Understanding Manycore Scalability of File Systems

Changwoo Min, Sanidhya Kashyap, Steffen Maass
Woonhak Kang, and Taesoo Kim

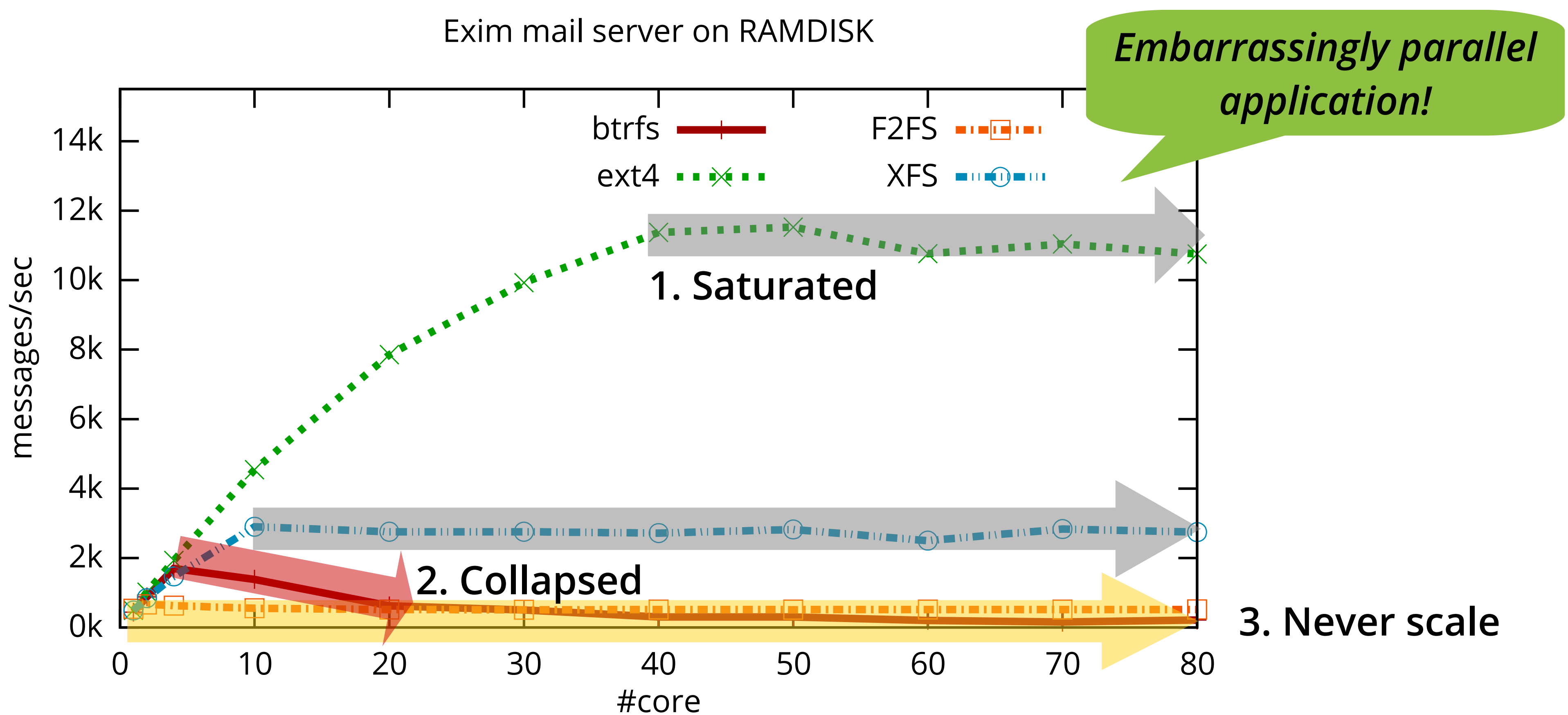
Georgia Institute of Technology

Application must parallelize I/O operations

- **Death of single core CPU scaling**
  - CPU clock frequency: 3 ~ 3.8 GHz
  - # of physical cores: up to 24 (Xeon E7 v4)
- **From mechanical HDD to flash SSD**
  - IOPS of a commodity SSD: 900K
  - Non-volatile memory (e.g., 3D XPoint): 1,000x ↑

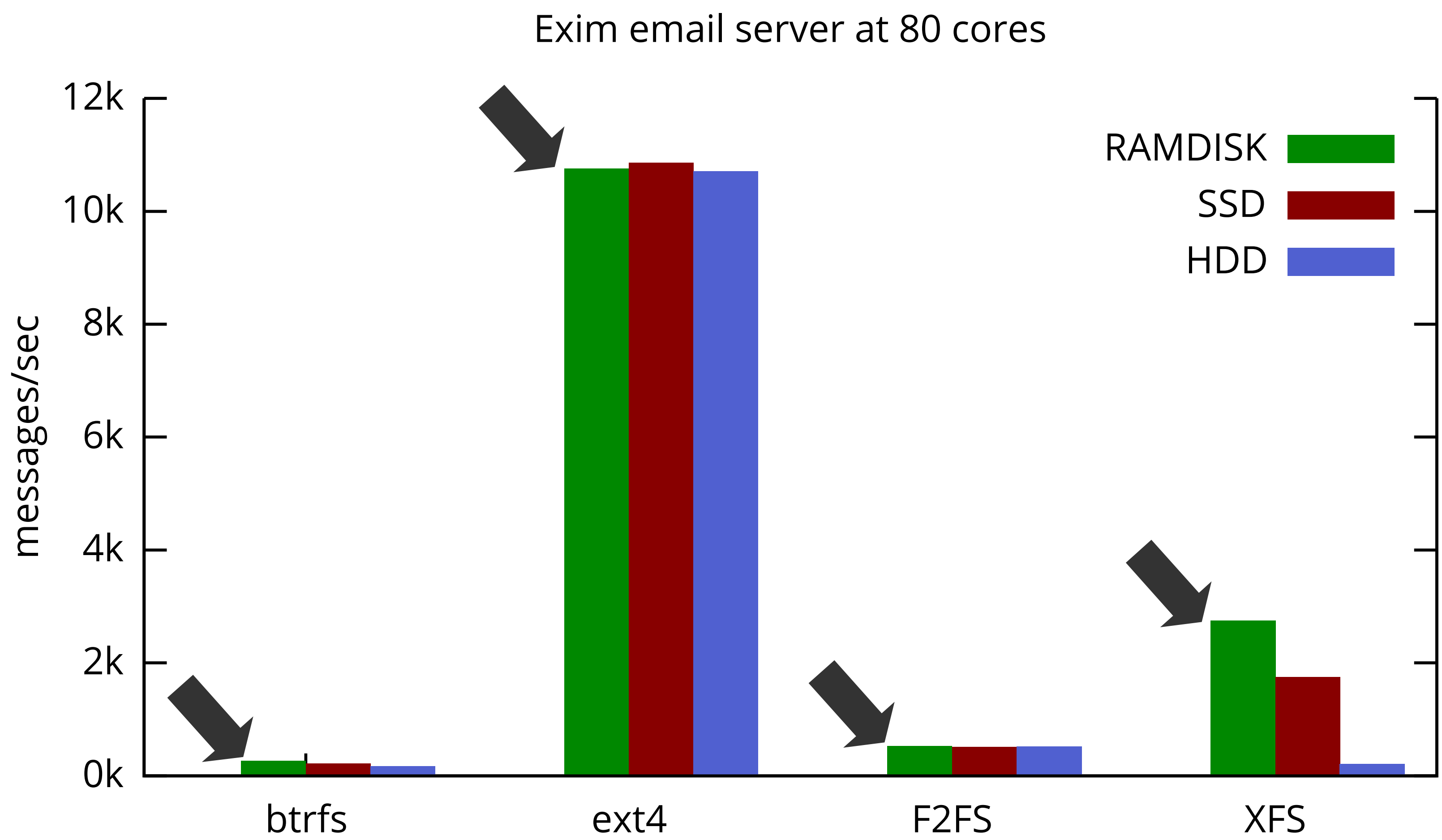
But file systems become a scalability bottleneck

Problem: Lack of understanding in internal scalability behavior



- Intel 80-core machine: 8-socket, 10-core Xeon E7-8870
- RAM: 512GB, 1TB SSD, 7200 RPM HDD

Even in slower storage medium file system becomes a bottleneck



Outline

- Background
- **FxMark design**
  - A file system benchmark suite for manycore scalability
- Analysis of five Linux file systems
- Pilot solution
- Related work
- Summary

Research questions

- What file system operations are not scalable?
- Why they are not scalable?
- Is it the problem of implementation or design?

Technical challenges

- Applications are usually stuck with a few bottlenecks

  → cannot see the next level of bottlenecks before resolving them

  → difficult to understand overall scalability behavior

- How to systematically stress file systems to understand scalability behavior

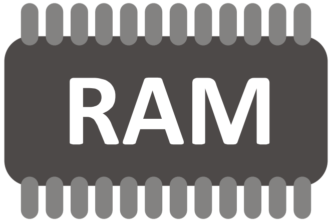
FxMark: evaluate & analyze manycore scalability of file systems

FxMark: 19 micro-benchmarks | 3 applications

File systems:

Memory FS	Journaling FS		CoW FS	Log FS
tmpfs	ext4 J/NJ	XFS	btrfs	F2FS

Storage medium:



 # core: 1, 2, 4, 10, 20, 30, 40, 50, 60, 70, 80

FxMark: evaluate & analyze manycore scalability of file systems

FxMark: 19 micro-benchmarks | 3 applications

File systems:

- tmpfs — Memory FS
- ext4 J/NJ — Journali
- btrfs — CoW FS
- F2FS — Log FS

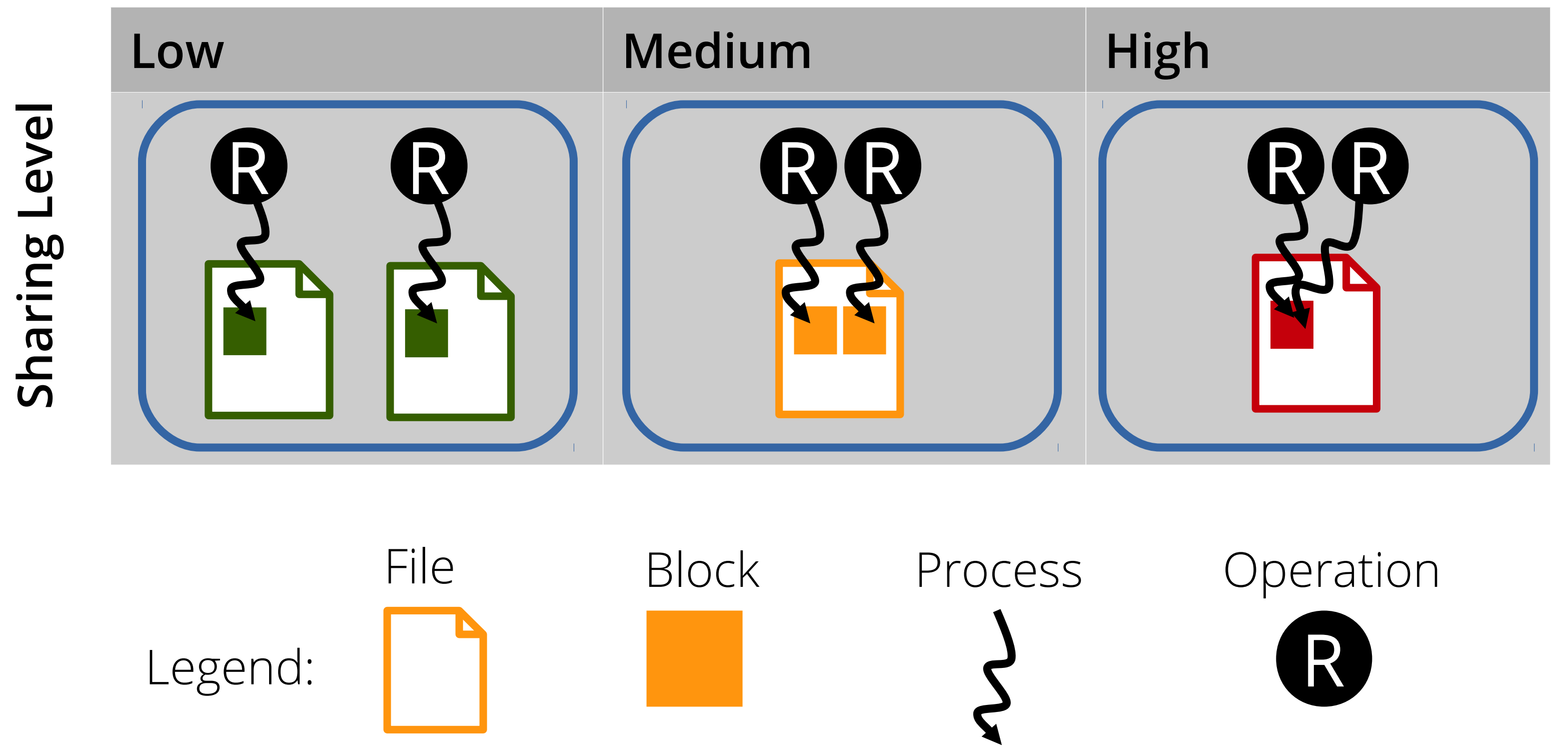
>4,700

Storage medium:

 # core: 1, 2, 4, 10, 20, 30, 40, 50, 60, 70, 80

Microbenchmark: unveil hidden scalability bottlenecks

- Data block read



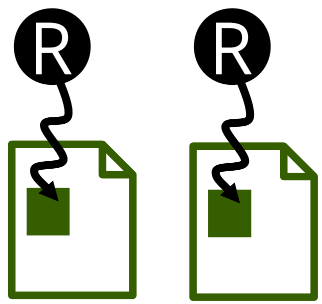
Stress different components with various sharing levels

Type	Mode	Operation	Sharing Level		
			LOW	MEDIUM	HIGH
DATA	READ	BLOCK READ	✓	✓	✓
	WRITE	OVERWRITE	✓	✓	-
		APPEND	✓	-	-
		TRUNCATE	✓	-	-
		SYNC	✓	-	-
META	READ	PATH NAME READ	✓	✓	✓
		DIRECTORY LIST	✓	✓	-
	WRITE	CREATE	✓	✓	-
		UNLINK	✓	✓	-
		RENAME	✓	✓	-

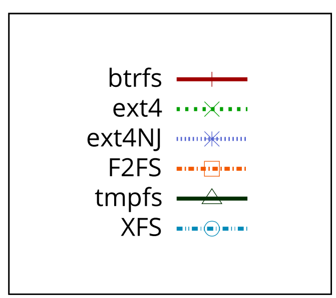
Evaluation

- **Data block read**

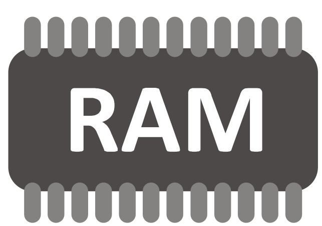
Low:

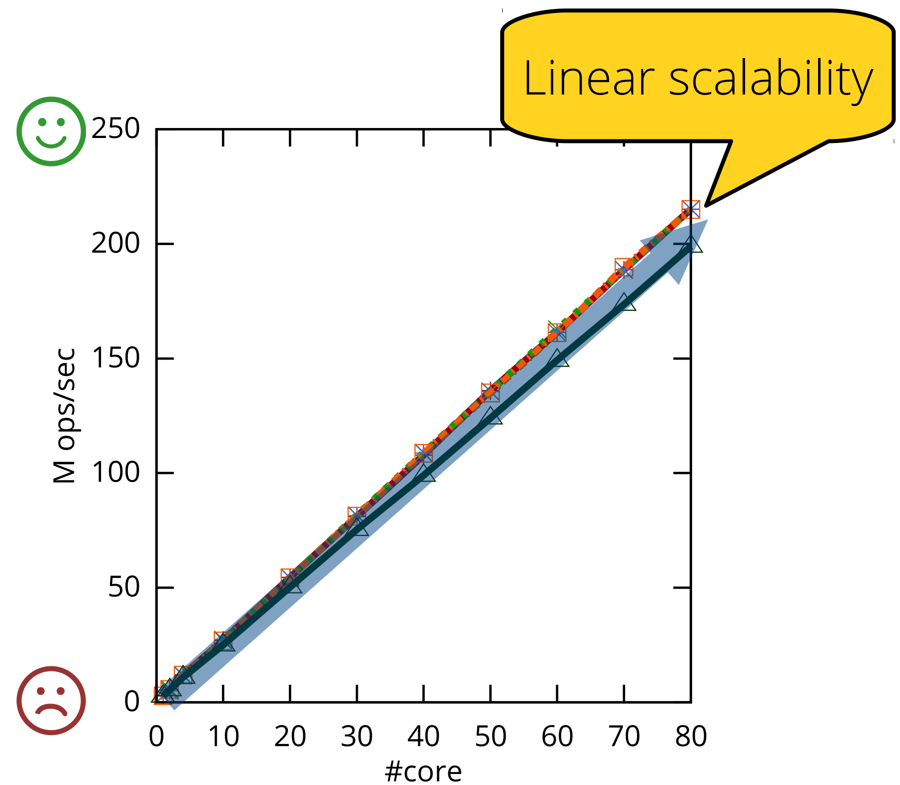


File systems:



Storage medium:

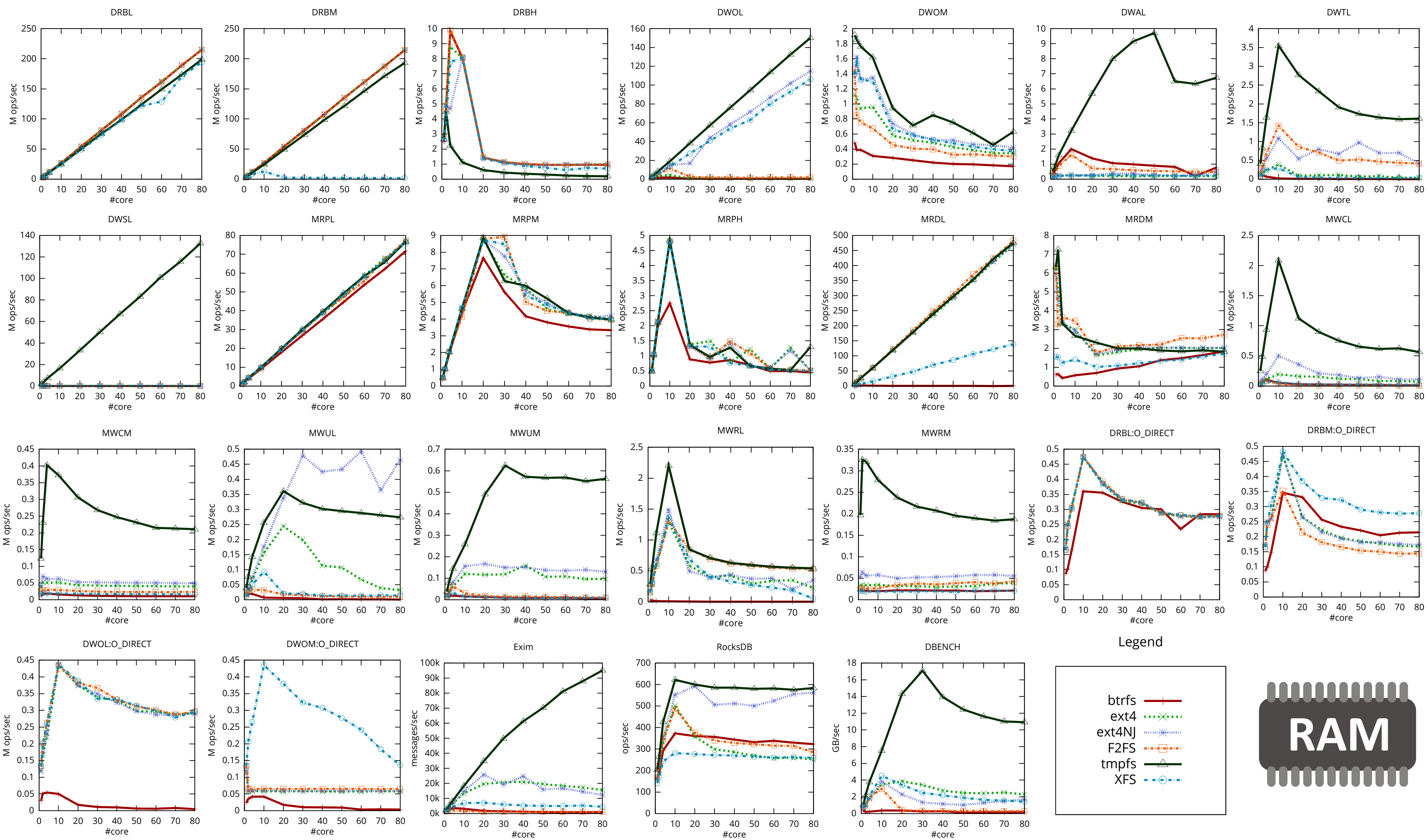




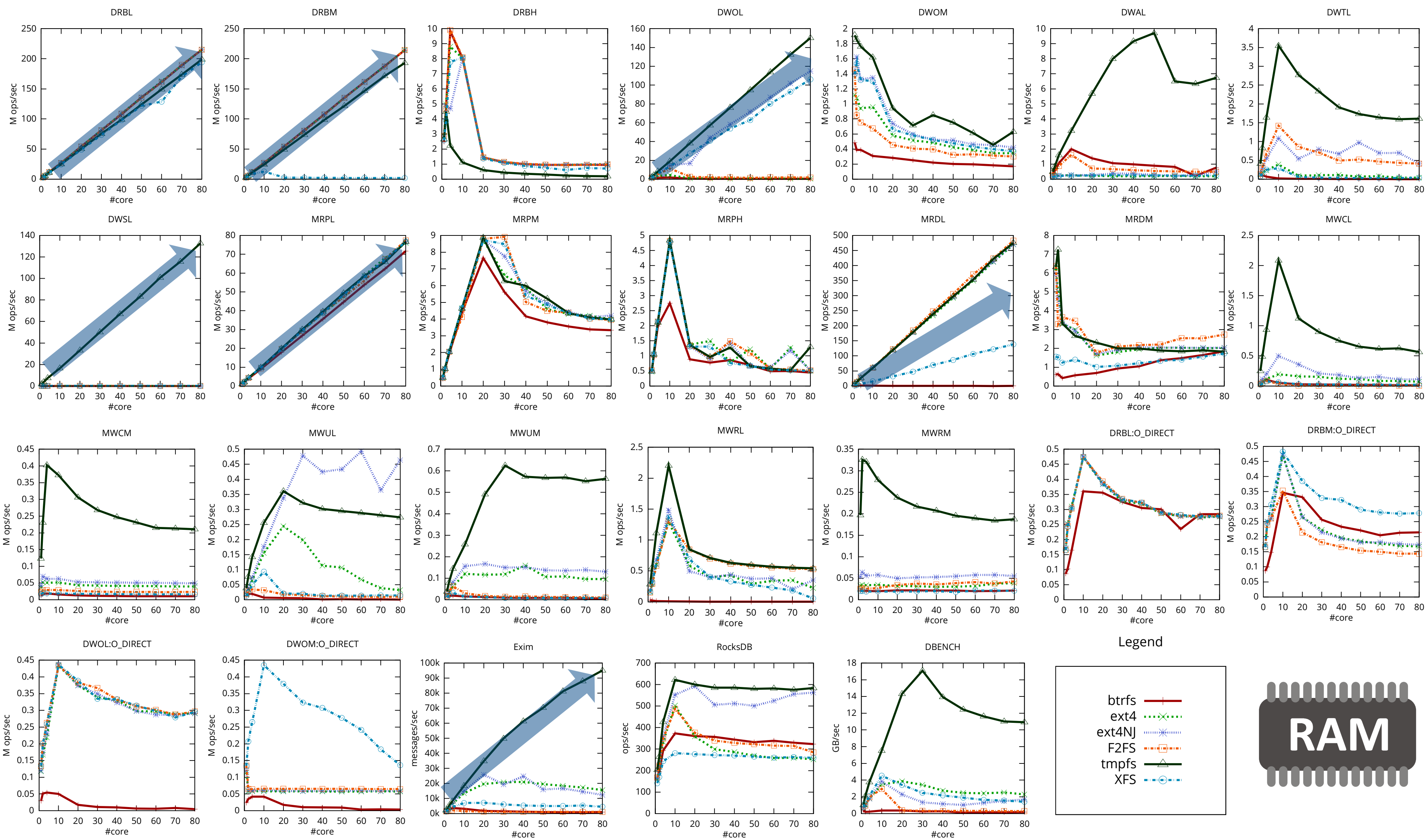
Outline

- Background
- FxMark design
- **Analysis of five Linux file systems**
  - What are scalability bottlenecks?
- Pilot solution
- Related work
- Summary

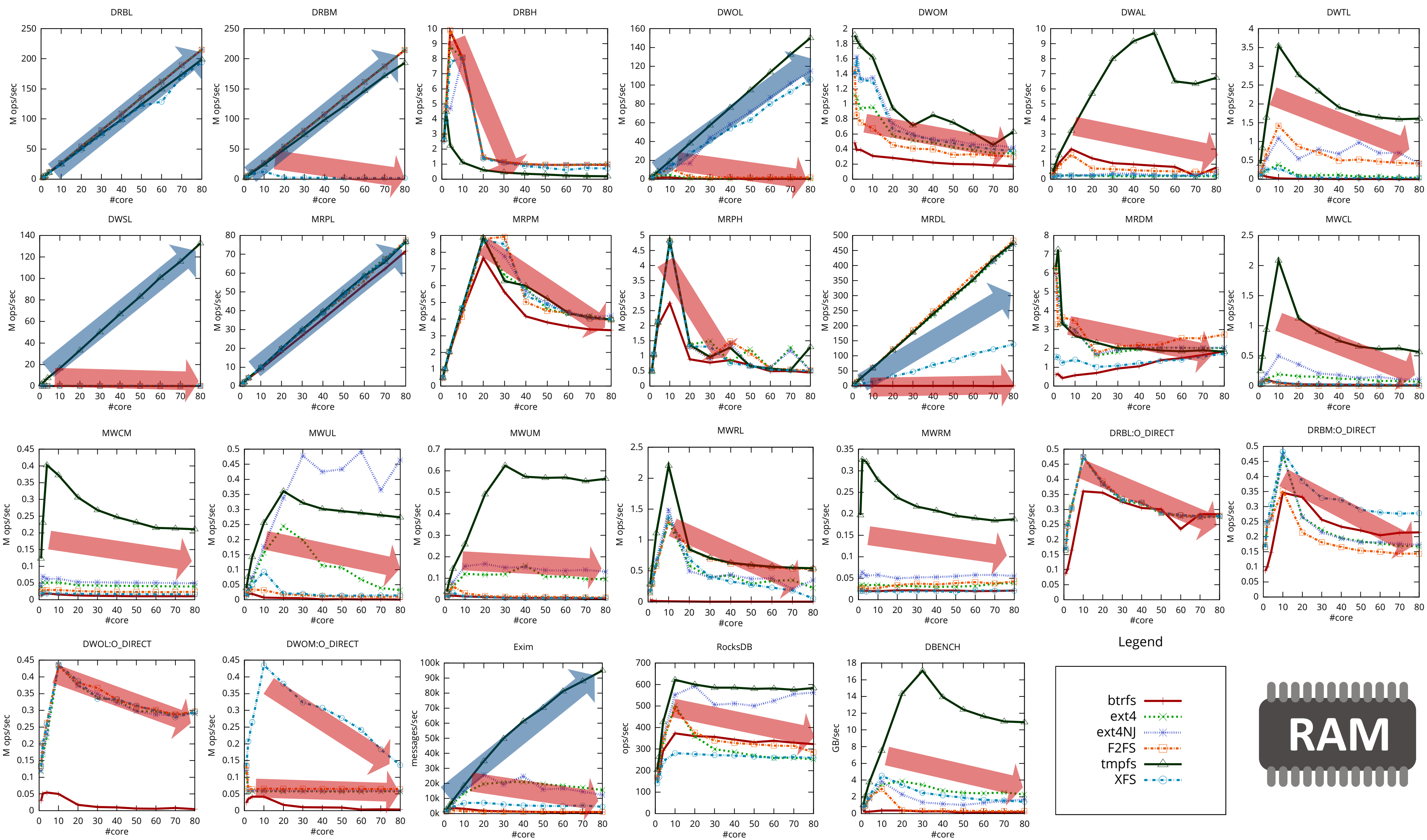
Summary of results: file systems are not scalable



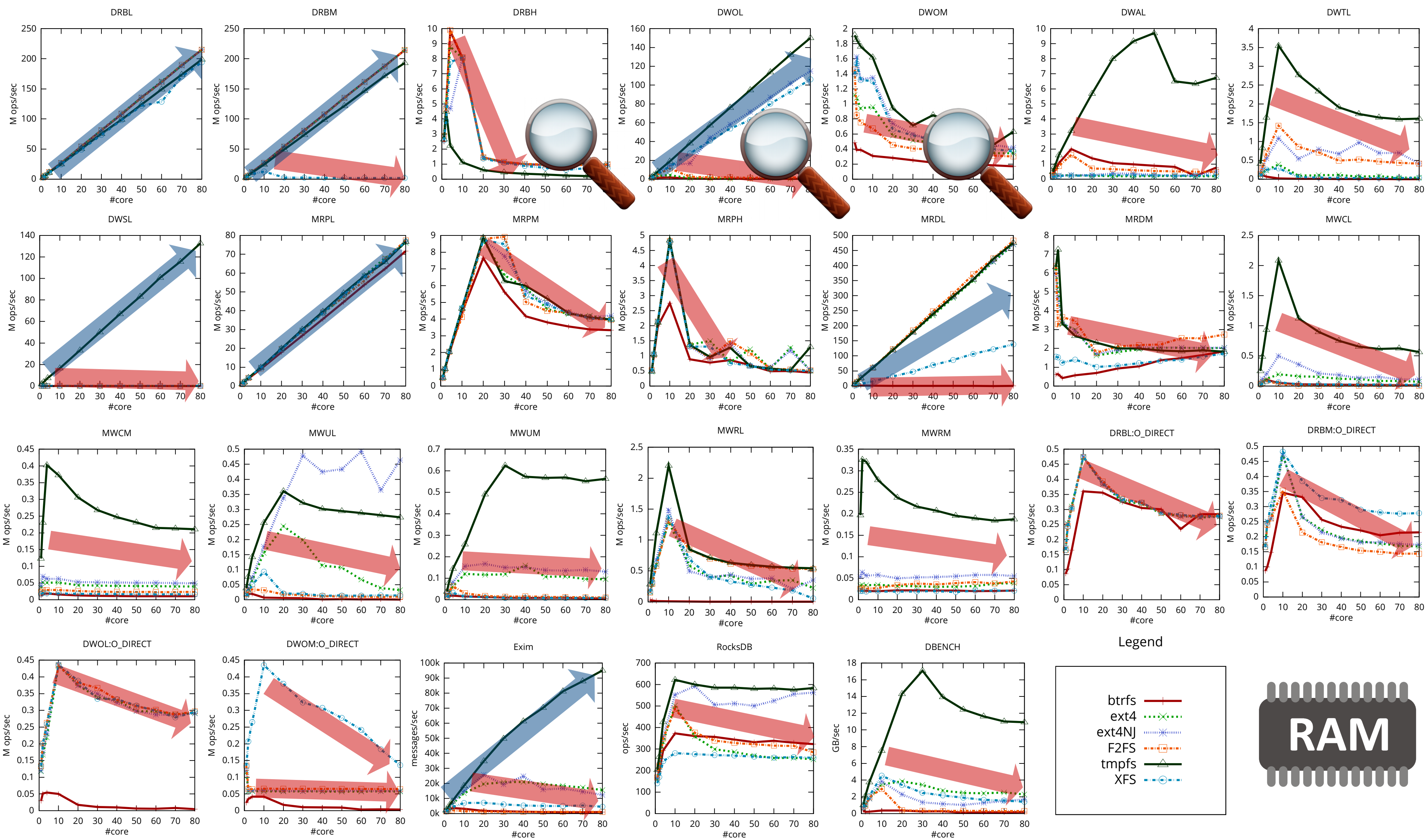
Summary of results: file systems are not scalable



Summary of results: file systems are not scalable

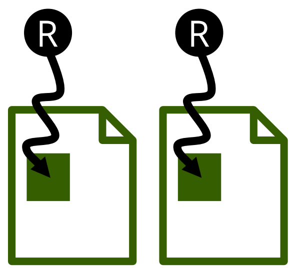


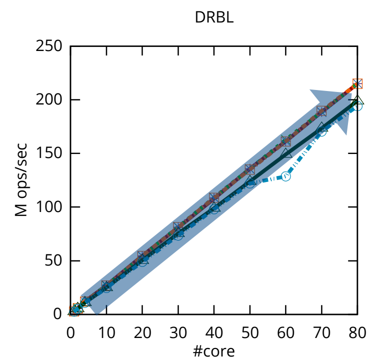
Summary of results:
file systems are not scalable



Data block read

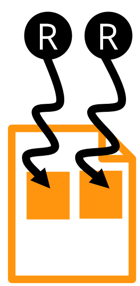
Low:

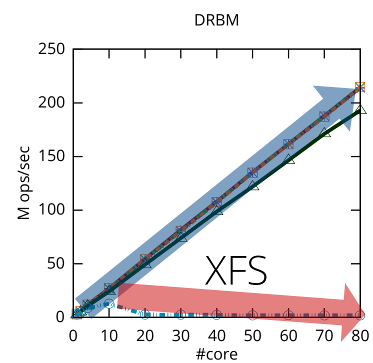




All file systems linearly scale

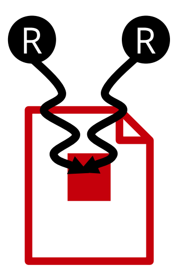
Medium:

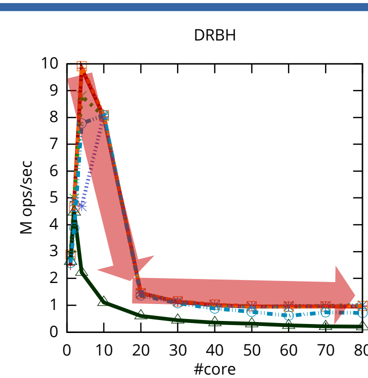




XFS shows performance collapse

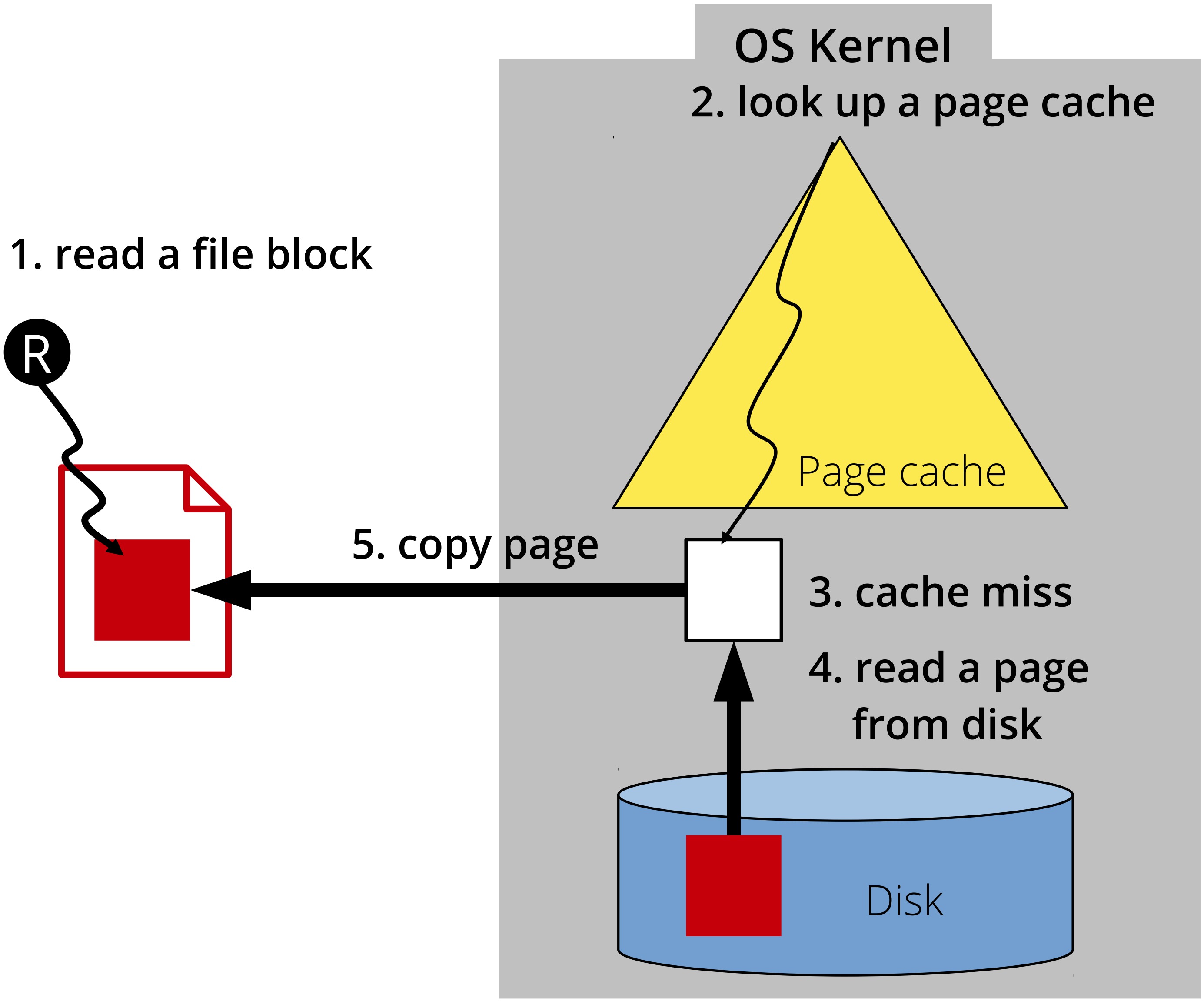
High:



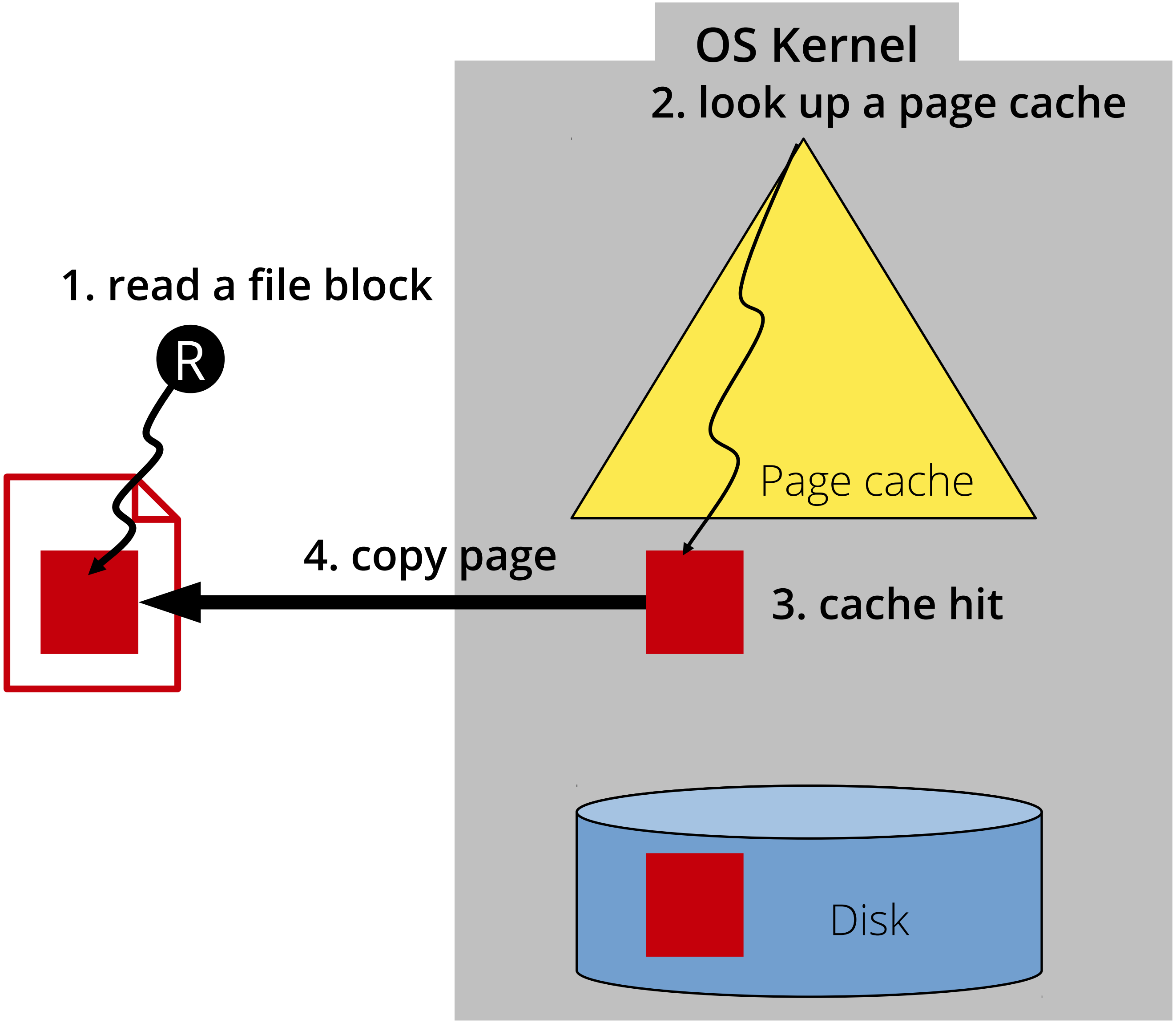


All file systems show performance collapse

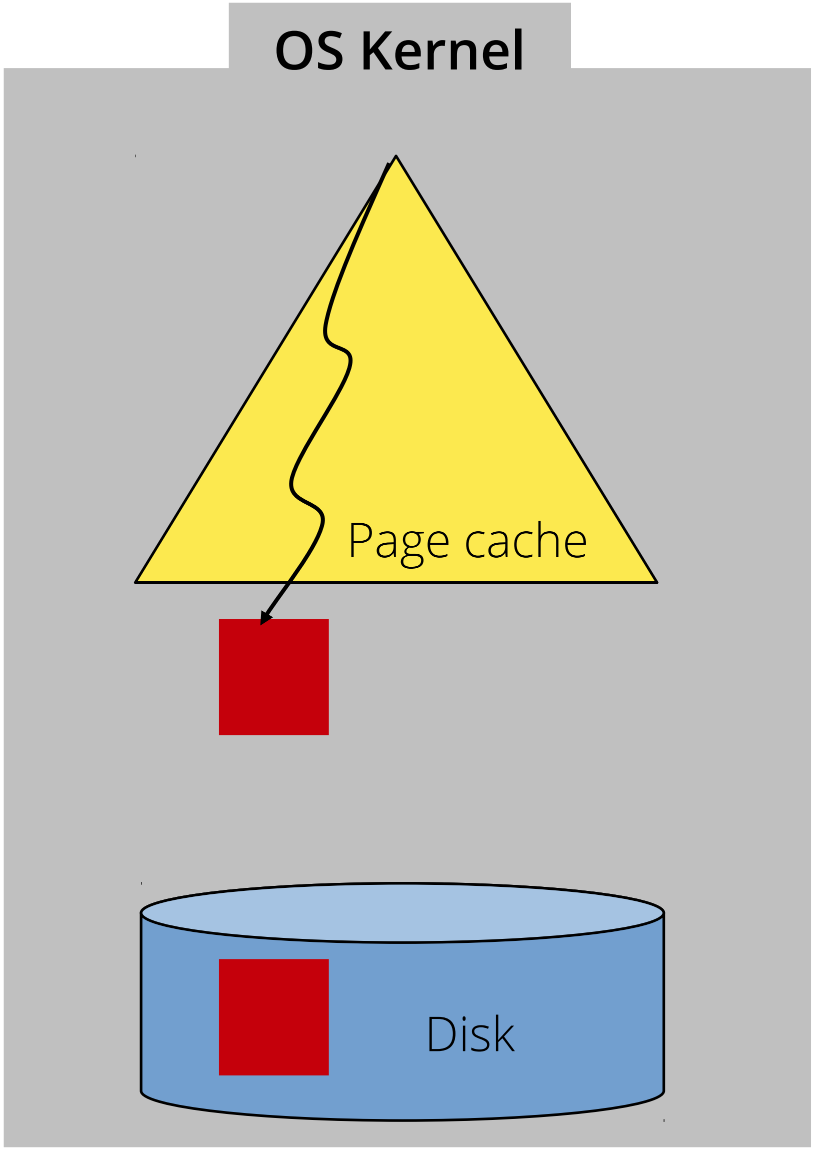
Page cache is maintained for efficient access of file data



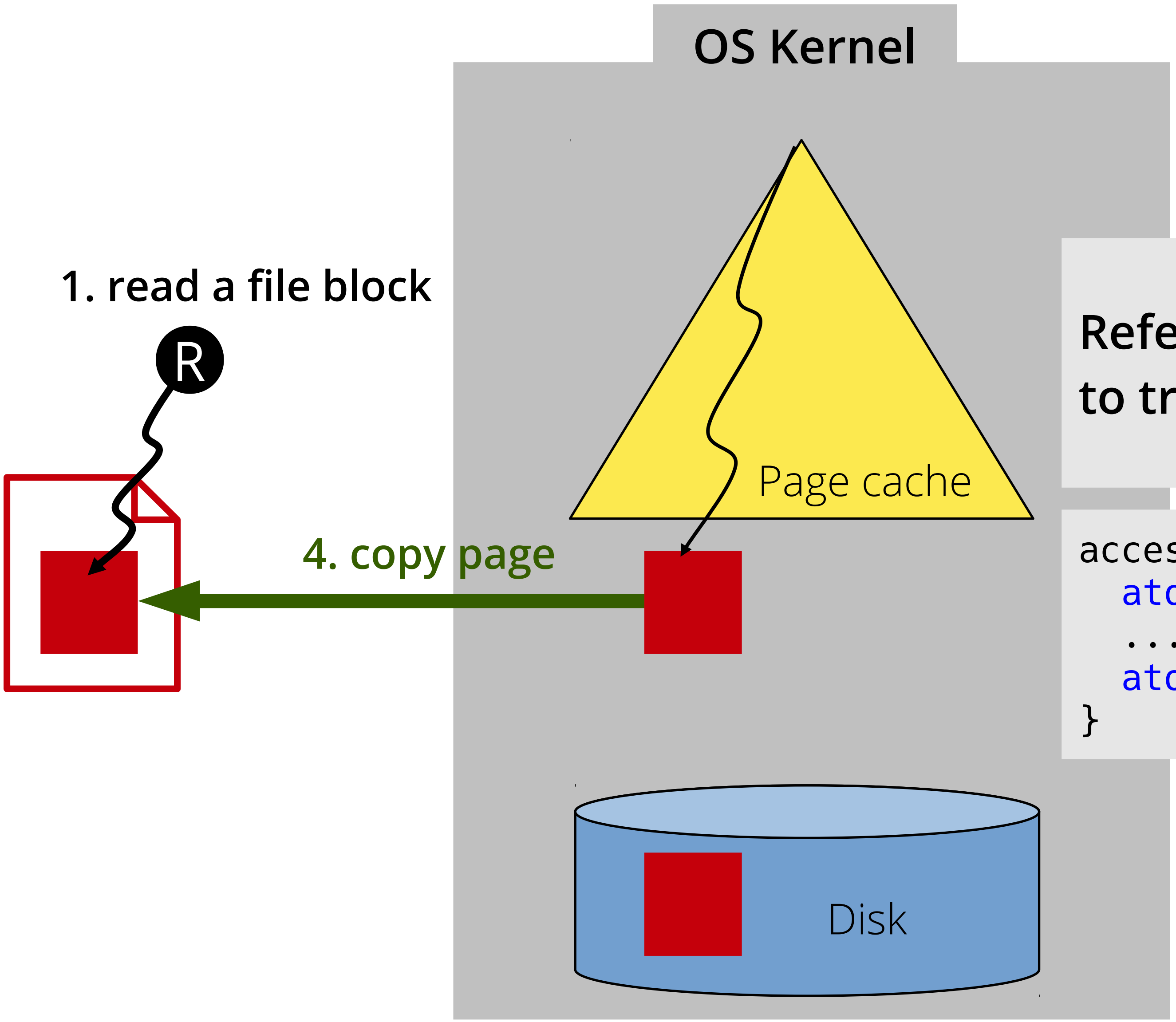
Page cache hit



Page cache can be evicted to secure free memory



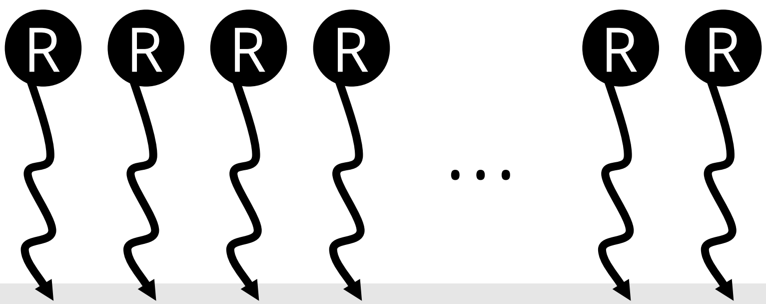
… only when not being accessed



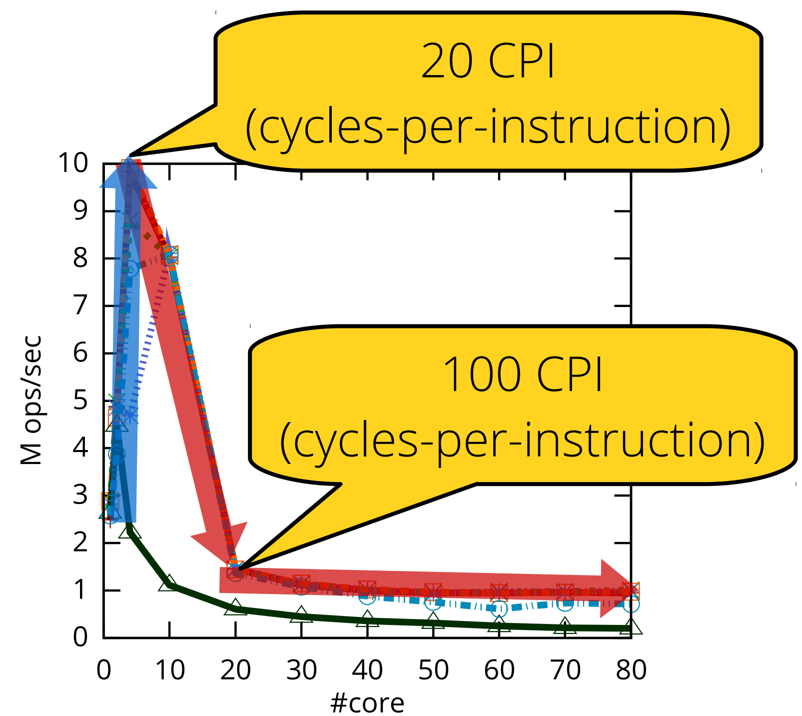
Reference counting is used to track # of accessing tasks

```
access_a_page(...) {
  atomic_inc(&page->_count);
  ...
  atomic_dec(&page->_count);
}
```

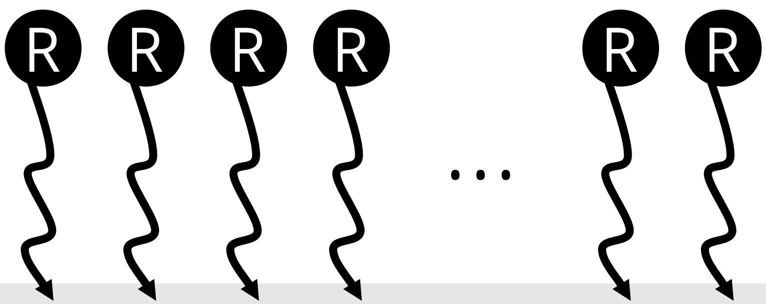
Reference counting becomes a scalability bottleneck



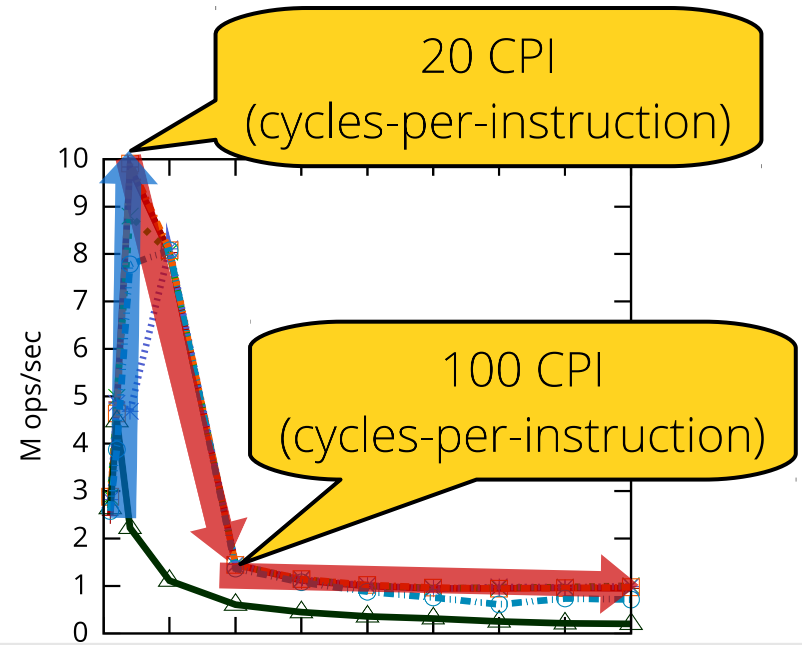

```
access_a_page(...) {
  atomic_inc(&page->_count);
  ...
  atomic_dec(&page->_count);
}
```



Reference counting becomes a scalability bottleneck



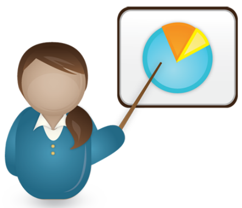

```
access_a_page(...) {
  atomic_inc(&page->_count);
  ...
  atomic_dec(&page->_count);
}
```



High contention on a page reference counter → Huge memory stall

Many more: directory entry cache, XFS inode, etc

Lessons learned



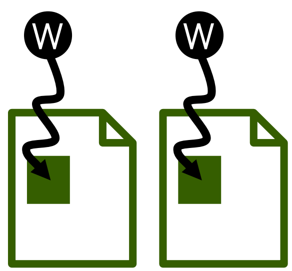
High locality can cause performance collapse

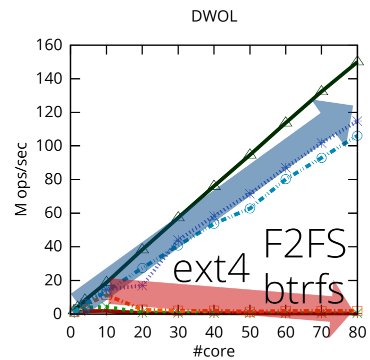
Cache hit should be scalable

→ *When the cache hit is dominant,*
the scalability of cache hit does matter.

Data block overwrite

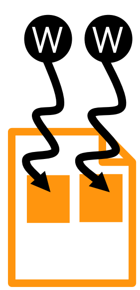
Low:

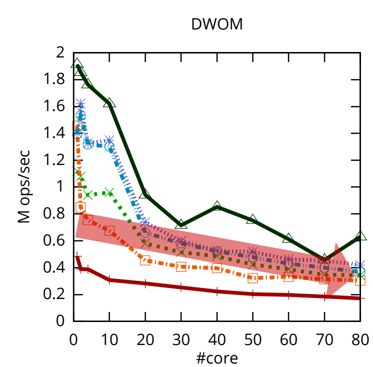




Ext4, F2FS, and btrfs show performance collapse

Medium:





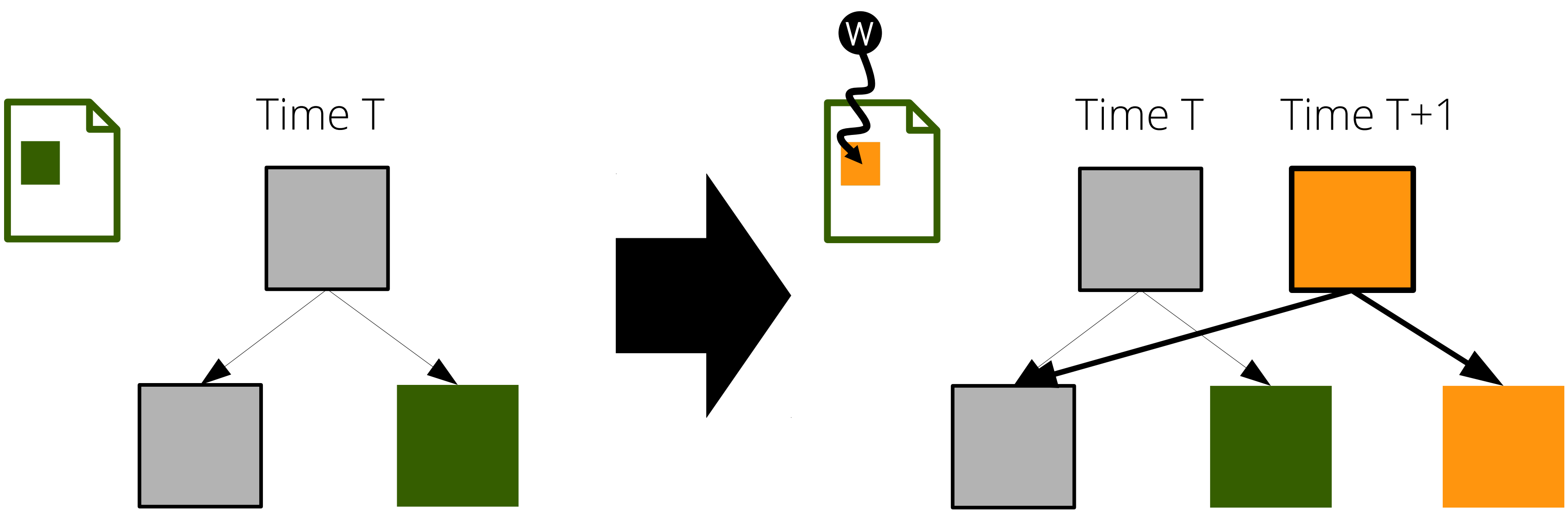
All file systems degrade gradually

Btrfs is a copy-on-write (CoW) file system

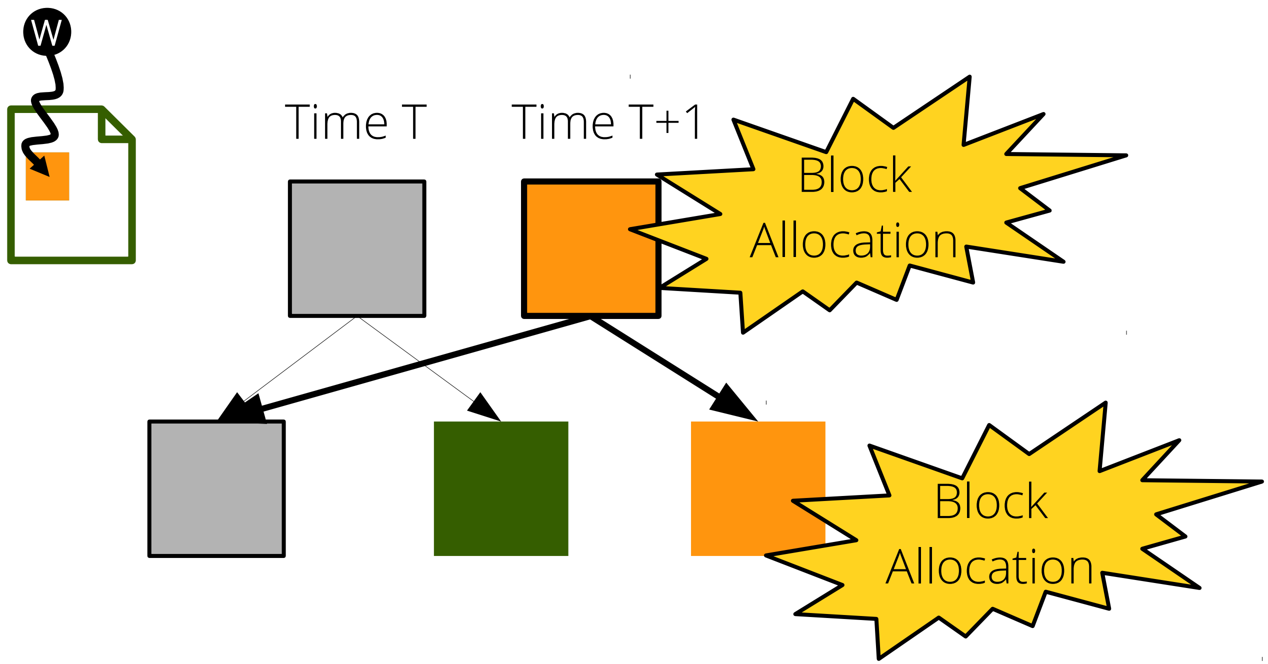
- Directs a write to a block to a new copy of the block

  → Never overwrites the block in place

  → Maintain multiple versions of a file system image



CoW triggers disk block allocation for every write



Disk block allocation becomes a bottleneck

Ext4 → journaling, F2FS → checkpointing

Lessons learned

Overwriting could be as expensive as appending

→ *Critical at log-structured FS (F2FS) and CoW FS (btrfs)*

Consistency guarantee mechanisms should be scalable

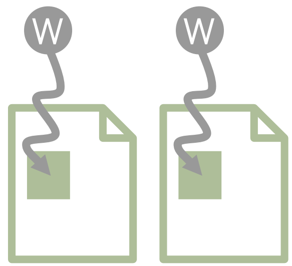
→ *Scalable journaling*

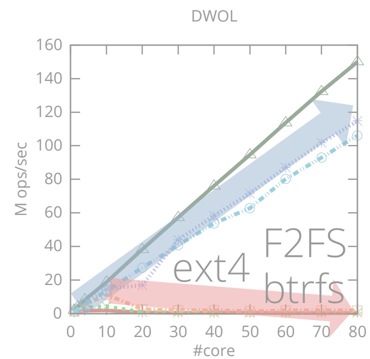
→ *Scalable CoW index structure*

→ *Parallel log-structured writing*

Data block overwrite

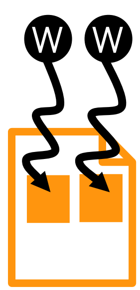
Low:

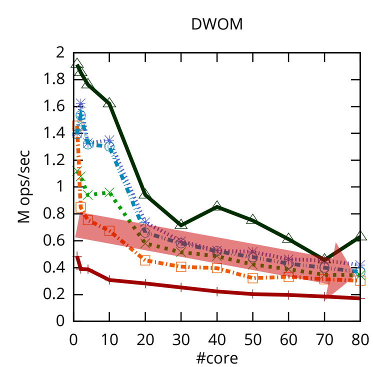




Ext4, F2FS, and btrfs show performance collapse

Medium:





All file systems degrade gradually

Entire file is locked regardless of update range

- All tested file systems hold an inode mutex for write operations
  - Range-based locking is not implemented

```
***_file_write_iter(...) {
  mutex_lock(&inode->i_mutex);
  ...
  mutex_unlock(&inode->i_mutex);
}
```

Lessons learned

A file cannot be concurrently updated

– *Critical for VM and DBMS, which manage large files*

Need to consider techniques used in parallel file systems

→ *E.g., range-based locking*

Summary of findings

- High locality can cause performance collapse
- Overwriting could be as expensive as appending
- A file cannot be concurrently updated
- All directory operations are sequential
- Renaming is system-wide sequential
- Metadata changes are not scalable
- Non-scalability often means wasting CPU cycles
- Scalability is not portable

See our paper

Summary of findings

Many of them are unexpected and counter-intuitive
→ Contention at file system level
to maintain data dependencies

- All directory operations are sequential
- Renaming is system-wide sequential
- Metadata changes are not scalable
- Non-scalability often means wasting CPU cycles
- Scalability is not portable

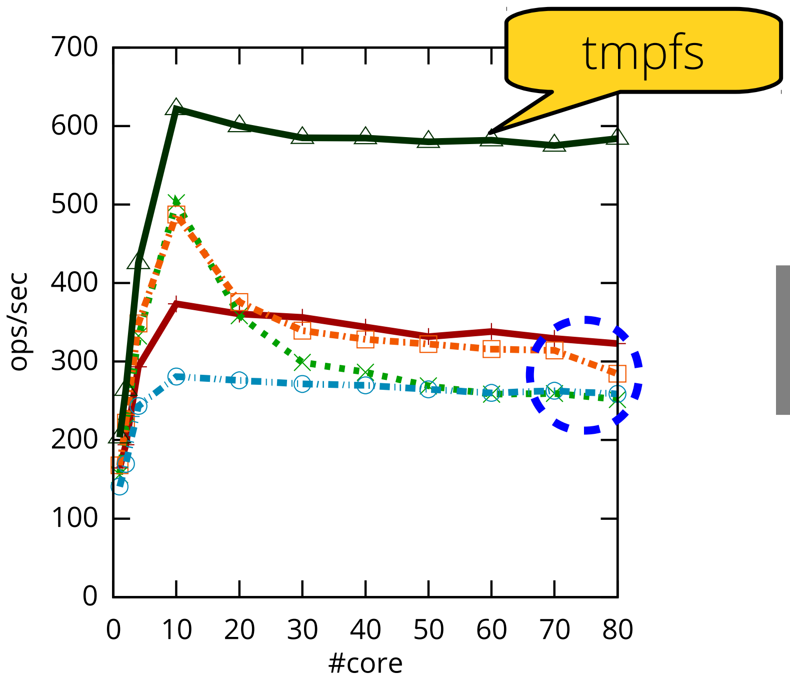
See our paper

Outline

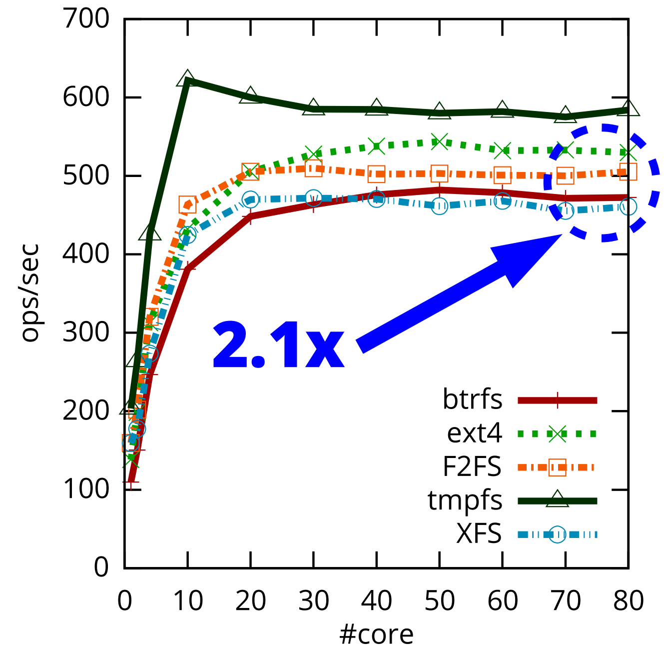
- Background
- FxMark design
- Analysis of five Linux file systems
- **Pilot solution**
  - If we remove contentions in a file system, is such file system scalable?
- Related work
- Summary

RocksDB on a 60-partitioned RAMDISK scales better

A single-partitioned RAMDISK



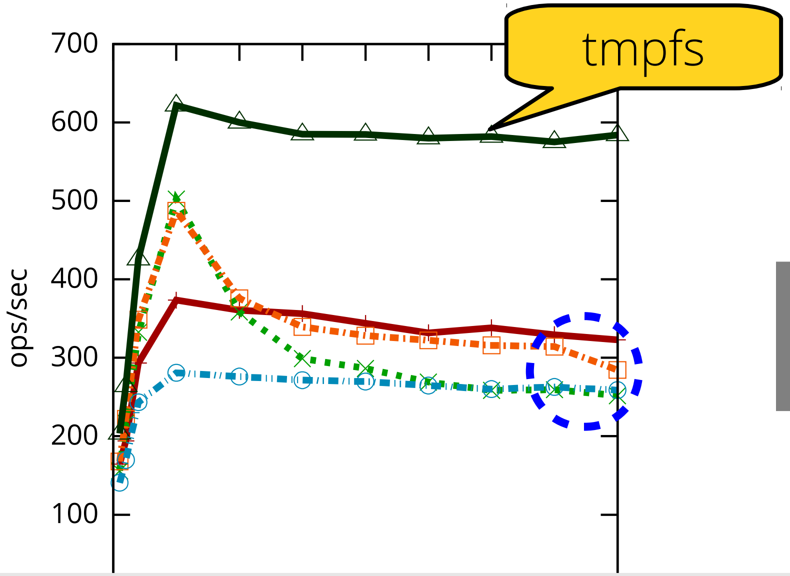
A 60-partitioned RAMDISK



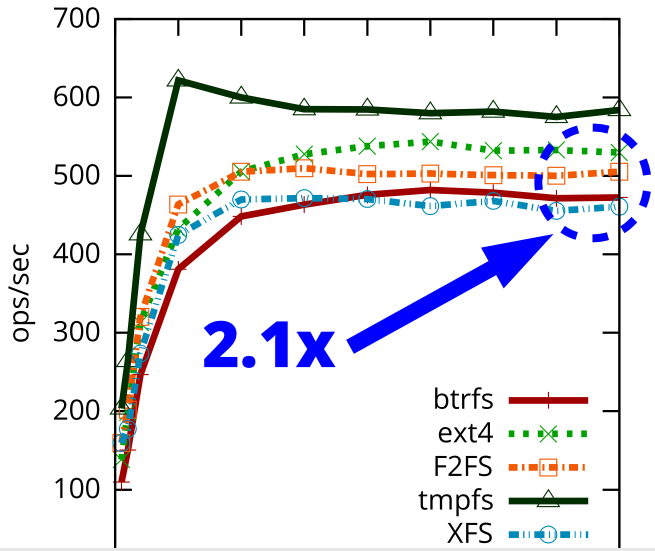
** Tested workload: DB_BENCH overwrite **

RocksDB on a 60-partitioned RAMDISK scales better

A single-partitioned RAMDISK



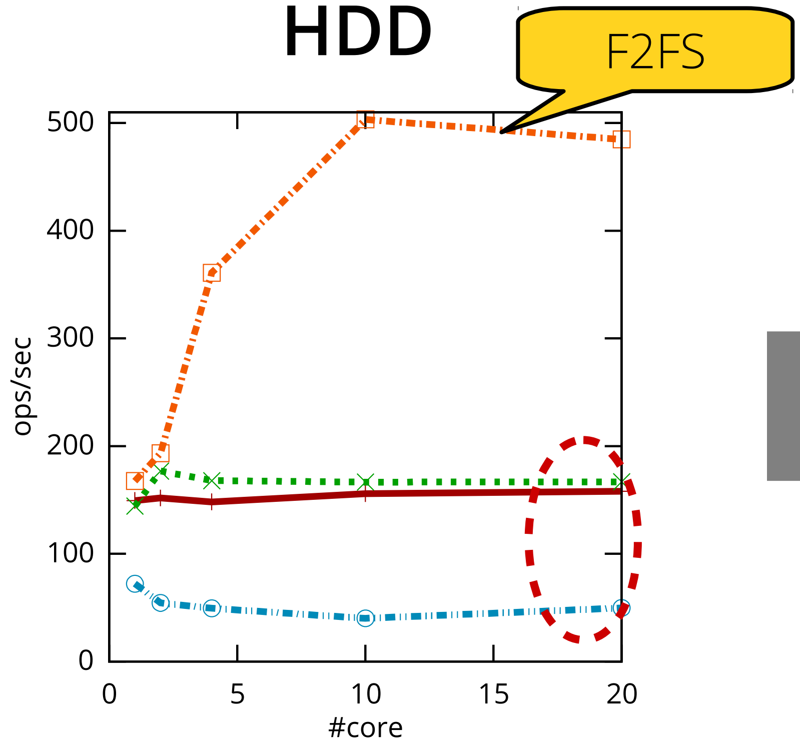
A 60-partitioned RAMDISK



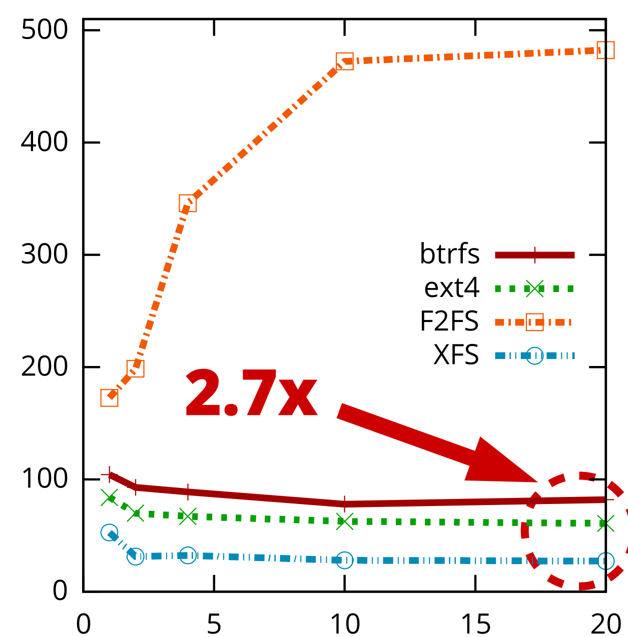
Reduced contention on file systems helps improving performance and scalability

But partitioning makes performance worse on HDD

A single-partitioned HDD



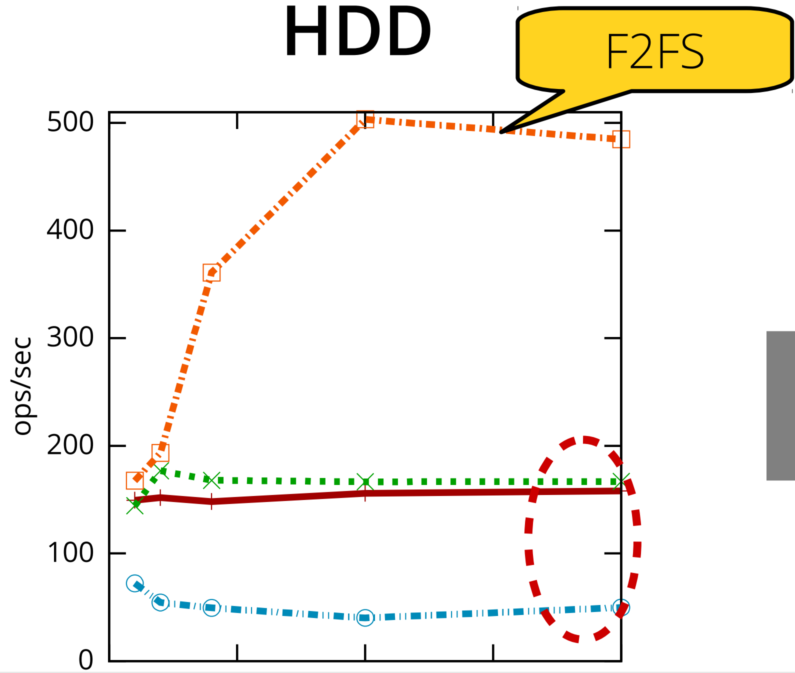
A 60-partitioned HDD



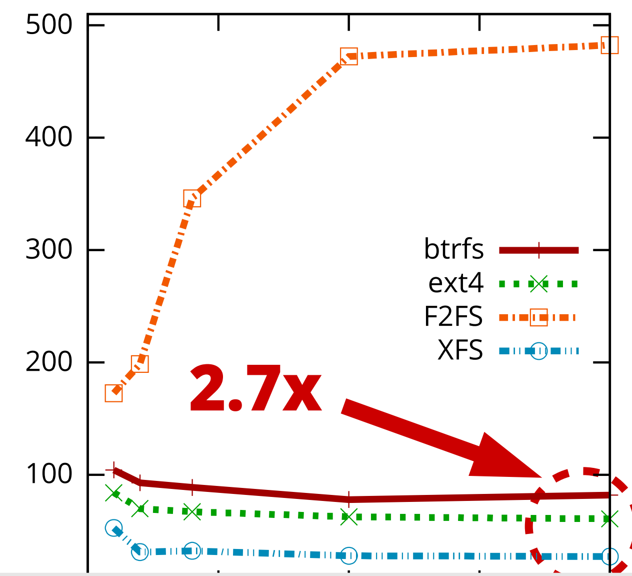
** Tested workload: DB_BENCH overwrite **

But partitioning makes performance worse on HDD

A single-partitioned HDD



A 60-partitioned HDD



But reduced spatial locality degrades performance → Medium-specific characteristics (e.g., spatial locality) should be considered

Related work

- Scaling operating systems
  - Mostly use memory file system to opt out the effect of I/O operations
- Scaling file systems
  - Scalable file system journaling
    - ScaleFS [MIT:MSThesis'14]
    - SpanFS [ATC'15]
  - Parallel log-structured writing on NVRAM
    - NOVA [FAST'16]

Summary

- Comprehensive analysis of manycore scalability of five widely-used file systems using FxMark
- Manycore scalability should be of utmost importance in file system design
- New challenges in scalable file system design
  - Minimizing contention, scalable consistency guarantee, spatial locality, etc.
- FxMark is open source
  - ***https://github.com/sslab-gatech/fxmark***