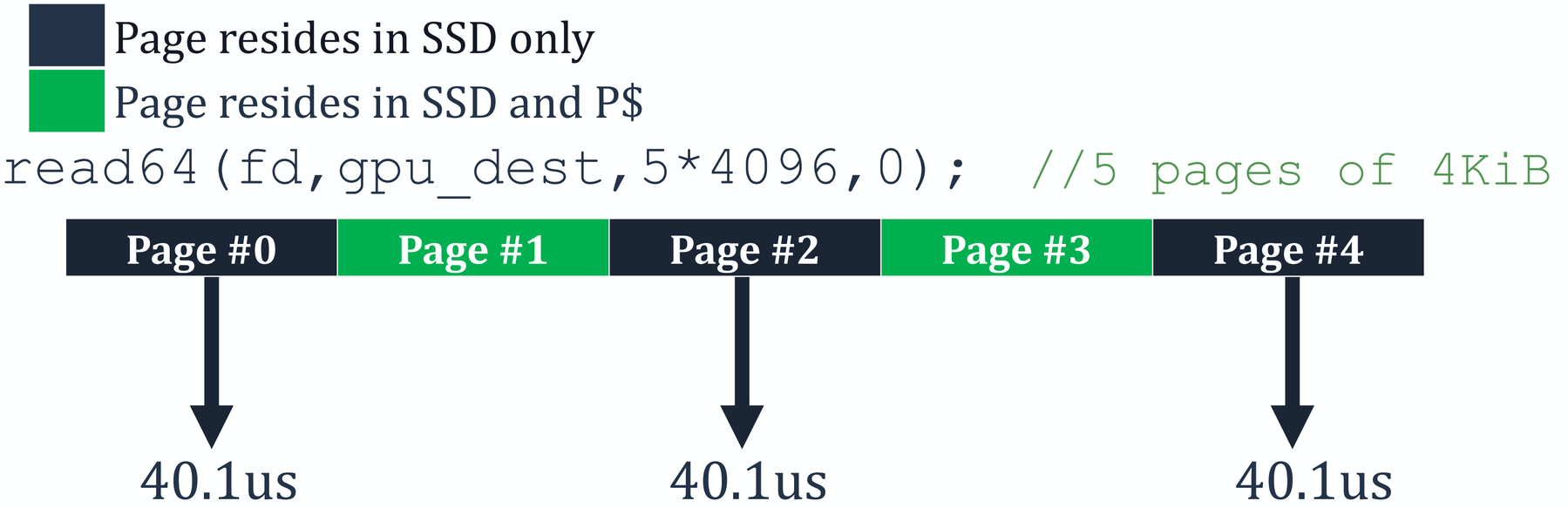
- You are here

```
pread64(fd,gpu_dest,5*4096,0);  //5 pages of 4KiB
```



Fine grained interleaving is a bad idea!

SPIN: Combine page cache and P2P



3 transfers of 4KiB via P2P: 120.3us

Single transfer of 20KiB via P2P: 74.3us

- You are here

SPIN: Combine page cache and P2P

Fine grained interleaving = poor performance!

SSDs:

- Short IO requests are less efficient (low parallelism)
- Invocation overhead per request

Optimization Problem:
Find the transfer schedule to minimize transfer time

● You are here

SPIN: Combine page cache and P2P

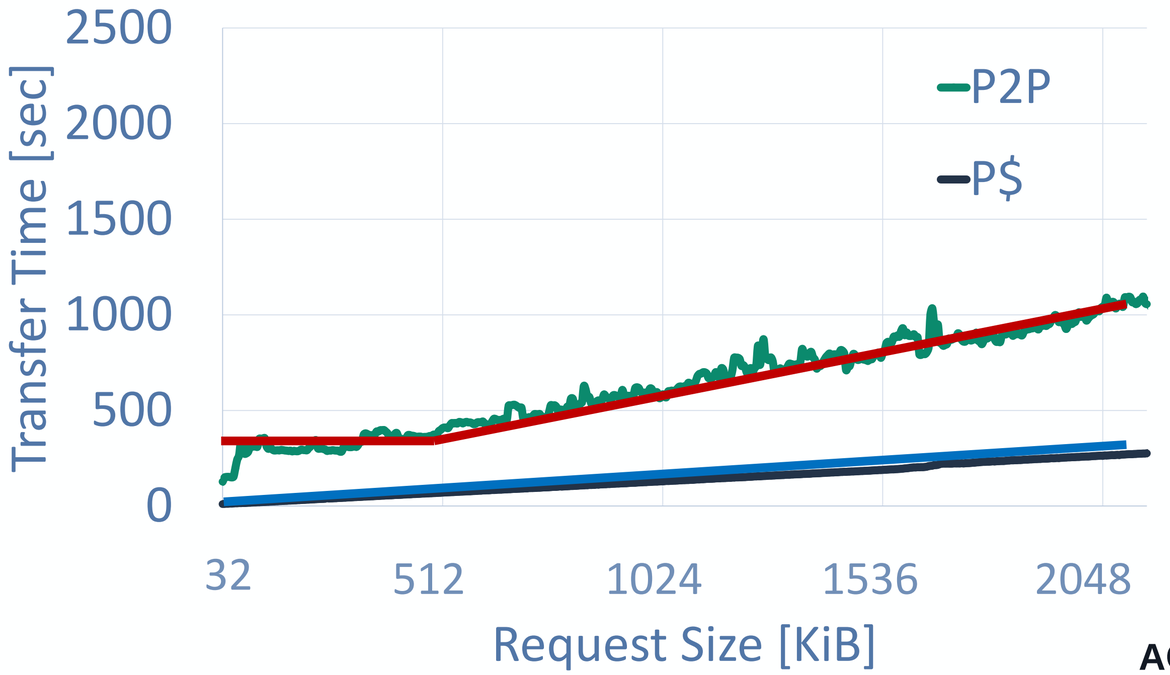
To solve the problem & get an optimal schedule we need:

$T_{p2p}(s)$ - P2P transfer time for a given request size

$T_{P\$}(s)$ - P$ transfer time for a given request size

We model the SSD and RAM performance characteristics:

- Assume P2P transfer time as piece-wise linear
- Assume RAM transfer time as linear



● You are here

SPIN: Combine page cache and P2P

Solution is polynomial in number of blocks
Costly to calculate for every transfer

We apply a greedy heuristic:
- Examine every 3 consecutive data chunks

Page resides in SSD only
Page resides in SSD and P$

Calculate:

$$T_{p2p}(|n| + |n+1| + |n+2|)$$

$$vs.$$

$$T_{p2p}(|n|) + T_{P\$}(|n+1|) + T_{p2p}(|n+2|)$$

You are here

SPIN: Combine page cache and P2P

Solution is polynomial in number of blocks
Costly to calculate for every transfer

We apply a greedy heuristic:

- Examine

Greedy Heuristic is only 1.6% slower than optimal scheduling

Calculate:

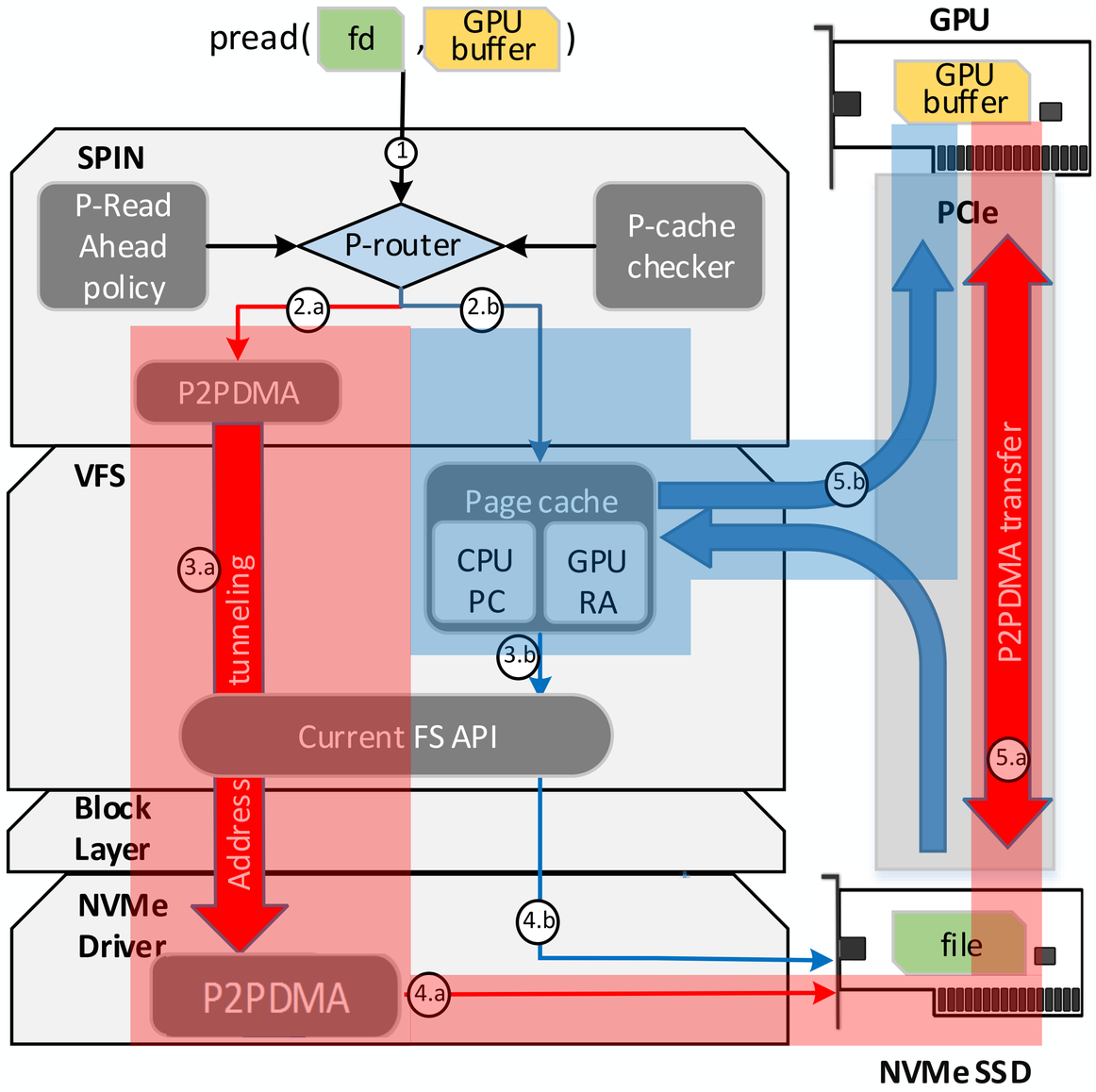
$$T_{p2p}(|n| + |n+1| + |n+2|)$$

$$vs.$$

$$T_{p2p}(|n|) + T_{P\$}(|n+1|) + T_{p2p}(|n+2|)$$

You are here

SPIN: Implementation: P2P & P$ Transfers



- P2P: Address tunneling mechanism
- P$: Memcpy from P$ to GPU mapped memory

- You are here

SPIN: Implementation & Evaluation

SPIN is implemented as a kernel module, patched NVME module & an LD_PRELOAD library

No kernel modifications are required

System Specs:

- Intel P3700 NVME SSD
- AMD Radeon R9 Fury & NVIDIA Tesla K40c
- Ubuntu + Linux kernel 3.19
- Intel Core i7-5930K (6 Phys Cores) & X99 Chipset
- 24GB DDR4 RAM

● You are here

SPIN: Implementation & Evaluation

We have evaluated the following:

Threaded IO (TIOtest) Benchmark (1-4 threads):

- Sequential reads (including software RAID)
- Random reads/writes
- Effects of P$ residency on read throughput
- Effects CPU & I/O stress on read throughput

Application Benchmarks

- Aerial imagery rendering
- GPU accelerated log server
- Image collage utilizing GPUFS

• You are here

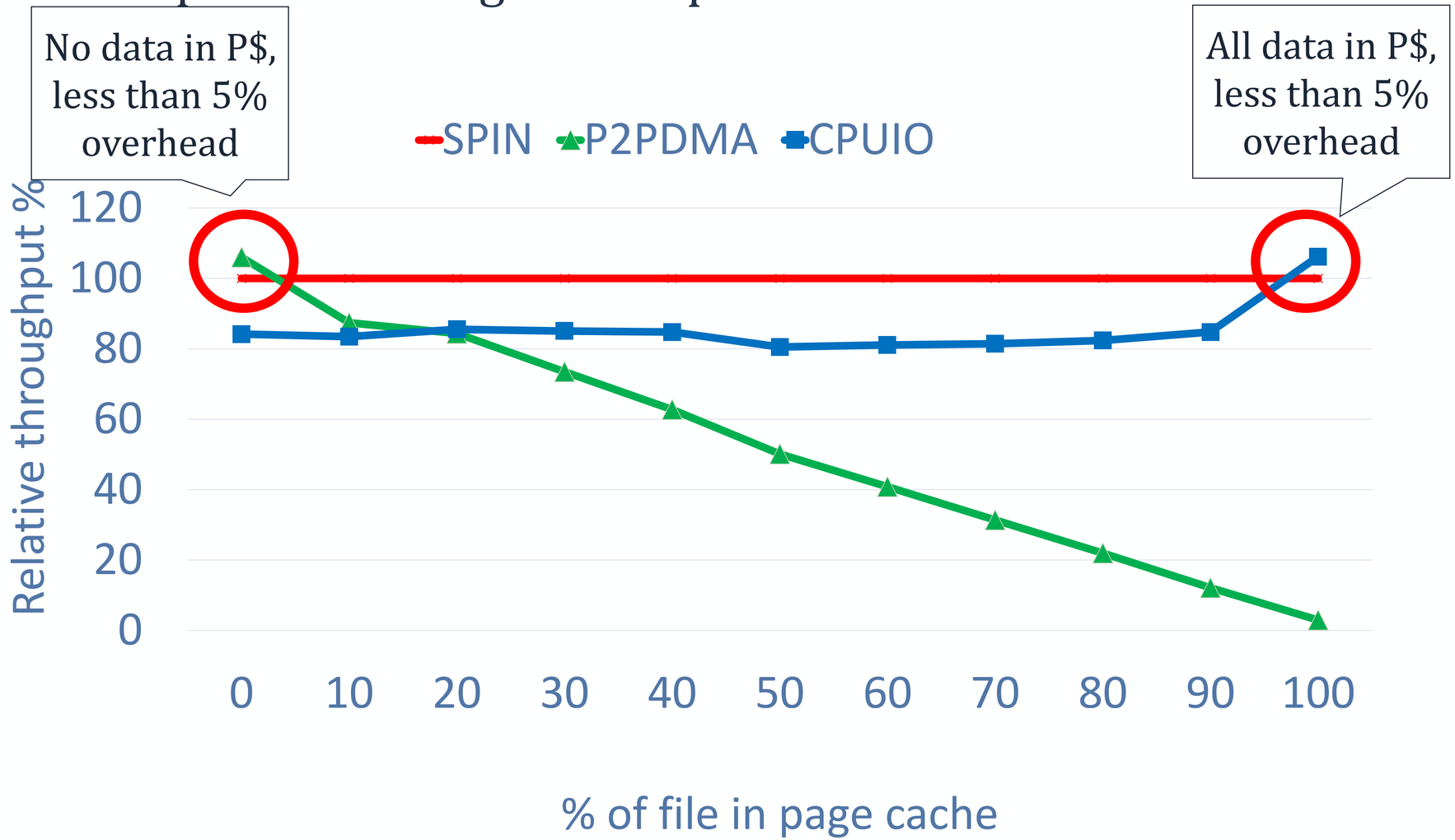
SPIN: Implementation & Evaluation

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

- You are here

Effect of P$ on Read Throughput

Potential performance gains for producer-consumer workloads



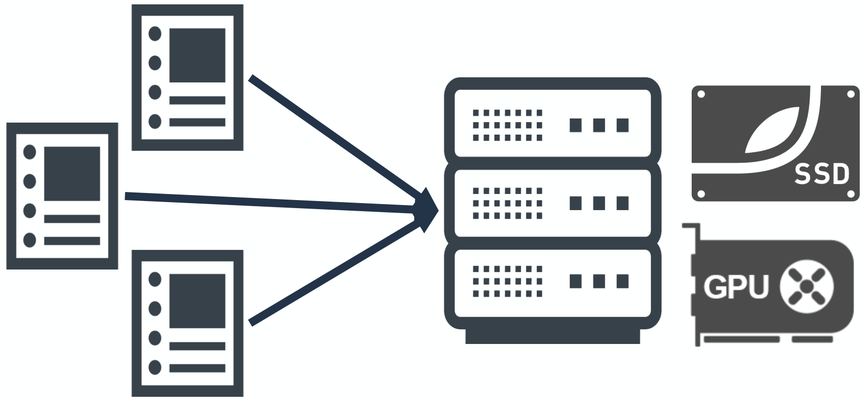
*512B reads

SPIN: Implementation & Evaluation

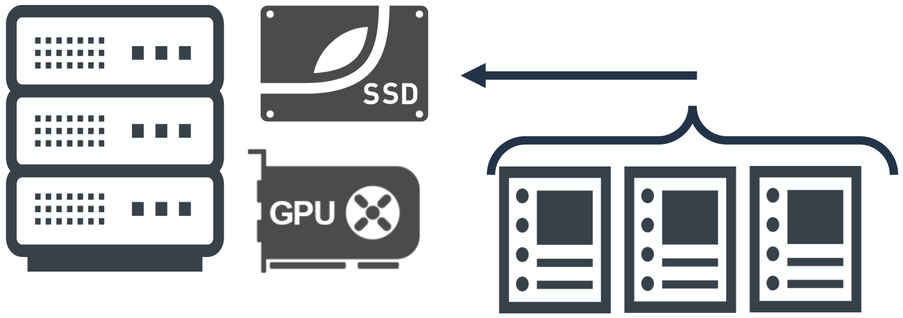
You are here

GPU Accelerated Log Server

- Store a log into SSD
- Analyze log using GPU acceleration for string matching
- Similar to fail2ban



Real time configuration:
- Log arrives to server
- Server stores logs in SSD
- GPU analyzes logs by reading file



Offline configuration:
- Log is already in SSD
- GPU analyzes logs by reading file

SPIN: Implementation & Evaluation

You are here

GPU Accelerated Log Server

- Store a log into SSD
- Analyze log using GPU acceleration for string matching
- Similar to fail2ban

We want our application to work efficiently in any configuration

Real time configuration:
- Log arrives to server
logs in SSD
logs by reading file

ation:
- Log is already in SSD
- GPU analyzes logs by reading file

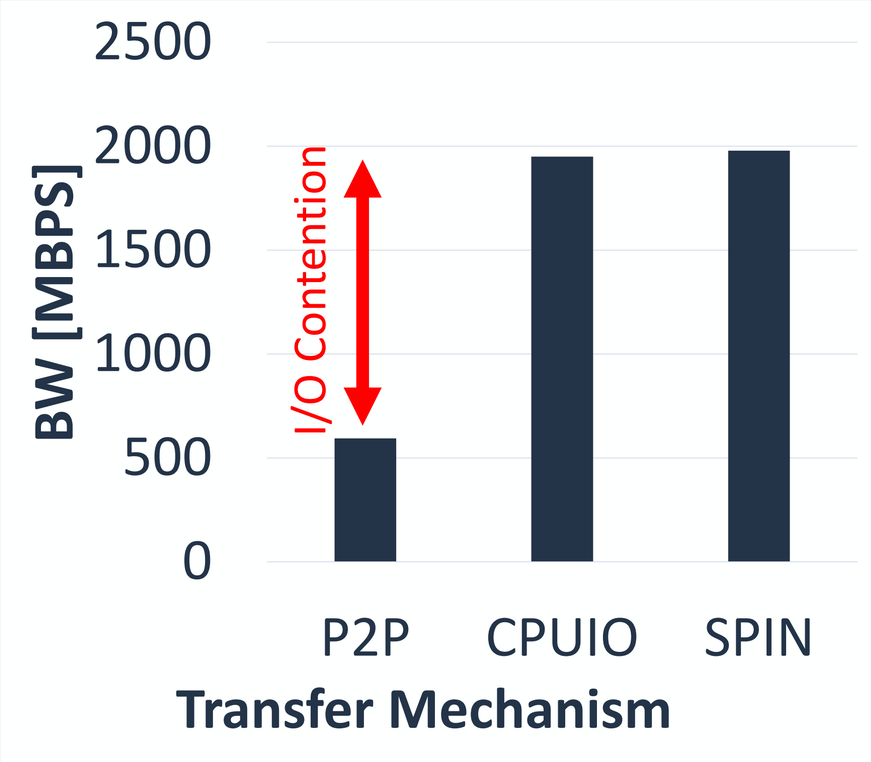
SPIN: Implementation & Evaluation

- Summary
- Background
- Observations
- Objective
- SPIN
- Conclusion

- You are here

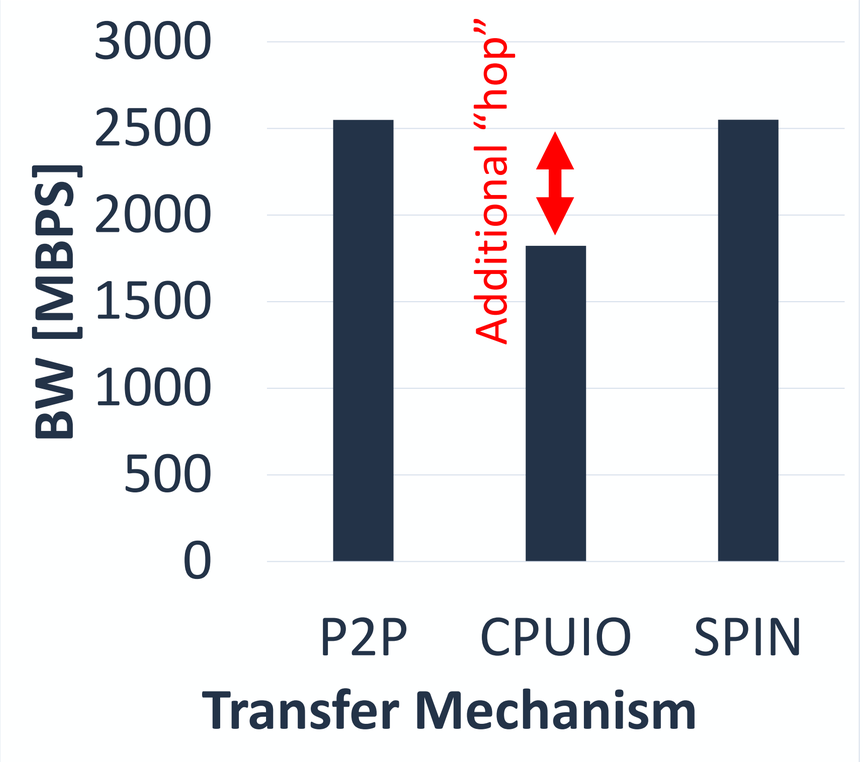
GPU Accelerated Log Server

Real Time configuration



Data resides in p$ and SSD
SPIN reads data from P$

Offline configuration



Data resides in SSD only
SPIN utilizes P2P

SPIN: Conclusion

- SPIN seamlessly integrates P2P as a first class citizen into the file I/O stack
- SPIN utilizes several mechanisms to speed up data transfers **transparently**
- With SPIN, the same code performs well under all setups

Thank you!


shaiberg1@tx.technion.ac.il


github.com/acsl-technion/spin

You are here