

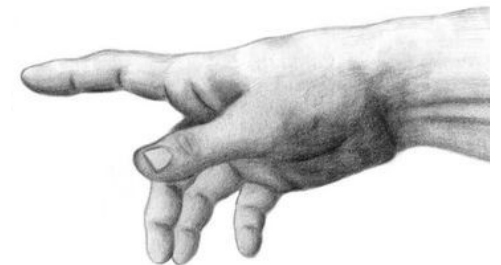
# ActivePointers: The case for software address translation on GPUs

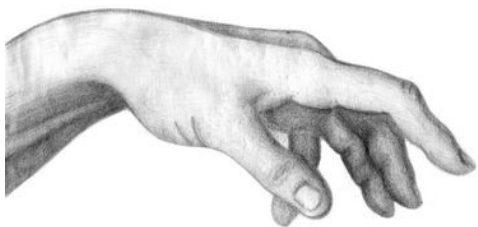
Sagi Shahar  
Shai Bergman  
**Mark Silberstein**

Technion – Israel Institute of Technology

# History of the world in 7 slides

# CPU





# Application

# CPU



# Application

## OS

## CPU





# CPU

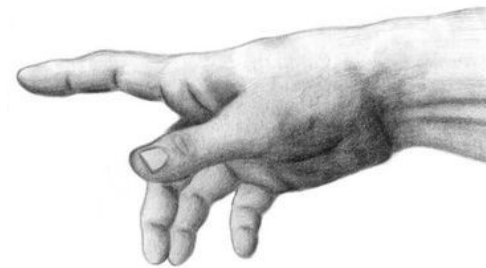


# Application

## OS

## CPU

## GPU





# Application

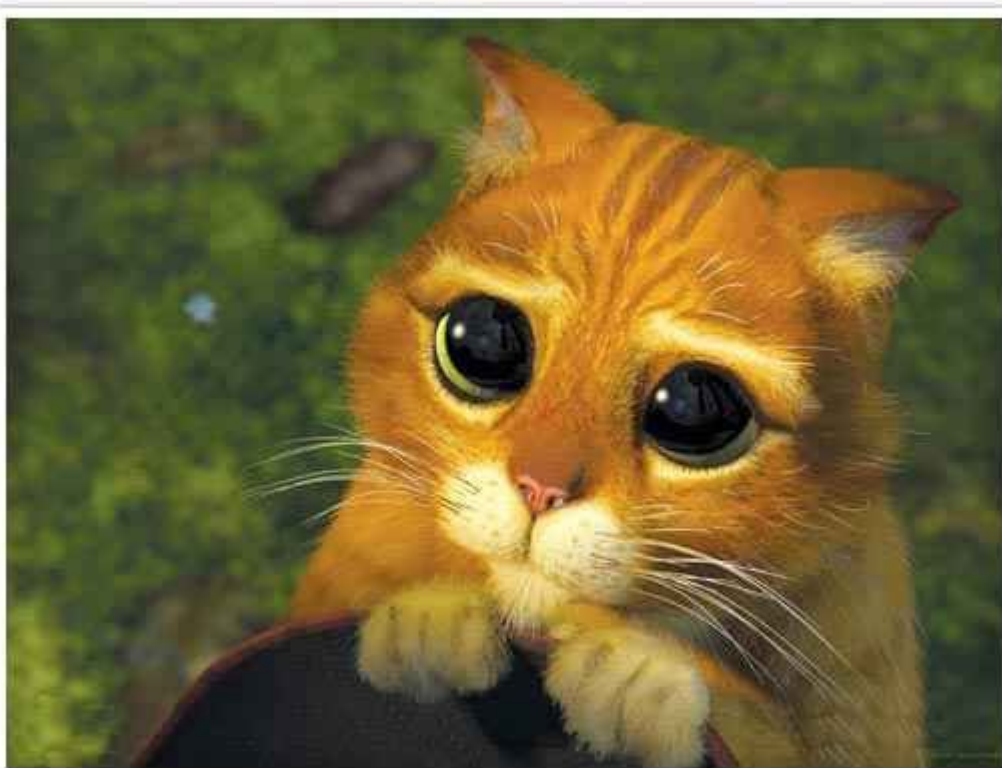
OS

?????

CPU

GPU





**OS**

**?????**

**CPU**

**GPU**



# Application



## OS

## ?????



## CPU

## GPU

# This is our world



## Application



### OS

### ?????



### CPU

### GPU



**Application**

**OS services**  
(GPUfs, GPUnet, GPUrdma)

**GPU**

# Agenda

- Motivation
- Background – GPU file system support
- ActivePointers: software translation layer for `mmap`
- Evaluation
- Conclusions

# Motivation: Processing large datasets on GPUs

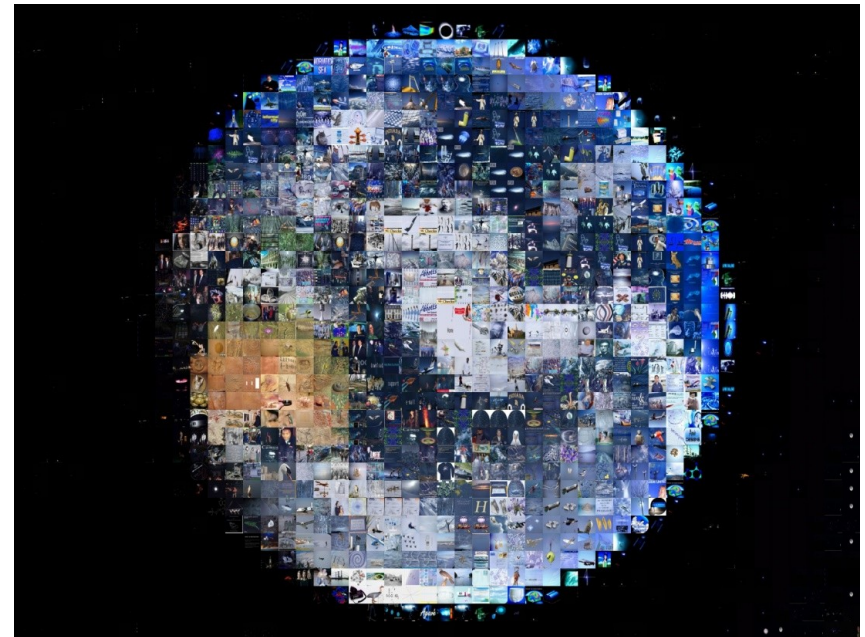
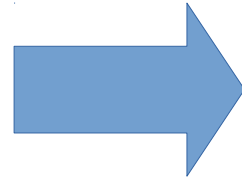


Image collage: for every block  
find “the best match” image in a DB

# Image collage: data driven access

For each input image block

- Compute **indexes** into DB file
- Read candidates from DB file
- Brute-force search to choose the best

Common pattern for  
DBs, text and image search

# Traditional programming model

## *GPU as a co-processor*

For each input image block

GPU

- Compute **indexes** into DB file

- Read candidates from DB file

CPU

- Brute-force search to choose the best

GPU



# File-system access from GPU

## *Peer-processor* model

For each input image block

- Compute **indexes** into DB file
- Read candidates from DB file
- Brute-force search to choose the best

GPU

**Easier programming**  
**Higher performance**

# File-system access from GPU

## *Peer-processor* model

For each input image block

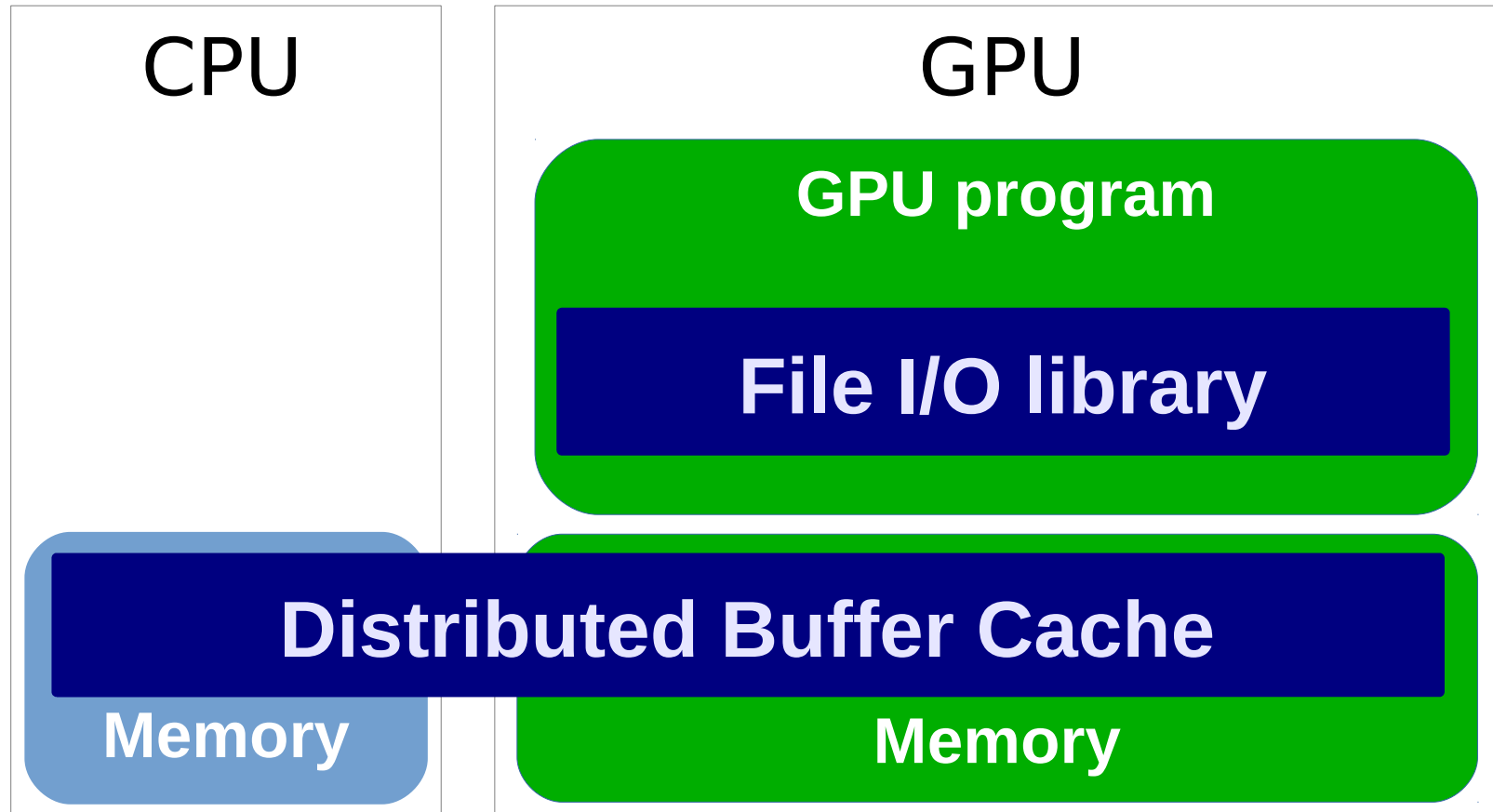
- Compute **indexes** into DB file
- Read candidates from DB file
- Brute-force search to choose the best

**GPUfs**  
[ASPLOS13]

**GPU**

**Easier programming**  
**Higher performance**

# GPUfs: FS API + Distributed Buffer Cache



Silberstein et al., *ASPLOS* 2013, *TOCS* 2014, *CACM* 2014

# Wanted: `mmap` ( )

- Typical usage: `mmap` the whole file into application address space
- File access through a regular pointer

## Benefits

Simplicity

On-demand data transfer

Performance

# Wanted: `mmap` ( )

- Typical usage: `mmap` the whole file into application address space
- File

**Challenge: lack of GPU hardware support for VM management**

Simplicity

On-demand data transfer

Performance

# Reminder: mmap on CPU

- allocate virtual memory region
  - no physical memory allocated first
- on first access - page fault
  - allocate page in a buffer cache
  - read from file
  - map the page into process's virtual address space

# Reminder: `mmap` on CPU

- allocate virtual memory region
  - no physical memory allocated first
- on first access - page fault
  - allocate page in a buffer cache
  - read from file
  - map the page into process's virtual address space

Can we implement `mmap` on GPU?

# mmap on GPU?

- allocate virtual memory region
  - no physical memory allocated first
- on first access - page fault
  - allocate page in a buffer cache
  - read from file
  - map the page into virtual memory

Recent  
GPUs

GPUfs



# mmap on GPU?

- allocate virtual memory region
  - no physical memory allocated first
- on first access - page fault
  - allocate page in a buffer cache
  - read from file
  - map the page into virtual memory

No GPU user control

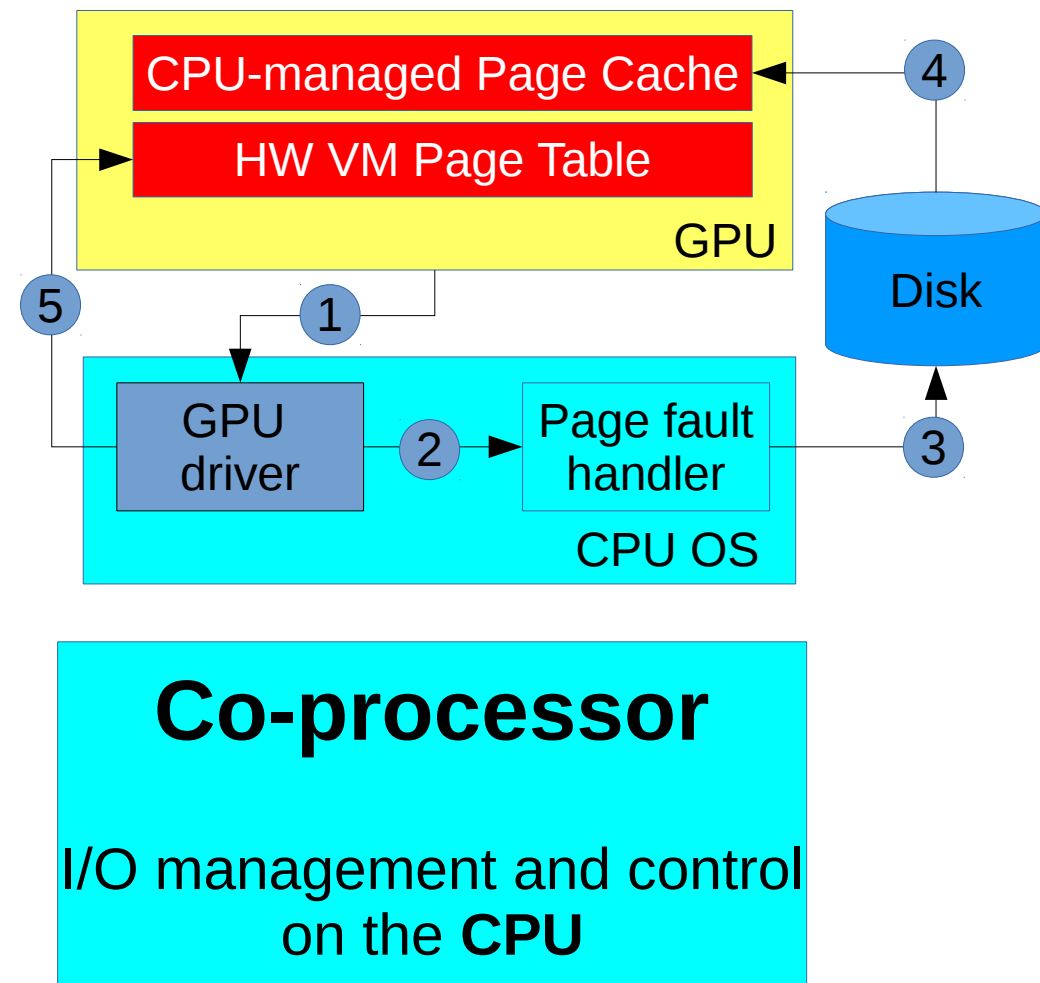
Recent GPUs

GPUfs

No GPU user control

# Today:

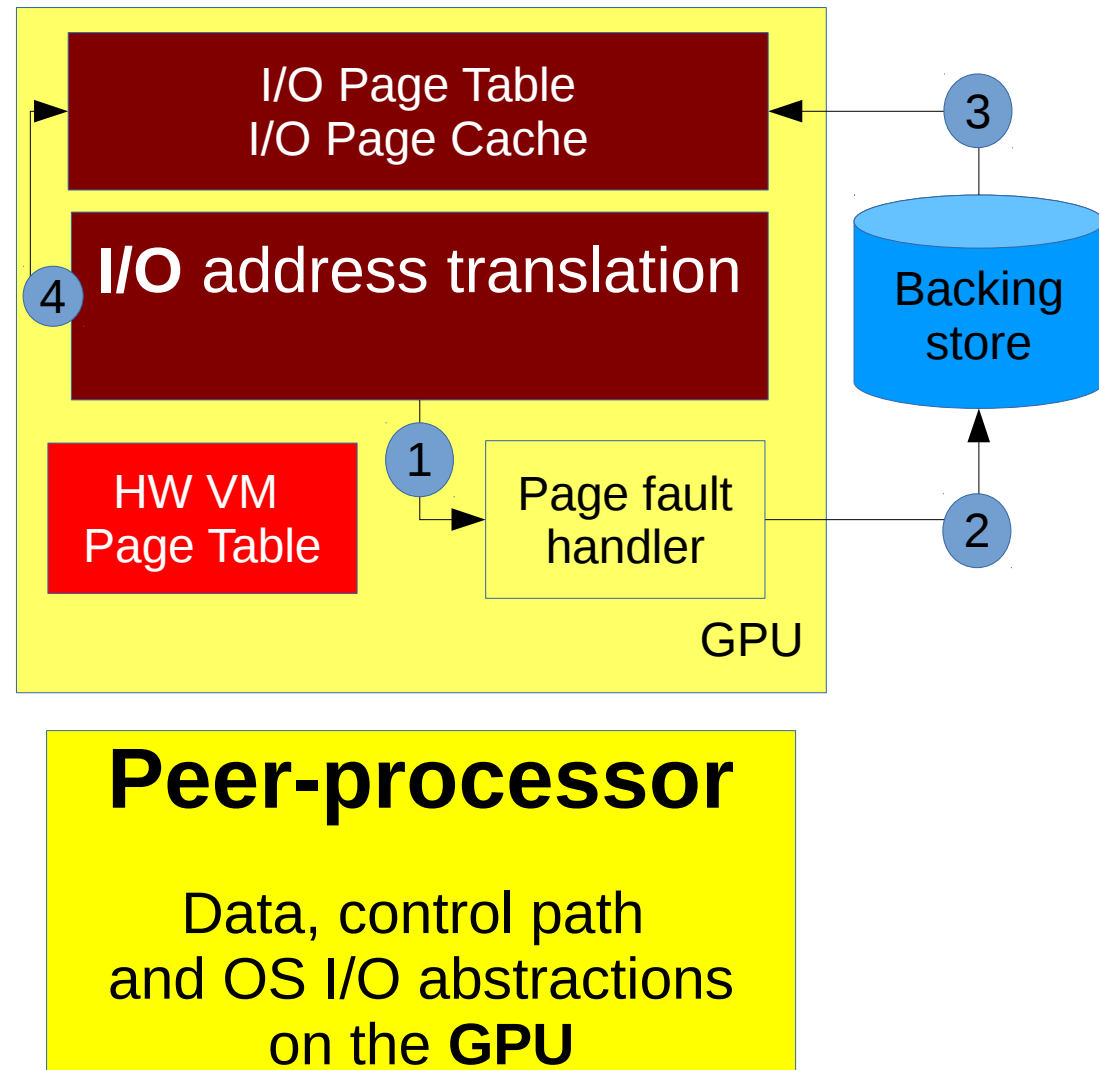
## CPU-centric VM management



- Pros:
  - compatible with the CPU OS
  - no extra GPU code
- Cons:
  - CPU in every page fault
  - CPU-GPU coordination for page cache management
  - No GPU page fault handlers

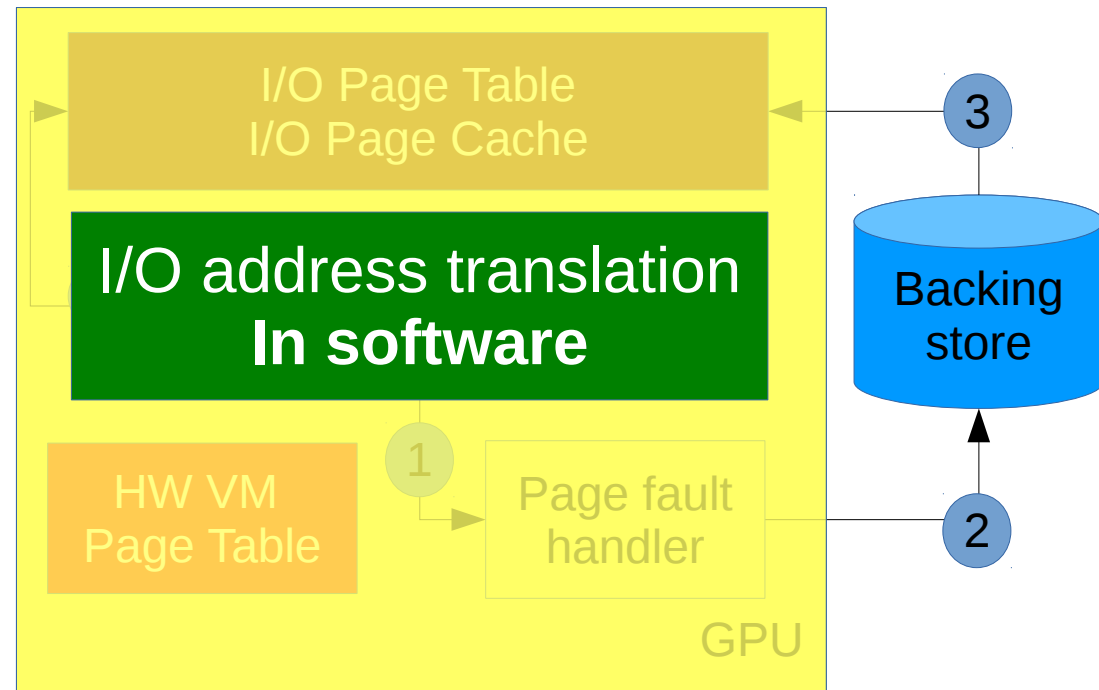
# This work: GPU-centric VM management for I/O operations

- Layered on top of regular VM
- Pros:
  - no CPU involvement on page faults
  - GPU handles page faults
  - High throughput and low page fault handling latency



# Software Address Translation Layer

- Pros:
  - No costly hardware TLB updates
  - Extensible
  - User-level access
  - Complementary to Hardware VM



Fully compatible with commodity GPUs

# Agenda

- Software address translation
- Evaluation
- Conclusions

# Desired behavior

- **GPU code**

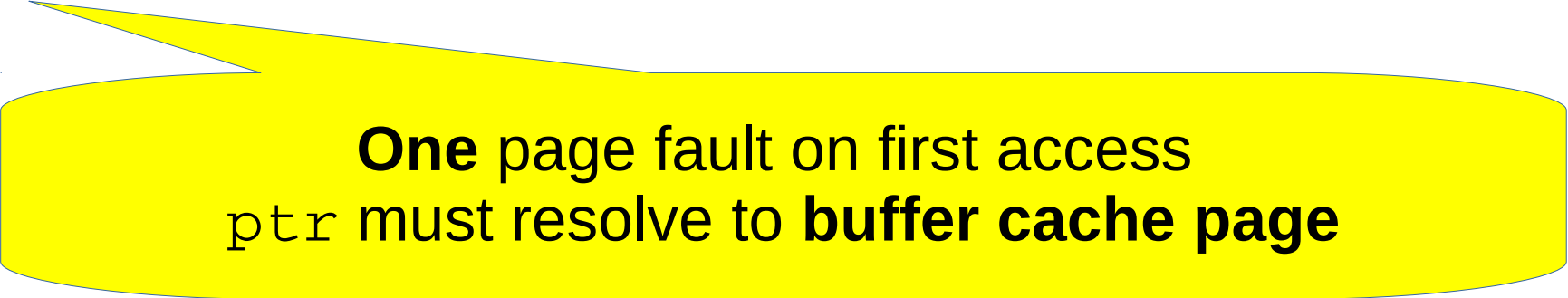
```
void* ptr=gmmmap(fd,offset,size)
```

```
for(int i=0;i<size;i++,ptr++)
```

```
{
```

```
    ptr[threadIdx.x]=25;
```

```
}
```



**One page fault on first access**  
**ptr must resolve to **buffer cache page****

# Desired behavior

- **GPU code**

```
void* ptr=gmmmap(fd,offset,size)

for(int i=0;i<size;i++,ptr++)
{
    ptr[threadIdx.x]=25;
}
```

**Requires page table lookup  
on every access!**

# Software TLB - inefficient

- One TLB per core (Threadblock) in shared memory

**Extra memory accesses for each read**  
**Contention on TLB updates**  
**How to handle TLB invalidations?**

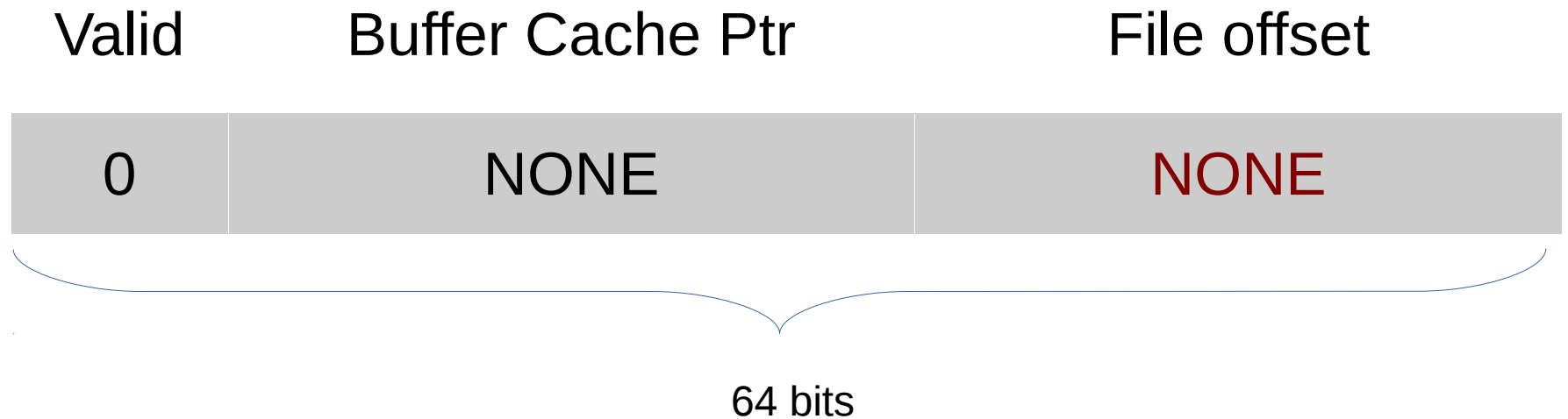


# ActivePointers

## Main design principles

- **Minimize page table lookups**
  - translation **is cached in hardware registers**
- **Pages locked** in the buffer cache as long as they are in use
  - keep reference count for each page

# ActivePtr structure



# Example

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	0	NONE	NONE

{

ActivePtr ptr;

ptr = gmmmap(fd, 4096, size);

float x = \*ptr;

ptr++;

float y = \*ptr;

ptr+=4096;


}

# Example

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	0	NONE	4096

{

```
ActivePtr ptr;
```



```
ptr = mmap(fd, 4096, size);
```

```
float x = *ptr;
```

```
ptr++;
```

```
float y = *ptr;
```

```
ptr+=4096;
```

```
}
```

# Example

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	0	NONE	4096

{

```
ActivePtr ptr;
```

```
ptr = gmmmap(fd, 4096, size);
```

```
float x = *ptr;
```

```
ptr++;
```

```
float y = *ptr;
```

```
ptr+=4096;
```

```
}
```

**Pagefault**

# Example

Page Table Entry

0xFFFC0000	→	RefCount	File	←
		0	4096	

Page  
fault  
handler

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	0	NONE	4096

{

```
ActivePtr ptr;
```

```
ptr = gmmmap(fd, 4096, size);
```

```
float x = *ptr;
```

```
ptr++;
```

```
float y = *ptr;
```

```
ptr+=4096;
```

```
}
```

Pagefault

# Example

Lock page

Page Table Entry

0xFFFC0000

RefCount	File
1	4096

Page  
fault  
handler

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	1	0xFFFC0000	4096

{

```
ActivePtr ptr;
```

```
ptr = gmmmap(fd, 4096, size);
```

```
float x = *ptr;
```

```
ptr++;
```

```
float y = *ptr;
```

```
ptr+=4096;
```

```
}
```

Pagefault

# Example

## Page Table Entry

0xFFFFC0000 →	RefCount	File
	1	4096

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	1	0xFFFFC0004	4100

{

```
ActivePtr ptr;
```

```
ptr = gmmmap(fd, 4096, size);
```

```
float x = *ptr;
```

```
ptr++;
```

```
float y = *ptr;
```

```
ptr+=4096;
```

```
}
```



# Example

## Page Table Entry

	RefCount	File
0xFFFFC0000 →	1	4096

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	1	0xFFFFC0004	4100

```
{  
    ActivePtr ptr;
```

```
    ptr = gmmmap(fd, 4096, size);
```

```
    float x = *ptr;
```

```
    ptr++;
```

```
    float y = *ptr;
```

```
    ptr+=4096;
```

```
}
```

Fault-free  
Lookup-free  
access

# Example

Unlock page

Page Table Entry

0xFFFC0000 →

RefCount	File
0	4096

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	0	NONE	8196

{

```
ActivePtr ptr;
```

```
ptr = gmmmap(fd, 4096, size);
```

```
float x = *ptr;
```

```
ptr++;
```

```
float y = *ptr;
```

```
ptr+=4096;
```

```
}
```

Crossing  
page boundary

# Example

Unlock page

Page Table Entry

0xFFFC0000

RefCount	File
0	4096

	Valid	Buffer Cache Ptr	File offset
<i>ptr</i>	0	NONE	4100

{

```
ActivePtr ptr;
```

```
ptr = gmmmap(fd, 4096, size);
```

```
float x = *ptr;
```

```
ptr++;
```

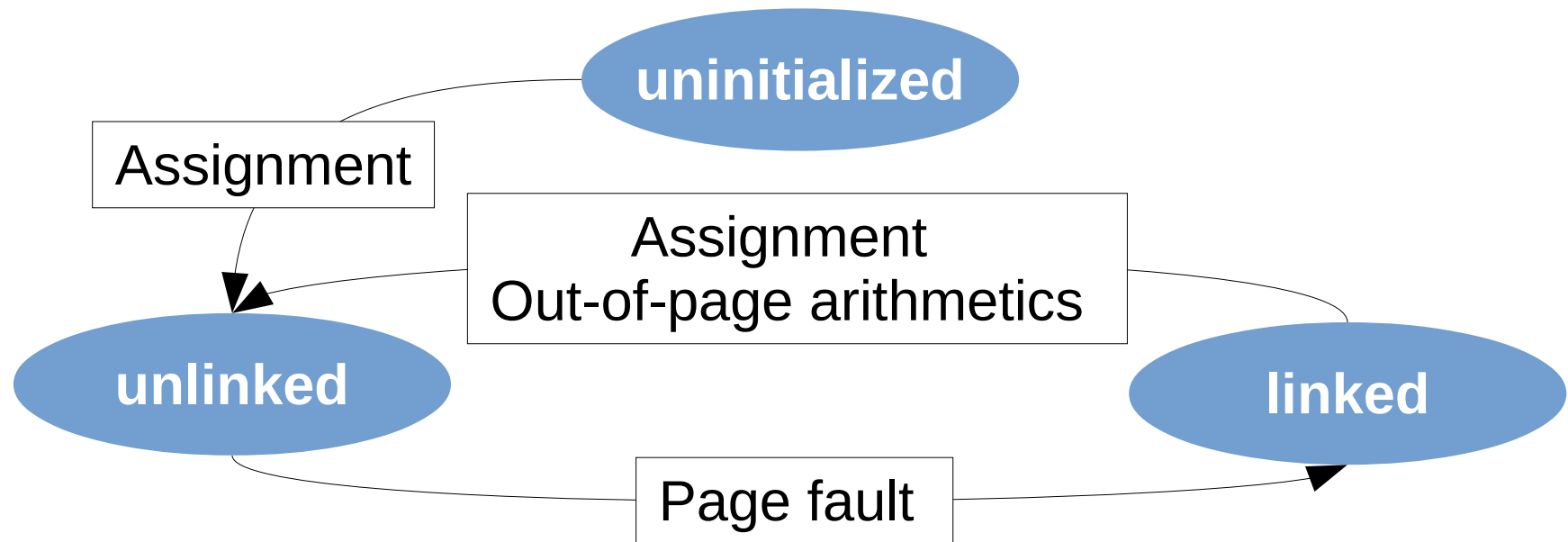
```
float y = *ptr;
```

```
// ptr+=4096;
```

}

ptr becomes inaccessible

# ActivePointers: state machine



# Challenge:

## Thread level address translation

- Reminder: warps = 32 threads in **lockstep**
- Warp threads may access different pointers
  - Faults for different pages
  - No faults

Divergence! Deadlocks!

# Idea:

## Translation aggregation mechanism

- Quickly identify fault-free accesses (fast path)
- Handle faults in order
- Aggregate faults to the same page
- Access the page cache using a non-divergent control flow

# Not covered in this talk...

- Translation aggregation algorithm
- Integration with GPUfs [SYSTOR16]
  - Highly concurrent page cache
  - Handling 4K pages
- Analysis of software TLB
- Optimizations

# Evaluation

- Commodity NVIDIA K80 GPU
- CUDA
- GPUfs



# Latency overheads

- A **single** GPU thread performing memory copy using ActivePointers

GPU cycles

Implementation	Read	Read+Inc
Raw access	225	257
ActivePointers	271 (+20%)	423 (+65%)

# Throughput overheads

- Same experiment with **full GPU occupancy**

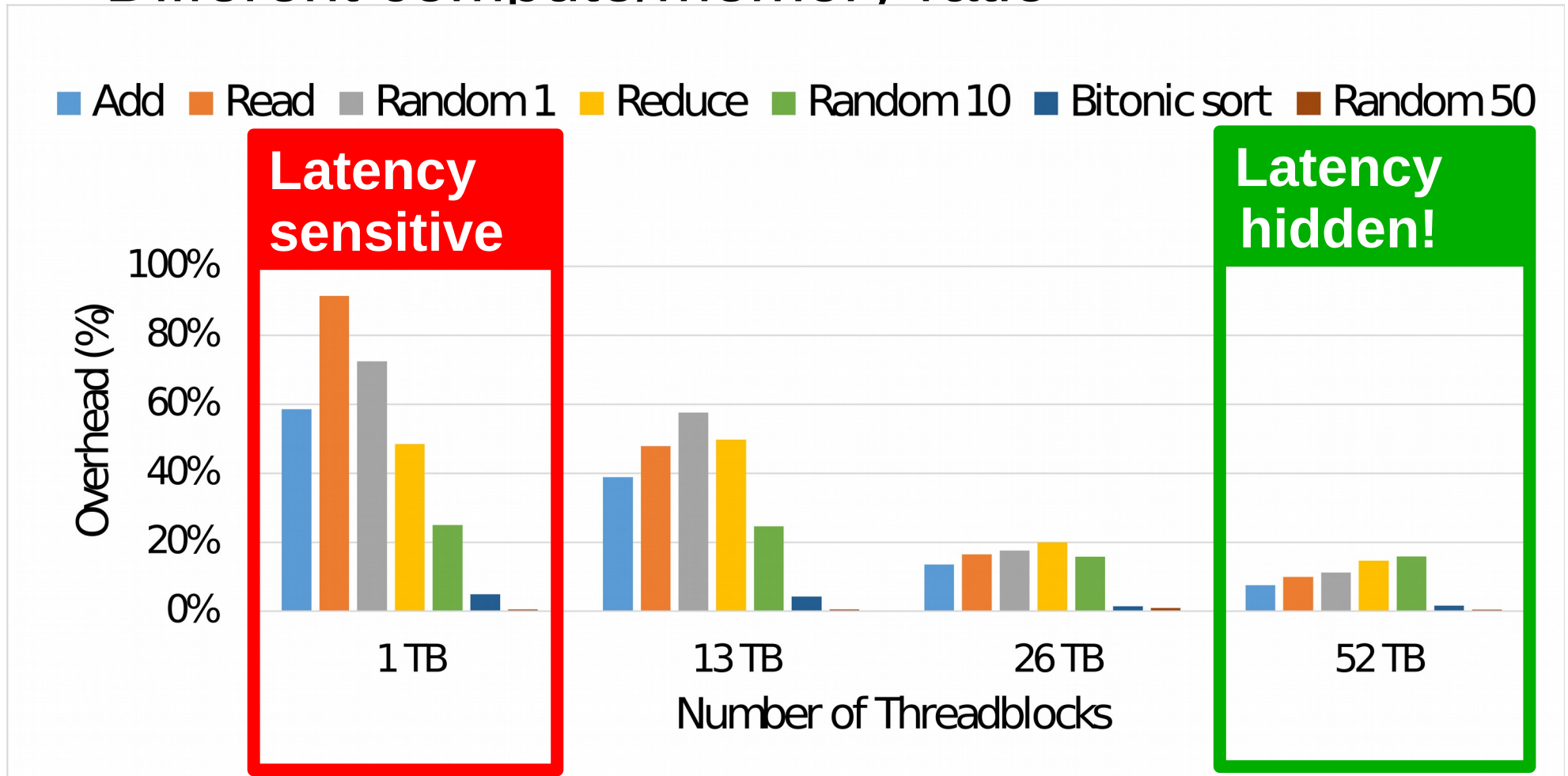
## Transfer bandwidth

4-byte	8-byte
99.65 GB/s (65.4%)	148.7 GB/s (97.6%)

Free-computation  
bubble

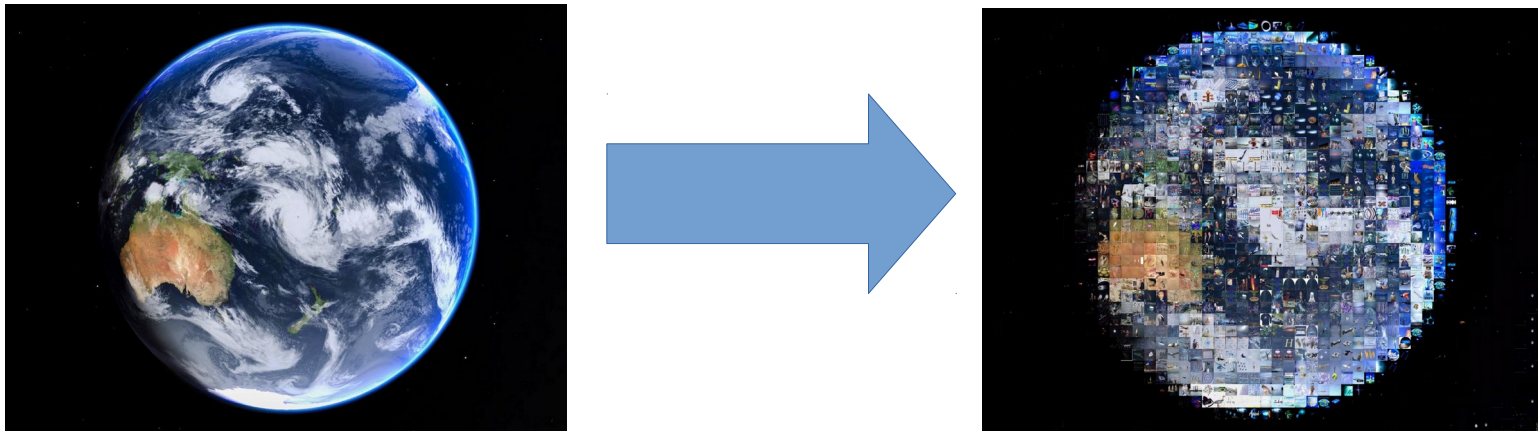
# Latency hiding – the key to performance

- Different compute/memory ratio



# Image collage

- End-to-end evaluation on K80 GPU



**Pointer access to 40GB DB file in GPU memory**

- **No measurable overhead**
- **2.6x** over 12 CPU cores with 256-bit AVX
- **3.5x** over CPU + GPU

# Take aways

- TLB-less address translation:
  - Beyond GPUs? (near-memory computing, FPGAs?)
  - Lightweight hardware support
- **GPU-centric VM management**
- GPU-as-peer-processor programming model

Source code at  
<https://github.com/gpufs>

# Thank you!

## Accelerated Computing Systems Lab

### Looking for postdocs

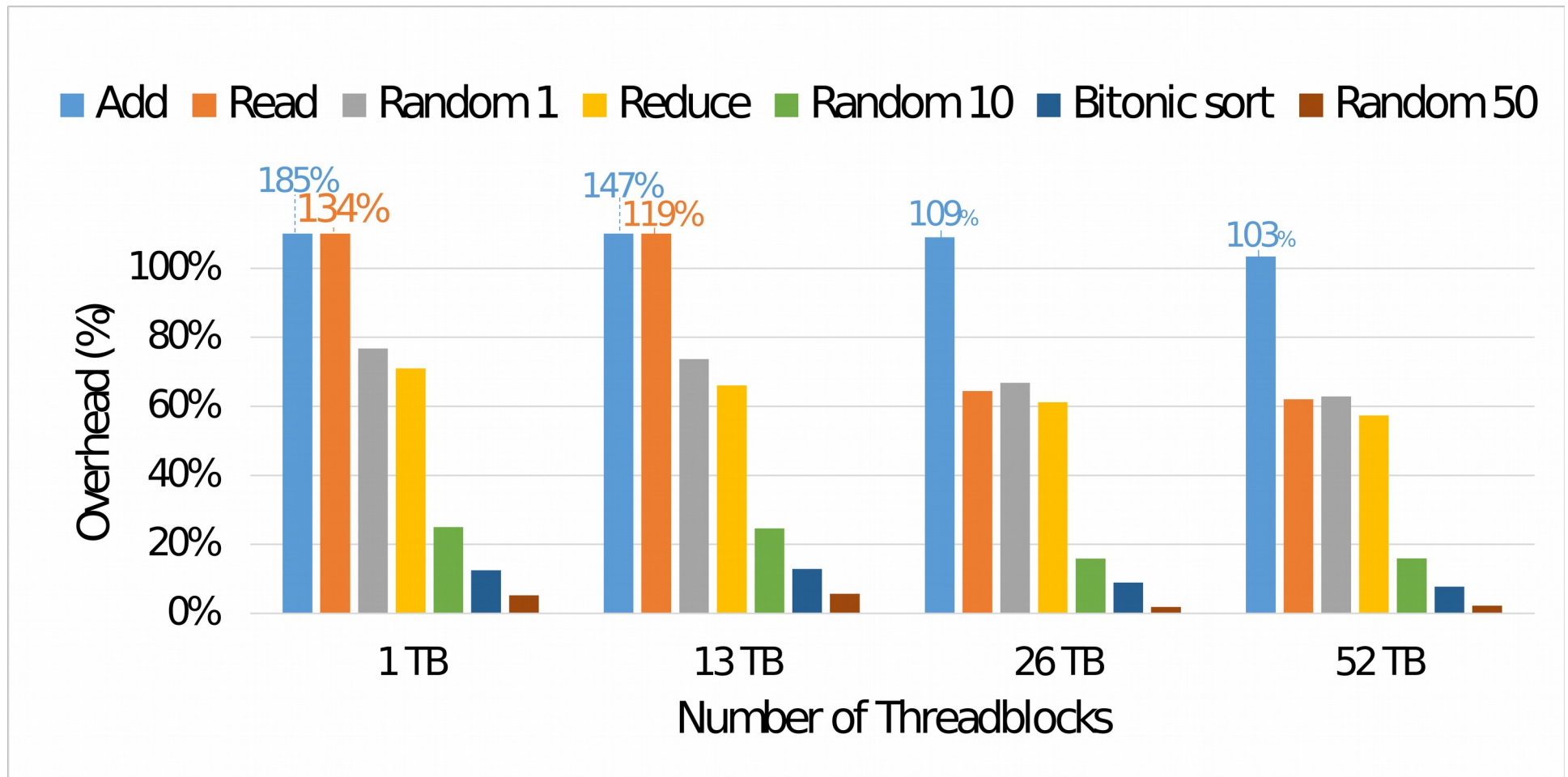


[mark@ee.technion.ac.il](mailto:mark@ee.technion.ac.il)

# Backup

# Latency hiding – the key to performance

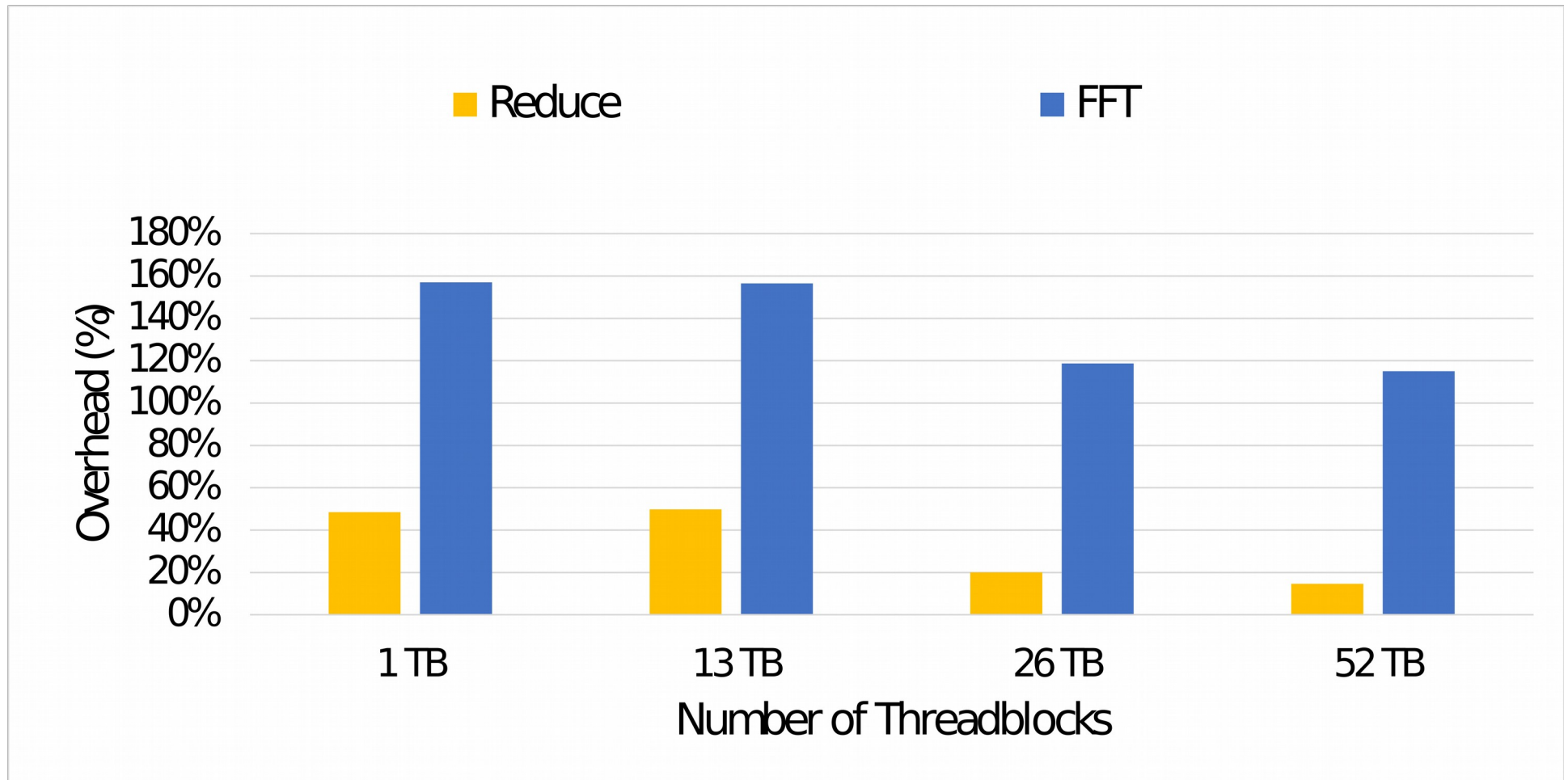
- Workloads with different compute/IO ratio (float)





# Lowlights

- Compiler heuristics?



# Discussion

- Register pressure
- compiler support
- Instructions for boundary checks
- I/O preemption

# Major compiler: register pressure

- Active pointer take 2 register (just like standard 64bit pointers)
- Additional meta-data is rarely accessed and can be stored in local memory
- Additional registers are required for page-faults and offset calculations
- Result in reduced optimization opportunities for the rest of the code

# Possible ways to cope with it

- Hardware acceleration can replace additional registers in offset calculation
- Most registers are only used in page-fault handling
- These registers are rarely accessed and can be moved to local memory
- The compiler heuristics need to be aware of Active pointers