EROFS: A Compression-friendly Readonly File System for Resource-scarce Devices

Xiang Gao[#], **Mingkai Dong**[*], Xie Miao[#],
Wei Du[#], Chao Yu[#], Haibo Chen[*,#]

[#]Huawei Technologies Co., Ltd.
[*]IPADS, Shanghai Jiao Tong University

USENIX ATC 2019, Renton, WA, USA

HUAWEI

SHANGHAI JIAO TONG UNIVERSITY

IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

Not enough space

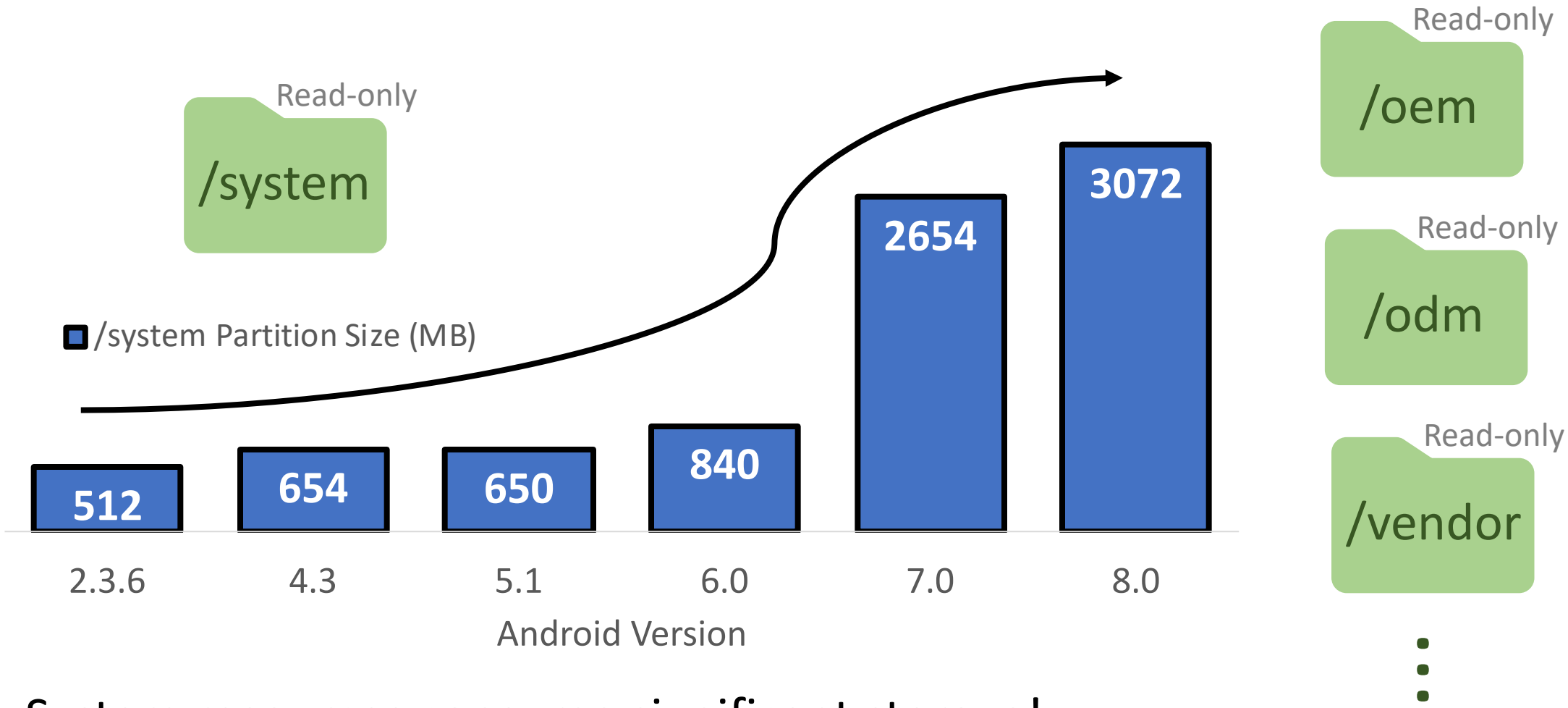
To install this app on this device, delete apps you no longer need. You can also free up space by deleting media files in Settings > Storage.

161MB needed

AirDroid
70.03MB
Last used 1 day ago

CANCEL CONTINUE

System resources in Android



System resources consume significant storage!

Read-only system partitions → compressed **read-only** file systems

Existing solution

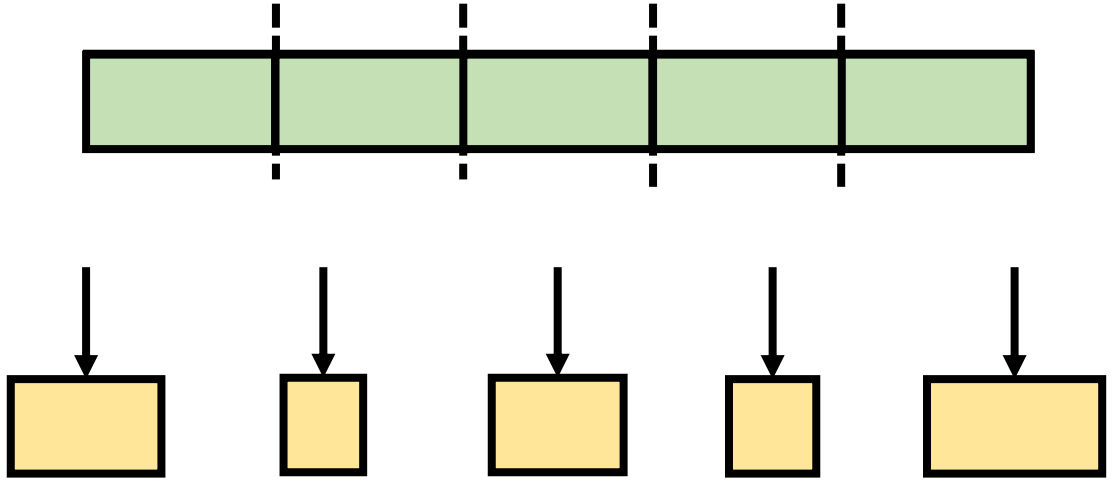
Squashfs is the state-of-the-art compressed read-only file system

We tried to use squashfs for system resources in Android

Result:

The system lagged and even froze for seconds and then rebooted

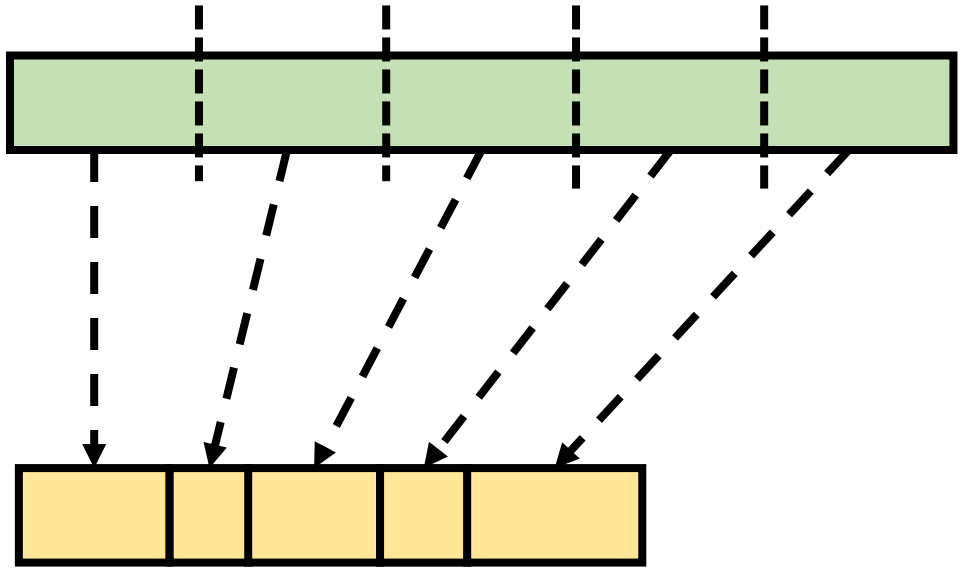
Why does Squashfs fail?



Fixed-sized input compression

1. Divide to fixed-sized chunks
2. Compress each chunk
3. Concatenate

Why does Squashfs fail?

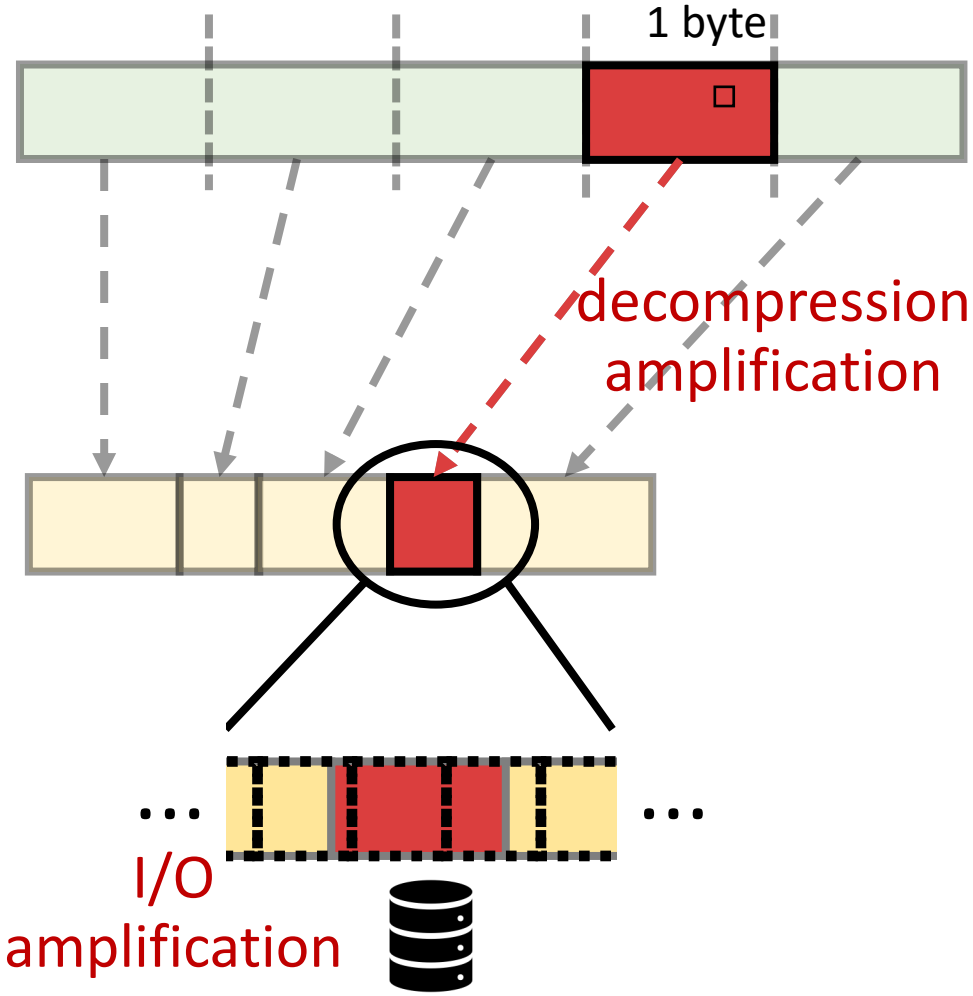


Fixed-sized input compression

1. Divide to fixed-sized chunks
2. Compress each chunk
3. Concatenate

Why does Squashfs fail?

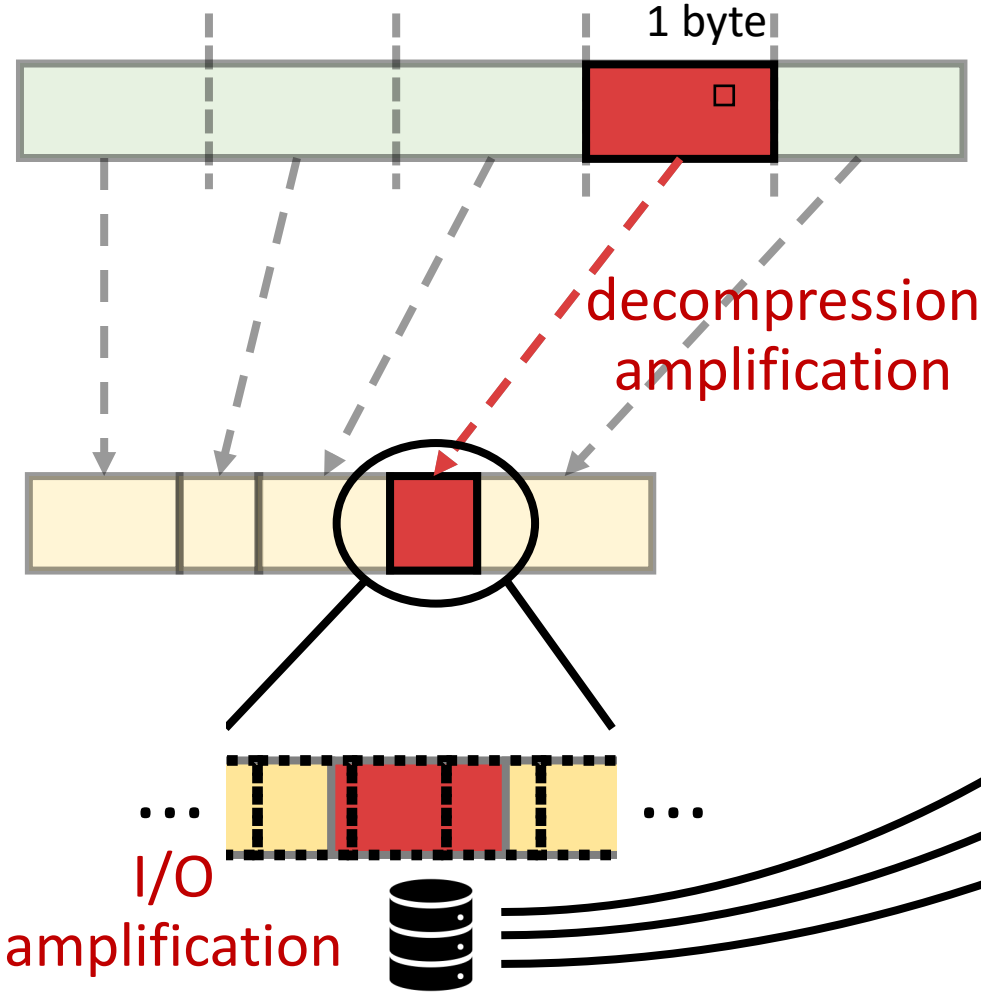
Read Amplification



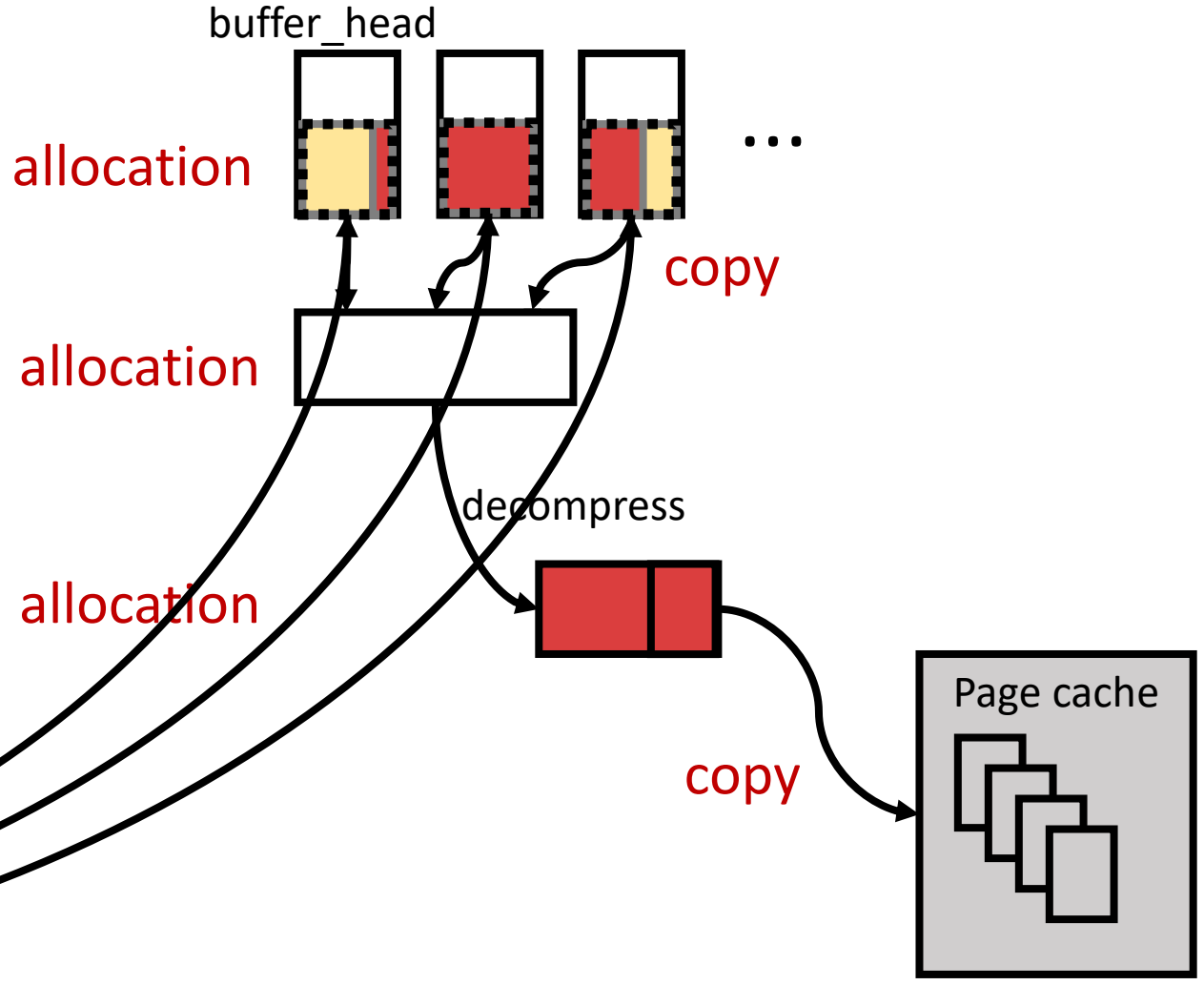
Fixed-sized input compression

Why does Squashfs fail?

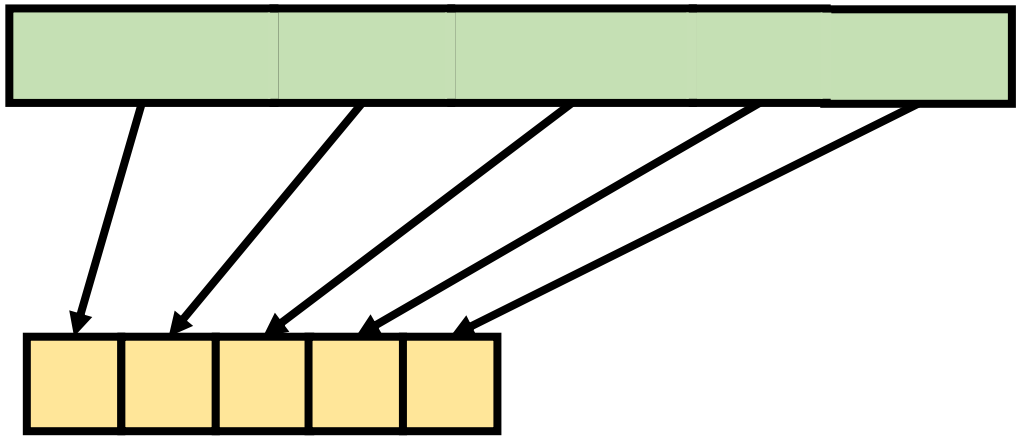
Read Amplification



Massive Memory Consumption



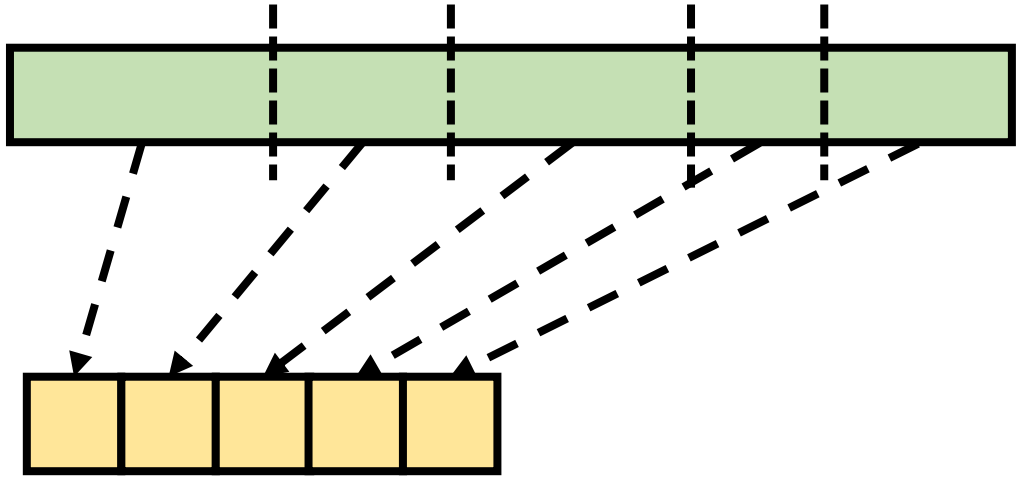
EROFS



Fixed-sized output compression

1. Prepare a large amount of data
2. Compress to a fixed-sized block
3. Repeat

EROFS



Fixed-sized output compression

1. Prepare a large amount of data
2. Compress to a fixed-sized block
3. Repeat

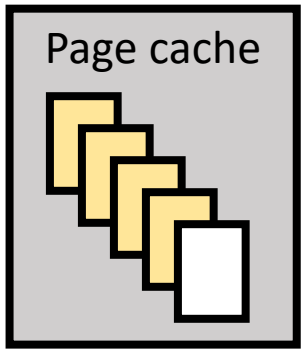
- ✓ Reduce read amplification
- ✓ Better compression ratio
- ✓ Allows in-place decompression

Choosing the page for I/O

Cached I/O

For partially decompressed blocks
Allocate a page in dedicated page cache for I/O
✓ Following decompression can reuse the cached page

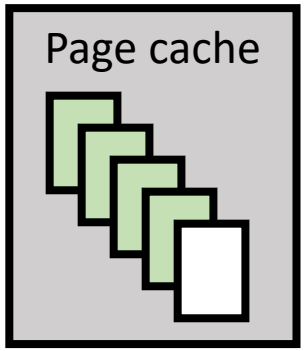
Page cache for partially compressed blocks



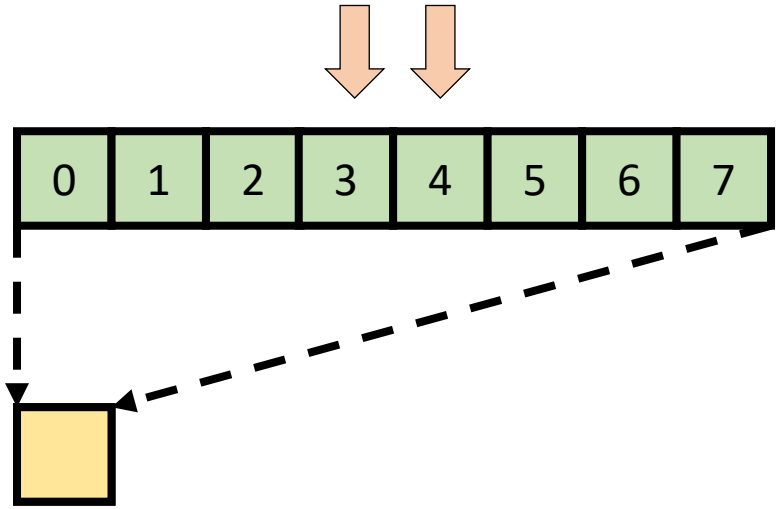
In-place I/O

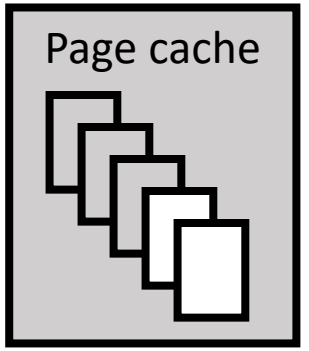
For blocks to be fully decompressed
Reuse the page allocated by VFS if possible
✓ Memory allocation and consumption are reduced

Page cache of requested file



Decompression

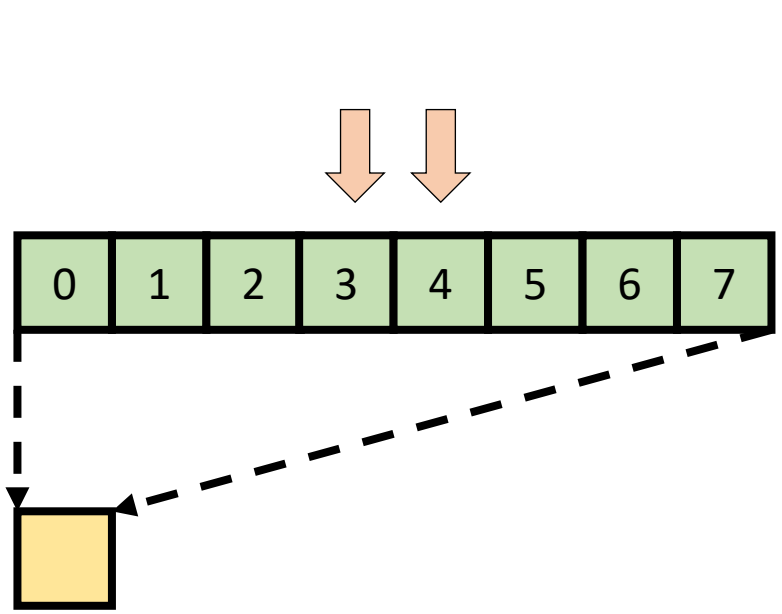




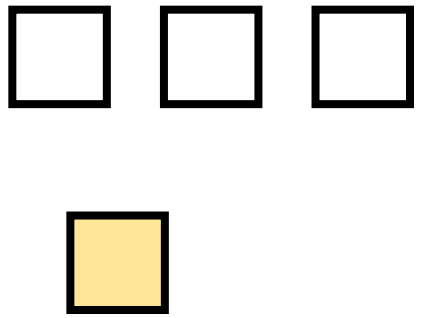
Decompression

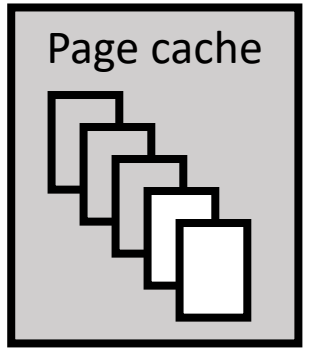
Vmap decompression

1. Count data blocks to decompress.
2. Allocate temporary physical pages or choose pages allocated by VFS.



How data is compressed

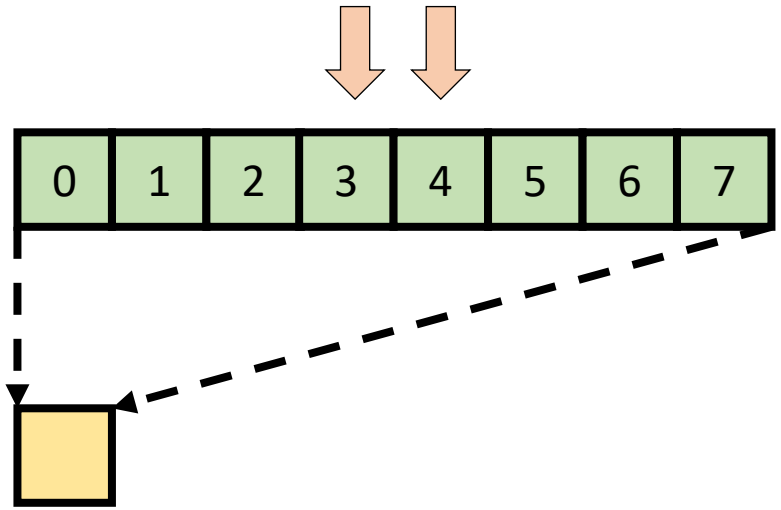




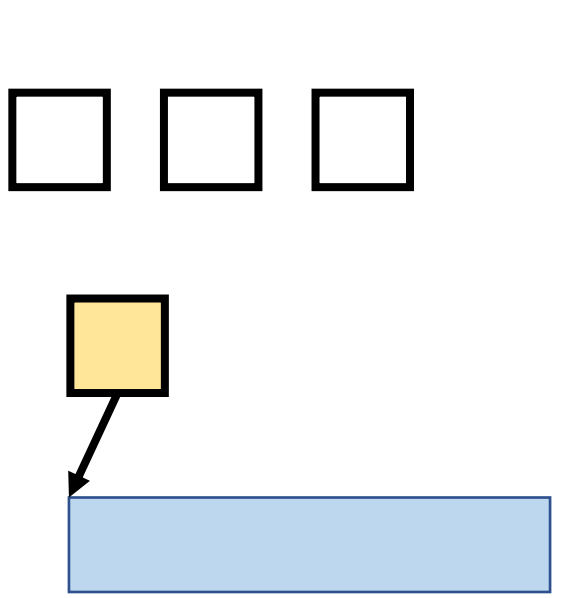
Decompression

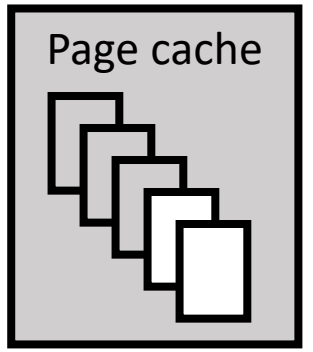
Vmap decompression

1. Count data blocks to decompress.
2. Allocate temporary physical pages or choose pages allocated by VFS.
3. Allocate a continuous VM area via vmap(), and map physical pages in the area.



How data is compressed

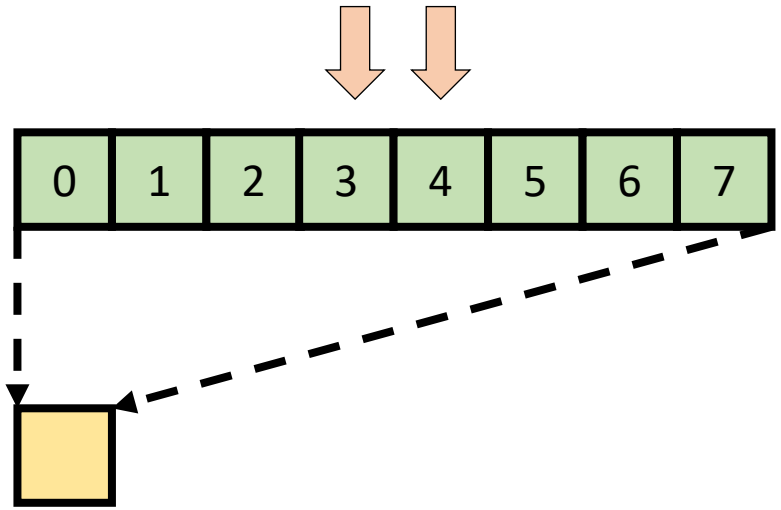




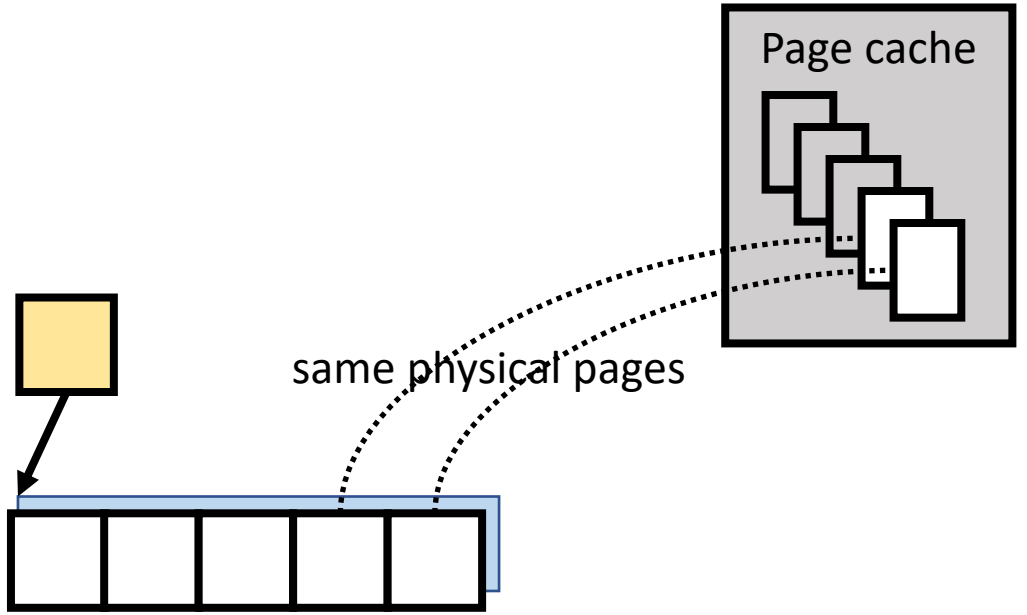
Decompression

Vmap decompression

1. Count data blocks to decompress.
2. Allocate temporary physical pages or choose pages allocated by VFS.
3. Allocate a continuous VM area via vmap(), and map physical pages in the area.



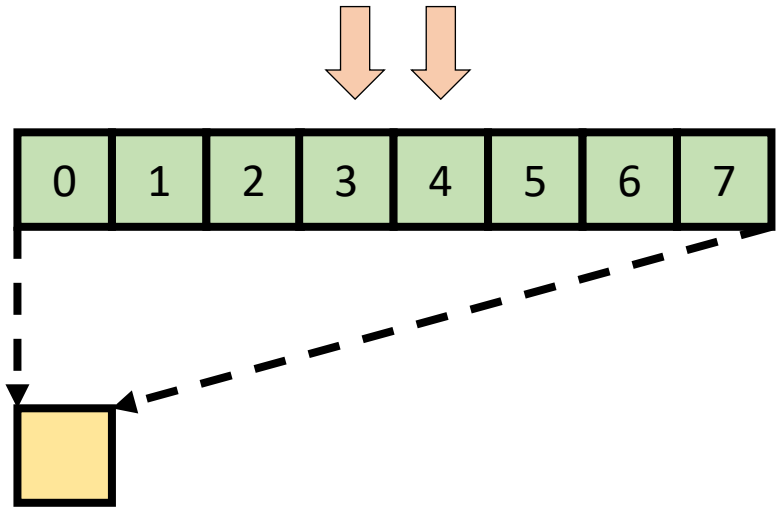
How data is compressed



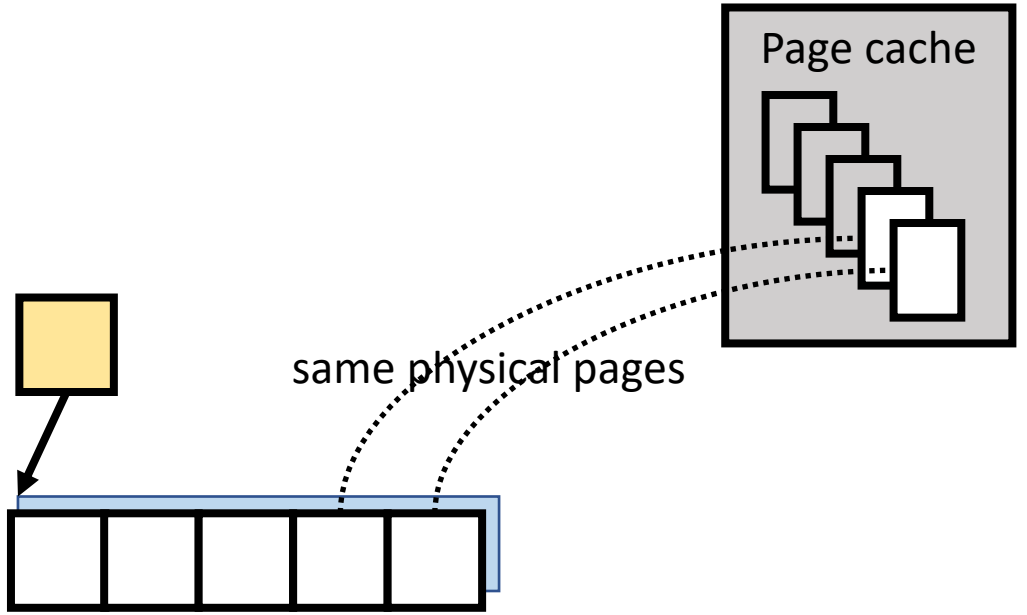
Decompression

Vmap decompression

1. Count data blocks to decompress.
2. Allocate temporary physical pages or choose pages allocated by VFS.
3. Allocate a continuous VM area via vmap(), and map physical pages in the area.
4. For in-place I/O, copy the compressed block to a temporary per-CPU page.



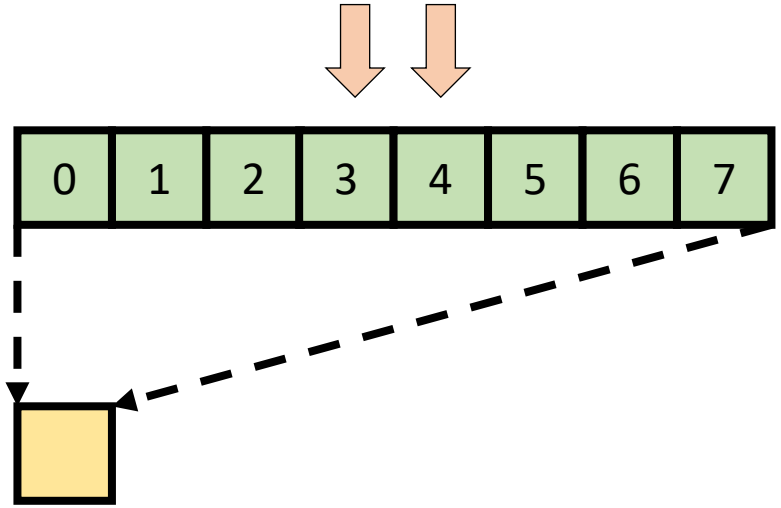
How data is compressed



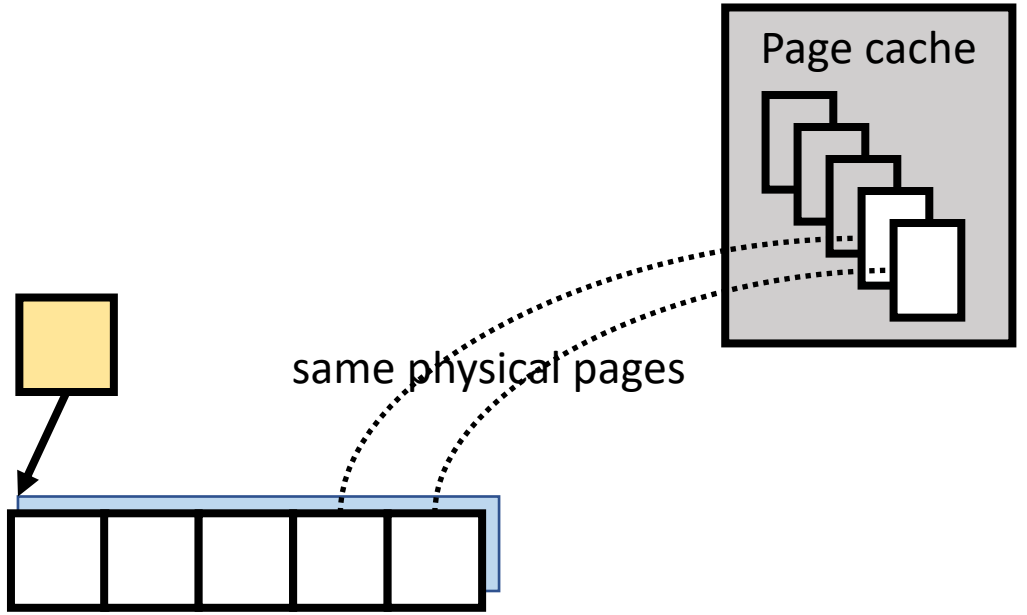
Decompression

Vmap decompression

1. Count data blocks to decompress.
2. Allocate temporary physical pages or choose pages allocated by VFS.
3. Allocate a continuous VM area via vmap(), and map physical pages in the area.
4. For in-place I/O, copy the compressed block to a temporary per-CPU page.
5. Decompress to the VM area.



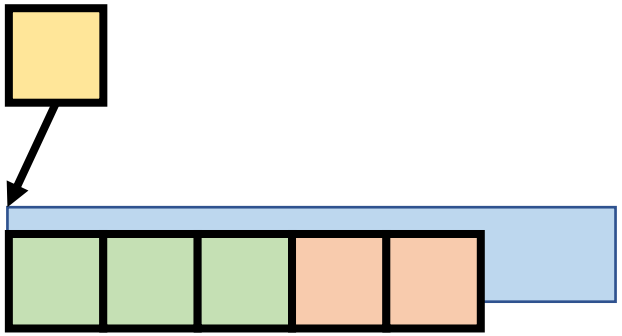
How data is compressed



Decompression

Vmap decompression

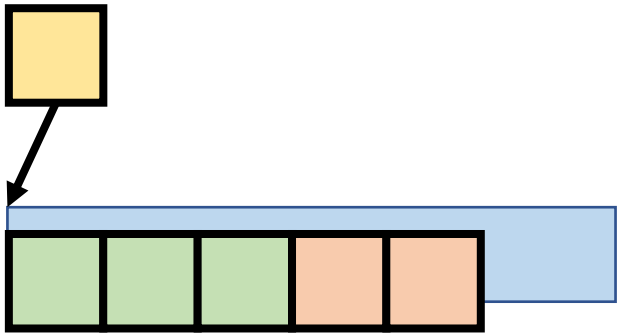
- ✓ For all cases
- ✗ Frequent vmap/vunmap
- ✗ Unbounded physical page allocations
- ✗ Data copy for in-place I/O



Decompression

Vmap decompression

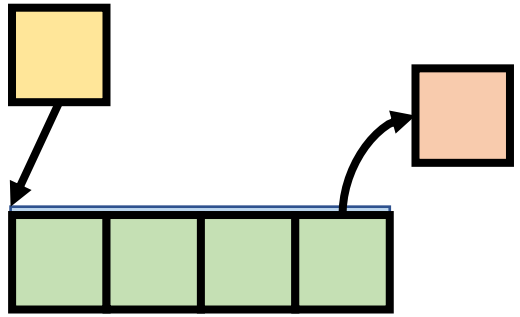
- ✓ For all cases
- ✗ Frequent vmap/vunmap
- ✗ Unbounded physical page allocations
- ✗ Data copy for in-place I/O



Buffer decompression

Pre-allocate four-page per-CPU buffers

- ✗ For decompression <4 pages
- ✓ No vmap/vunmap
- ✓ No physical page allocation
- ✓ No data copy for in-place I/O



Decompression

Vmap decompression

- ✓ For all cases
- ✗ Frequent vmap/vunmap
- ✗ Unbounded physical page allocations
- ✗ Data copy for in-place I/O

Buffer decompression

Pre-allocate four-page per-CPU buffers

- ✗ For decompression <4 pages
- ✓ No vmap/vunmap
- ✓ No physical page allocation
- ✓ No data copy for in-place I/O

Details in the paper

Rolling decompression

- ✓ For decompression < a pre-allocated VM area size
- ✓ No vmap/vunmap
- ✓ No physical page allocation
- ✗ Data copy for in-place I/O

In-place decompression

- ✓ No data copy for in-place I/O

- ➢ Decompression policy
- ➢ Optimizations

Evaluation Setup

Platform	CPU	DRAM	Storage
HiKey 960	Kirin 960 (Cortex-A73 x 4 + Cortex-A53 x 4)	3 GB	32 GB UFS
Low-end smartphone	MT6765 (Cortex-A53 x 8)	2 GB	32 GB eMMC
High-end smartphone	Kirin 980 (Cortex-A76 x 4 + Cortex-A55 x 4)	6 GB	64 GB UFS

Micro-benchmarks

- Platform: HiKey 960
- Tool: FIO
- Workload: enwik9, silesia.tar

Application benchmarks

- Platform: smartphones
- Workload: 13 popular applications

Evaluated file systems

EROFS: LZ4, 4KB-sized output

Squashfs: LZ4, {4,8,16,128}KB chunk size

Btrfs: LZO, 128KB chunk size
readonly mode w/o integrity checks

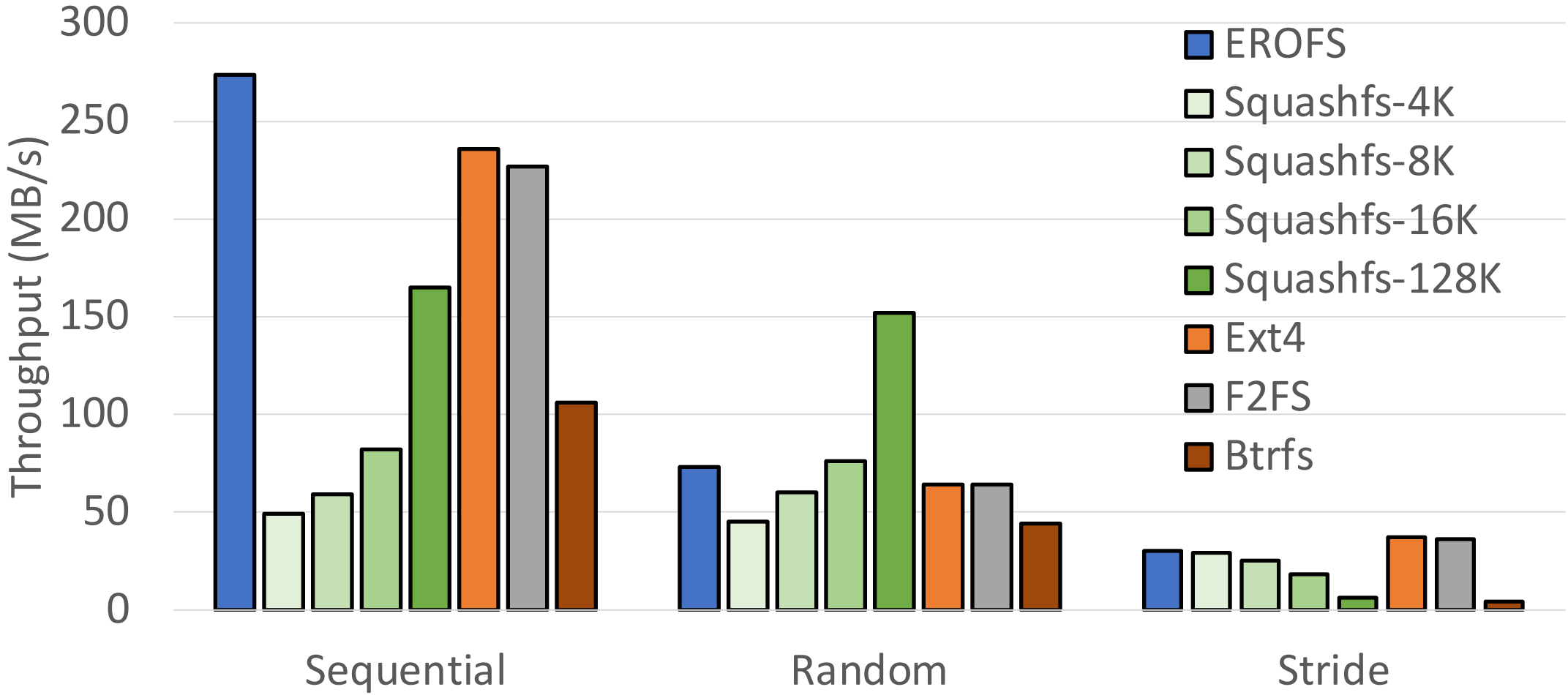
Ext4: no compression

F2FS: no compression

More results in the paper

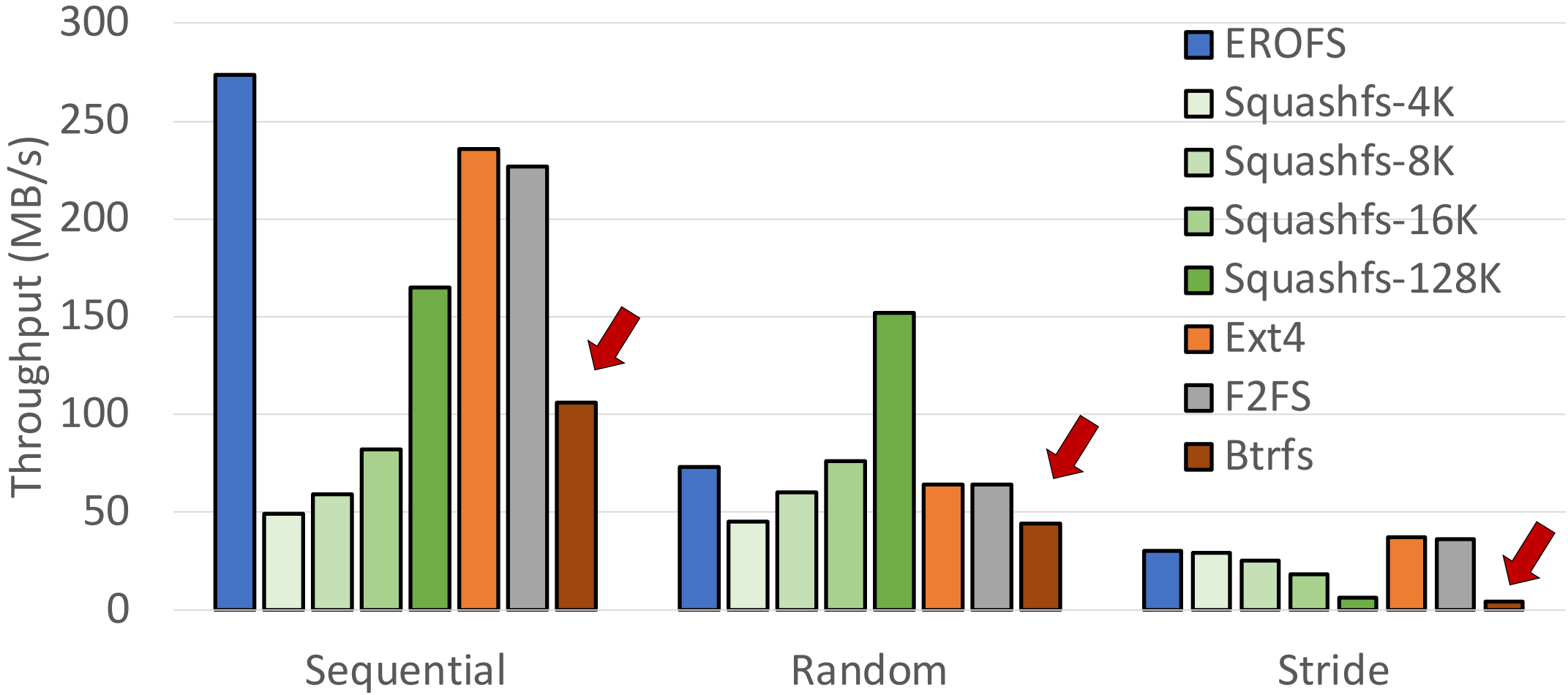
Micro-benchmark: FIO Throughput

FIO, enwik9, HiKey 960 A73 2362 MHz



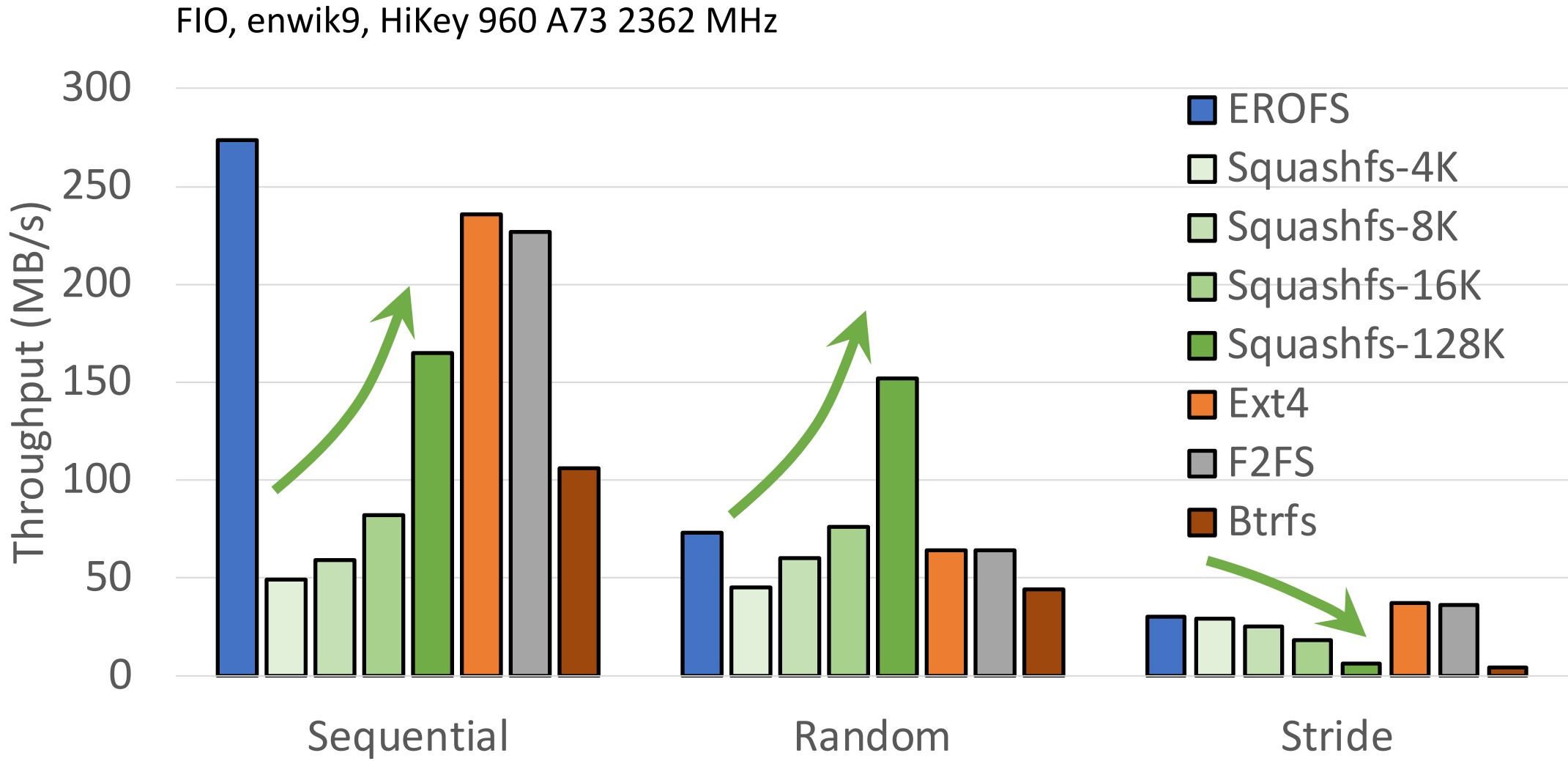
Micro-benchmark: FIO Throughput

FIO, enwik9, HiKey 960 A73 2362 MHz



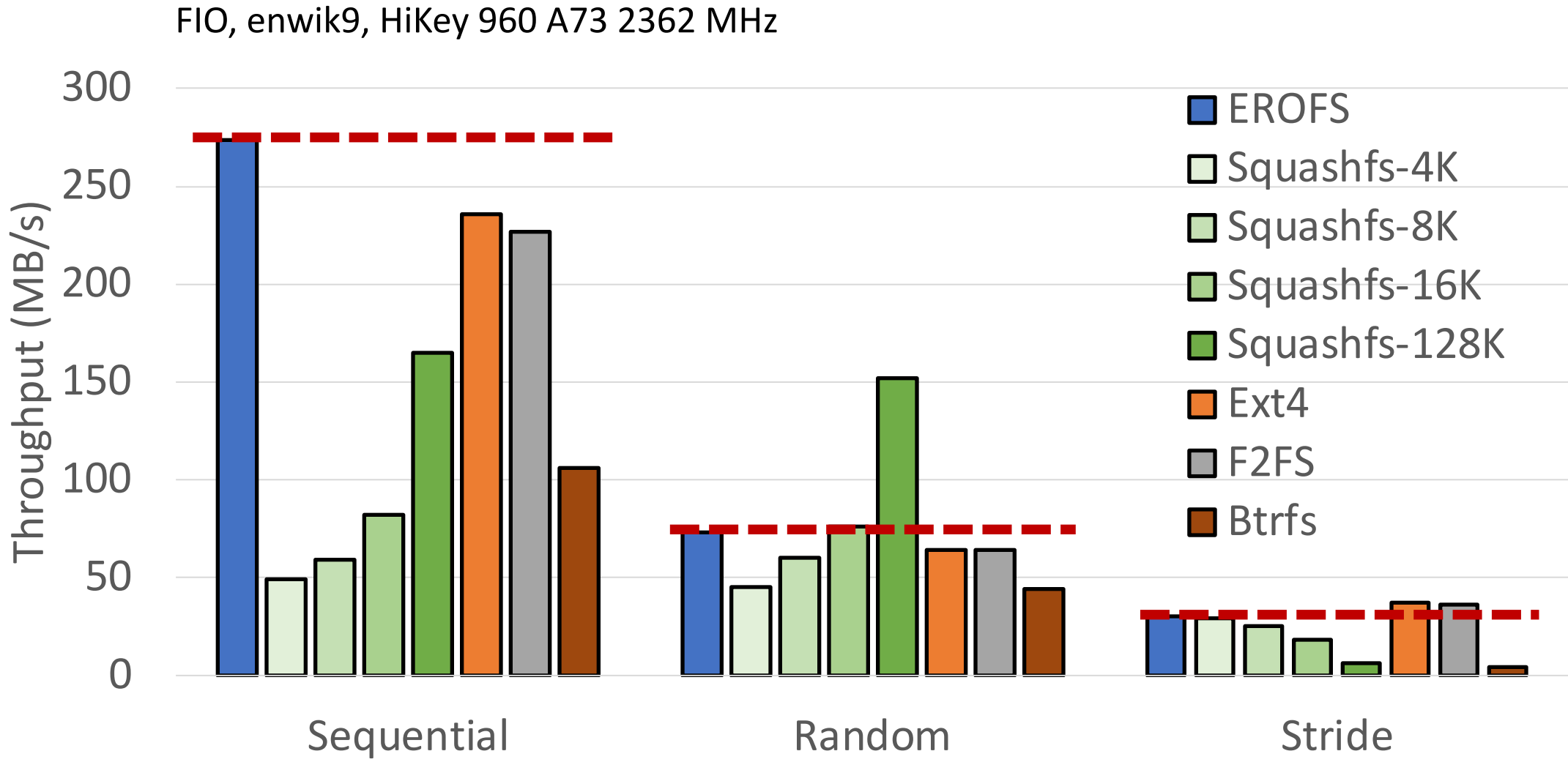
Btrfs performs worst in all cases.

Micro-benchmark: FIO Throughput



Larger chunk size brings better performance for Squashfs, if the cached results can are used.

Micro-benchmark: FIO Throughput



EROFS performs comparable or even better than Ext4.

Read Amplification and Resource Consumption

	IO (MB)			Consumption (MB)	
	Seq.	Random	Stride	Storage	Memory
Requested/Ext4	16	16	16	953.67	988.51
Squashfs-4K	10.65	26.19	26.23	592.43	1597.50
Squashfs-8K	9.82	33.52	34.08	530.43	1534.09
Squashfs-16K	9.05	46.42	48.32	479.38	1481.12
Squashfs-128K	7.25	165.27	203.91	379.76	1379.84
EROFS	**10.14**	**26.12**	**25.93**	**533.67**	**1036.88**

Read Amplification and Resource Consumption

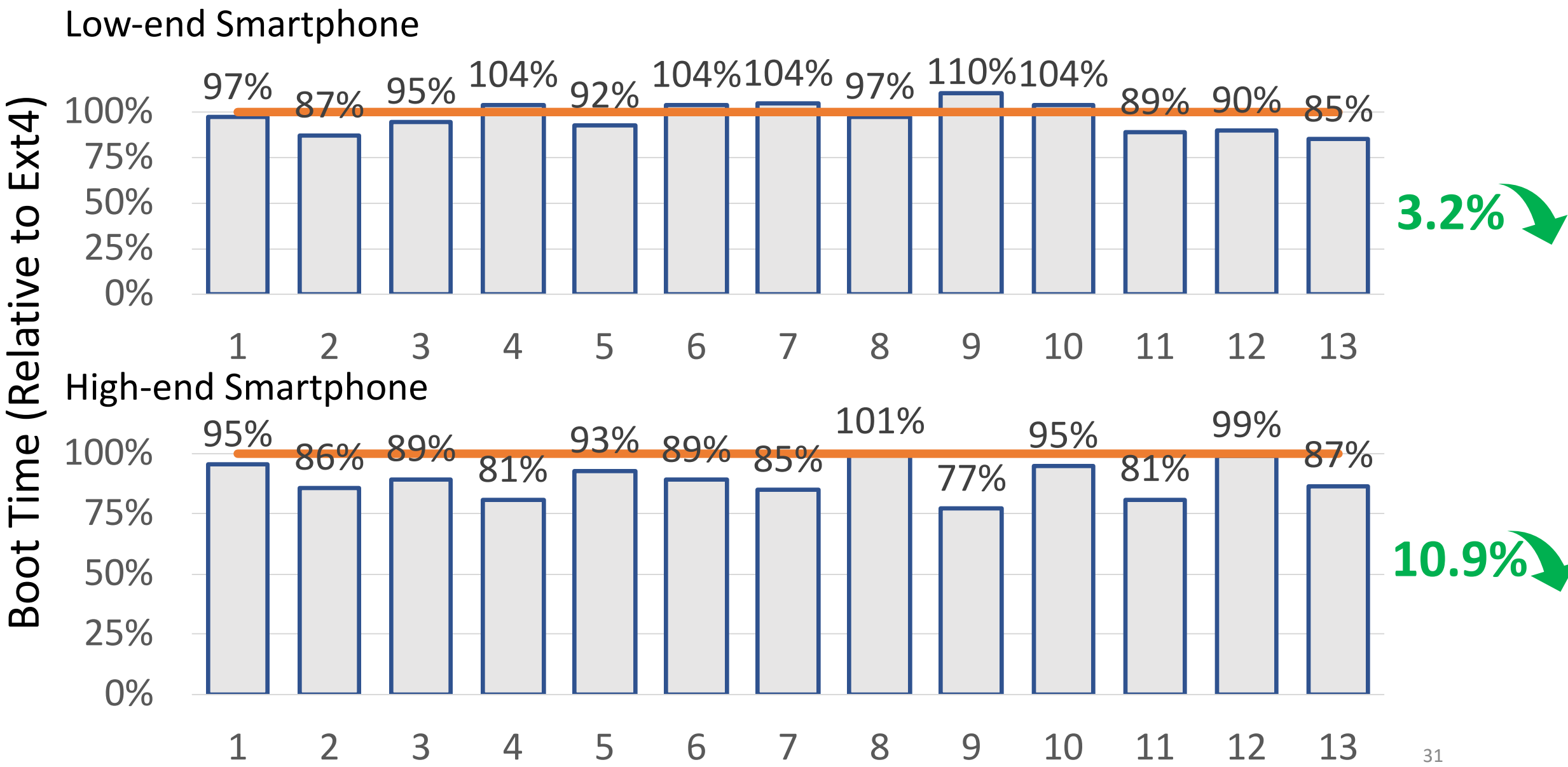
	IO (MB)			Consumption (MB)	
	Seq.	Random	Stride	Storage	Memory
Requested/Ext4	16	16	16	953.67	988.51
Squashfs-4K	10.65	26.19	26.23	592.43	1597.50
Squashfs-8K	9.82	33.52	34.08	530.43	1534.09
Squashfs-16K	9.05	46.42	48.32	479.38	1481.12
Squashfs-128K	7.25	**165.27**	**203.91**	379.76	1379.84
EROFS	**10.14**	**26.12**	**25.93**	**533.67**	**1036.88**

84% **87%**

Read Amplification and Resource Consumption

	IO (MB)			Consumption (MB)	
	Seq.	Random	Stride	Storage	Memory
Requested/Ext4	16	16	16	**953.67**	988.51
Squashfs-4K	10.65	26.19	26.23	592.43	**1597.50**
Squashfs-8K	9.82	33.52	34.08	530.43	1534.09
Squashfs-16K	9.05	46.42	48.32	479.38	1481.12
Squashfs-128K	7.25	**165.27**	**203.91**	379.76	1379.84
EROFS	**10.14**	**26.12**	**25.93**	**533.67**	**1036.88**
		84% ↓	**87%** ↓	**44%** ↓	**92%** ↓

Real-world Application Boot Time



Deployment

- ✓Deployed in HUAWEI EMUI 9.1 as a top feature
- ✓Upstreamed to Linux 4.19
- ✓System storage consumption decreased **>30%**
- ✓Performance **comparable** or even **better** than Ext4
- ✓Running on **10,000,000+** smartphones

Conclusion

EROFS: an **E**nhanced **R**ead-**O**nly **F**ile **S**ystem with compression support

Fixed-sized output compression with four decompression approaches:

- Vmap decompression
- Buffer decompression
- Rolling decompression
- In-place decompression

Thank you & questions?

Running on **10,000,000+** smartphones

✓Reduce system storage consumption **>30%**

✓Provide **comparable** or even **better** performance than Ext4

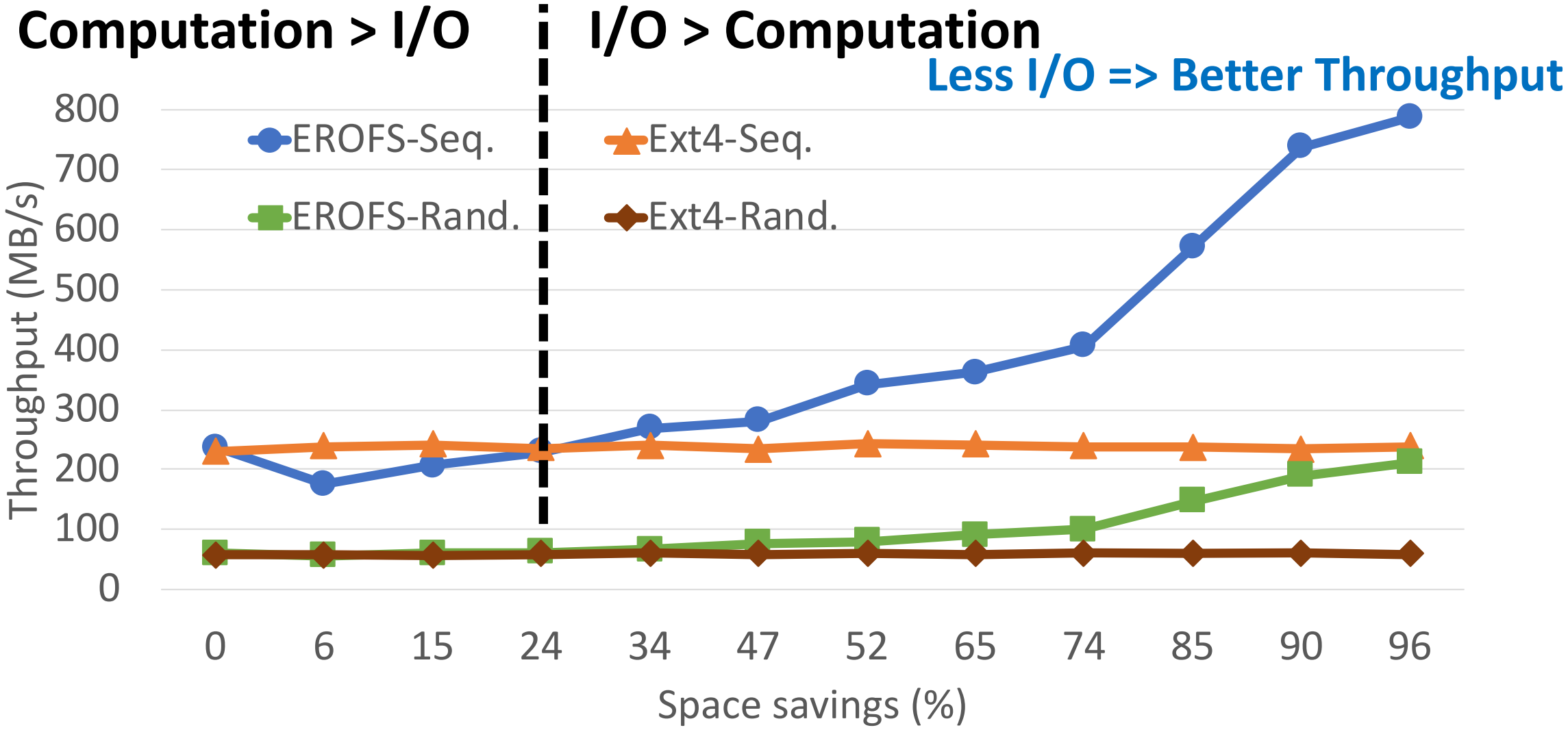
HUAWEI

SHANGHAI JIAO TONG UNIVERSITY

IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

Backup Slides

Throughput and space savings



FIO, customized workload generated from enwik9, HiKey 960 A73 2362 MHz

Decompression

Observation:

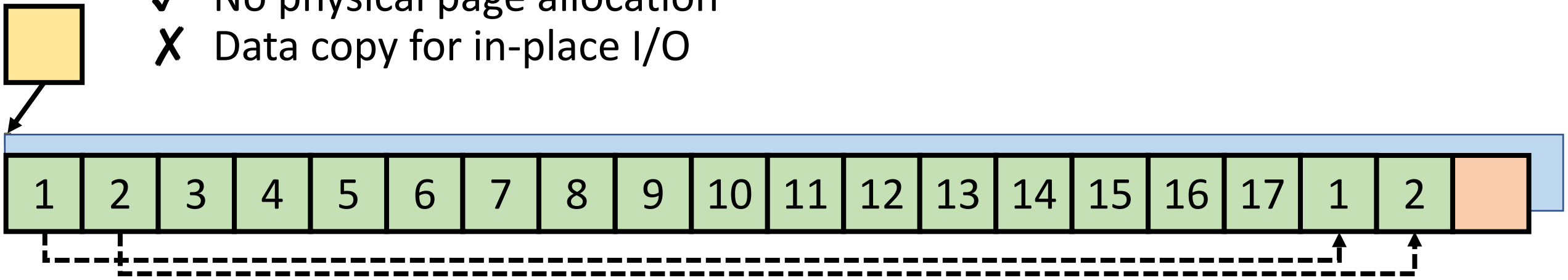
In decompression, LZ4 looks backward at most 64KB of decompressed data.

Rolling decompression

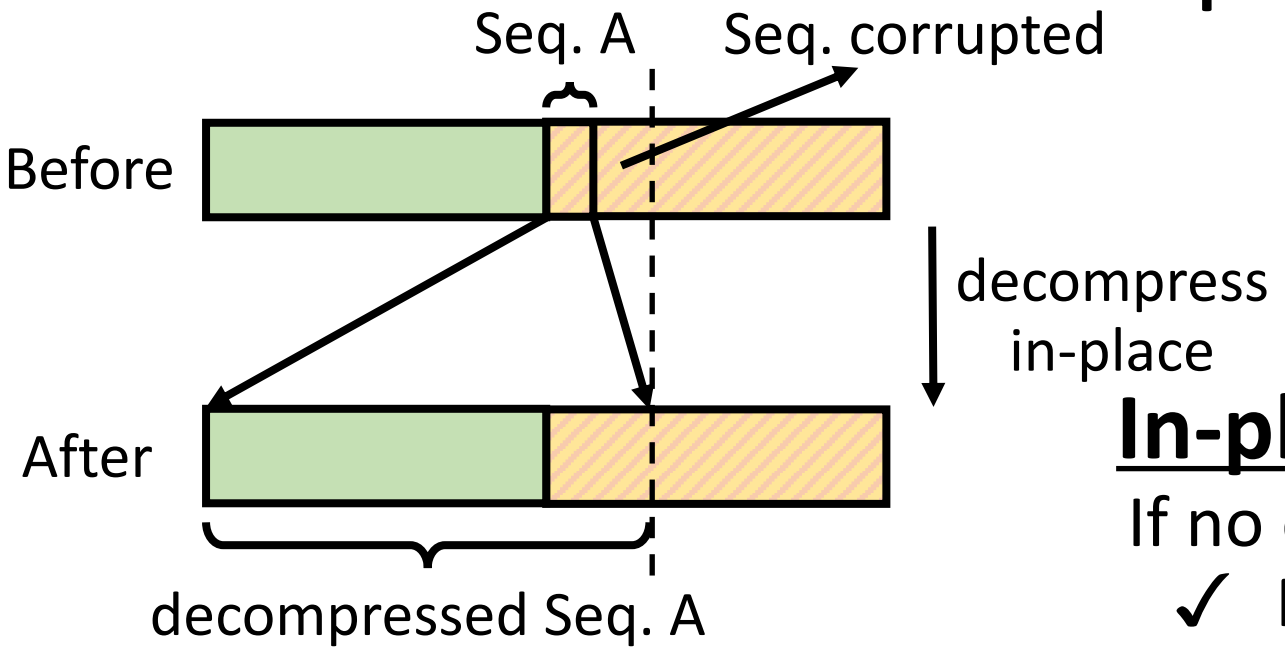
Pre-allocate a large VM area and 17 physical pages per CPU

Use 17 physical pages ***in turn***.

- ✓ For decompression < the VM area size
- ✓ No vmap/vunmap
- ✓ No physical page allocation
- ✗ Data copy for in-place I/O



Decompression



In-place decompression

If no corruption will happen

- ✓ For decompression < the VM area size
- ✓ No vmap/vunmap
- ✓ No physical page allocation
- ✓ No data copy for in-place I/O