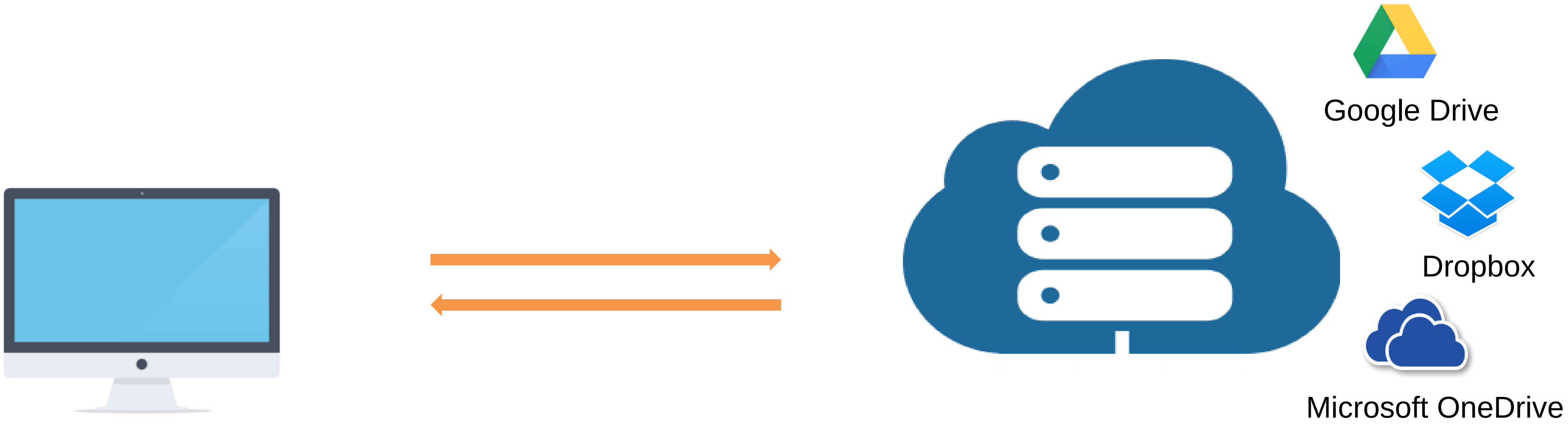
Knockoff: Cheap versions in the cloud

Xianzheng Dou, Peter M. Chen, Jason Flinn

UNIVERSITY OF MICHIGAN

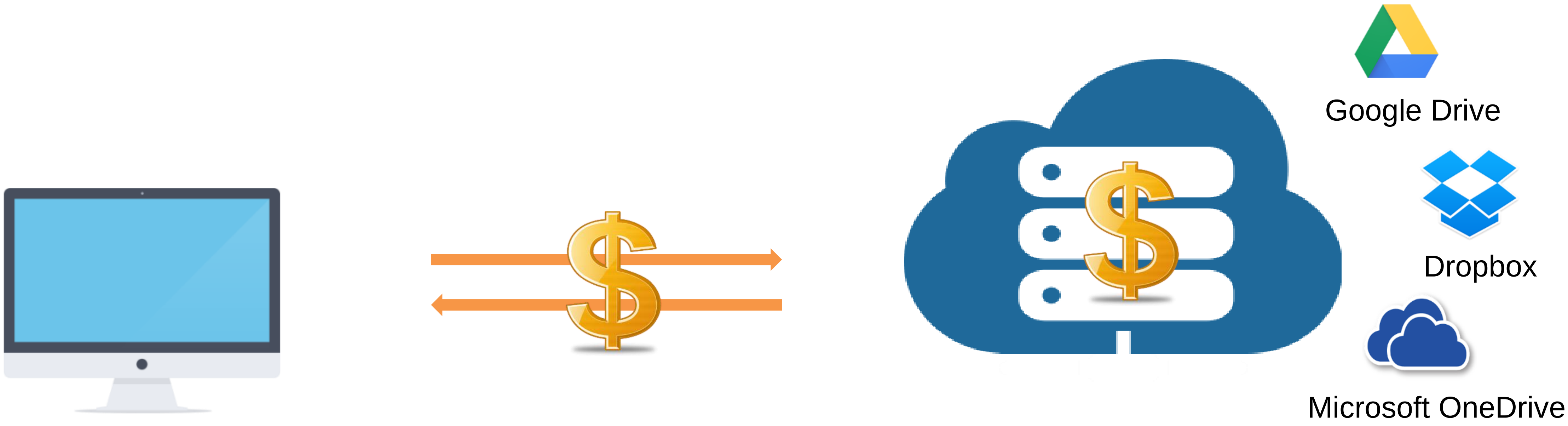
Cloud-based storage



Pros:

- Ease-of-management
- Reliability

Cloud-based storage



Challenges:

- Storage costs
- Communication costs

Versioning increases costs

Google Drive

Dropbox

Microsoft OneDrive

Pros:

- Recovery of lost data
- Auditing
- Troubleshooting

Versioning

Reducing costs: a new direction

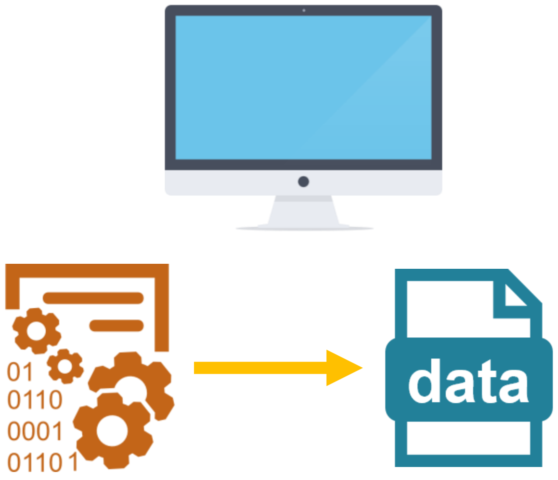
- Established methods exploit similarities in data
  - Chunk-based deduplication
  - Delta compression
  - Greater work for incremental gains

- Our goal: explore an orthogonal new dimension
  - Deterministically recompute data in lieu of communication, storage

File: data or computation?

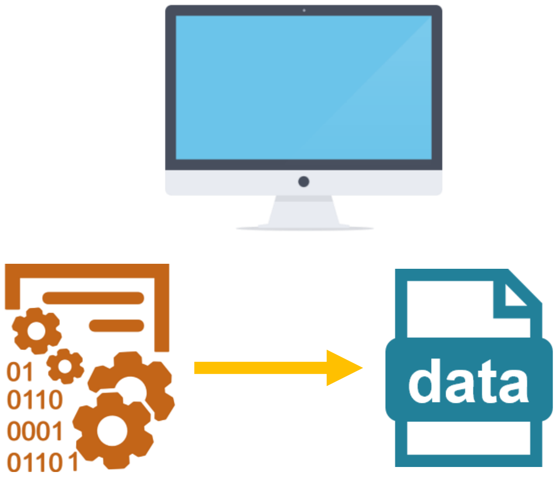
Computation File data

File: data or computation?



Computation File data

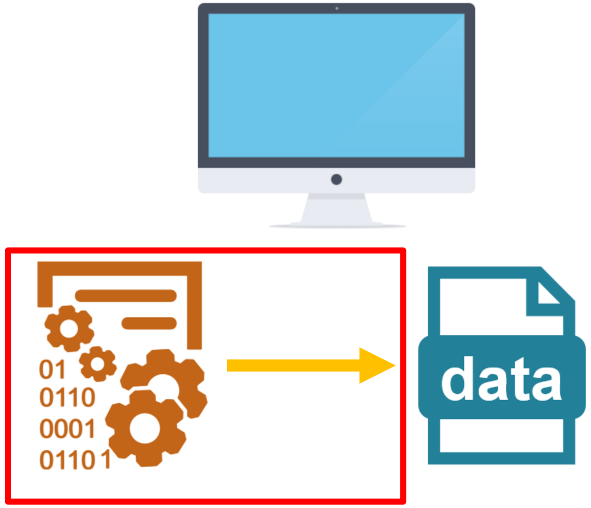
File: data or computation?



Computation

File data

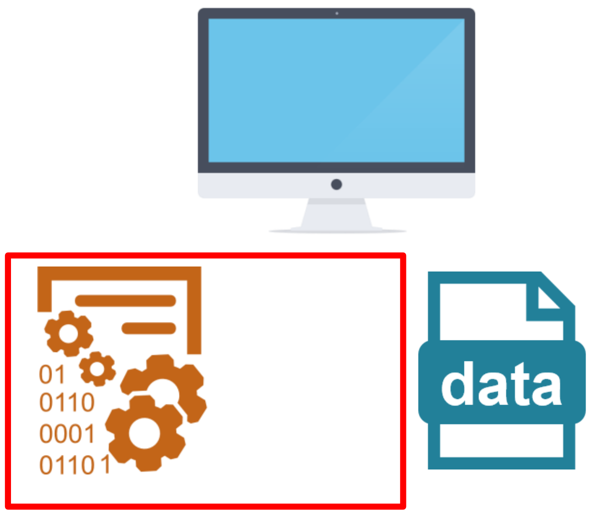
File: data or computation?



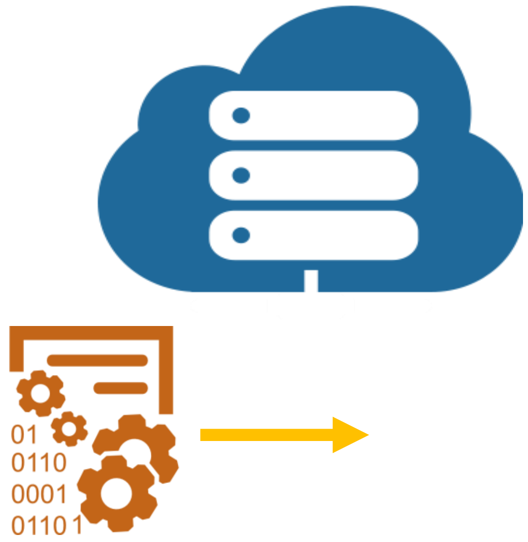
Computation

File data

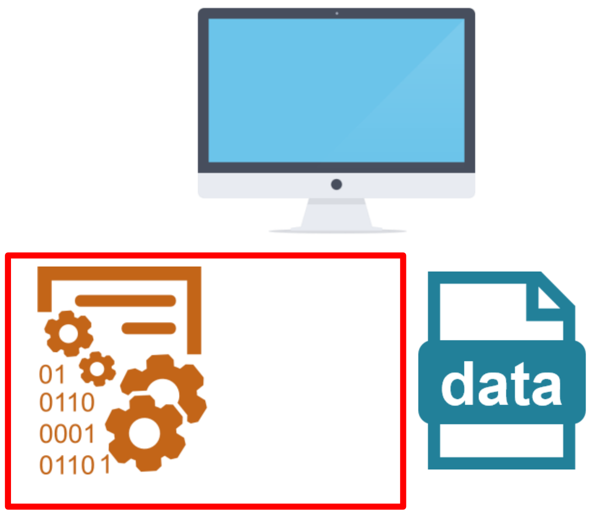
File: data or computation?



Computation File data

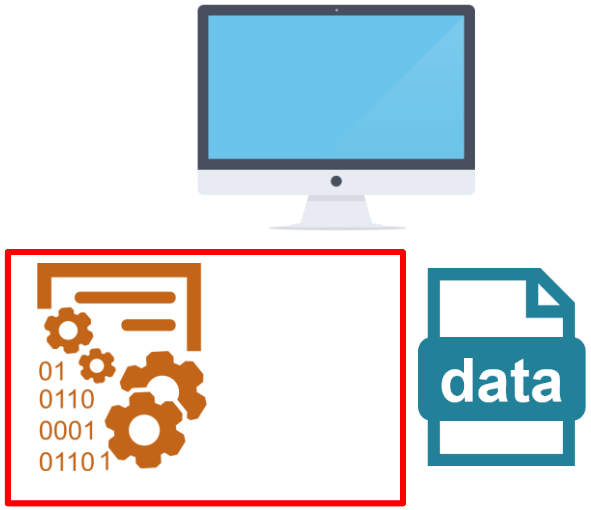


File: data or computation?



Computation File data

File: data or computation?



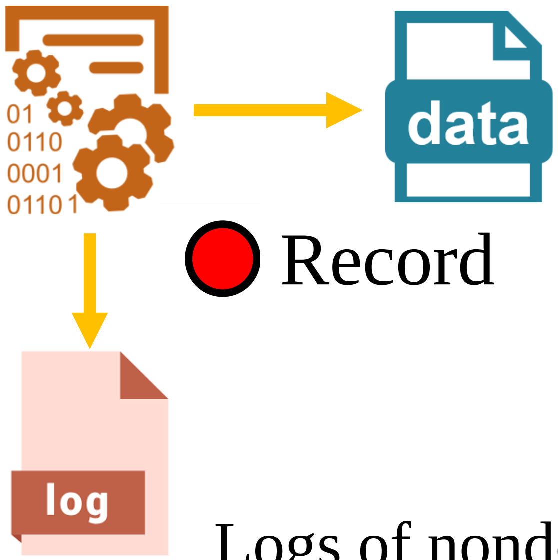
Computation File data

Different output data

How can we address non-determinism?

File: data or computation?

- Deterministic record and replay

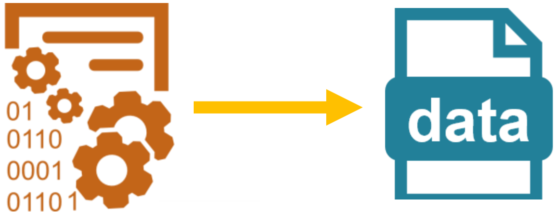


Record

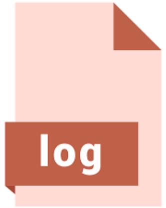
Logs of nondeterminism

File: data or computation?

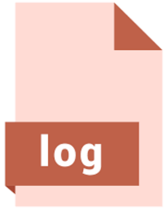
- Deterministic record and replay



Record

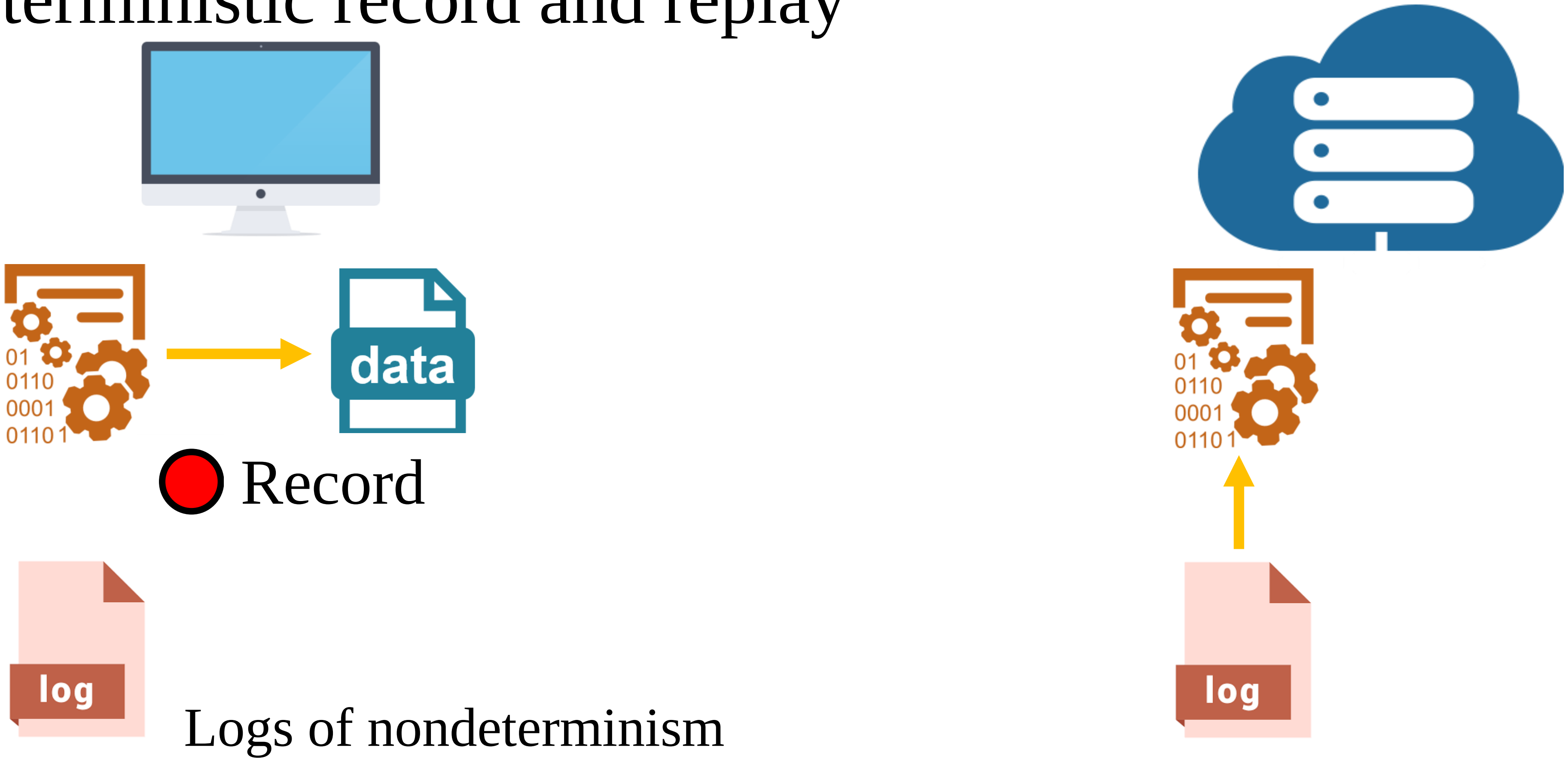


Logs of nondeterminism



File: data or computation?

- Deterministic record and replay



File: data or computation?

- Deterministic record and replay

Knockoff

- Selectively substitutes computation for data
- Benefits
  - Reduction compared to chunk-based deduplication
    - Communication costs: 21%
    - Storage costs: 19%
  - Benefits increases as we retain versions more frequently
  - A new fined-grained versioning policy

Outline

- Introduction
- Writing files
- Storing files
- Evaluation

Knockoff

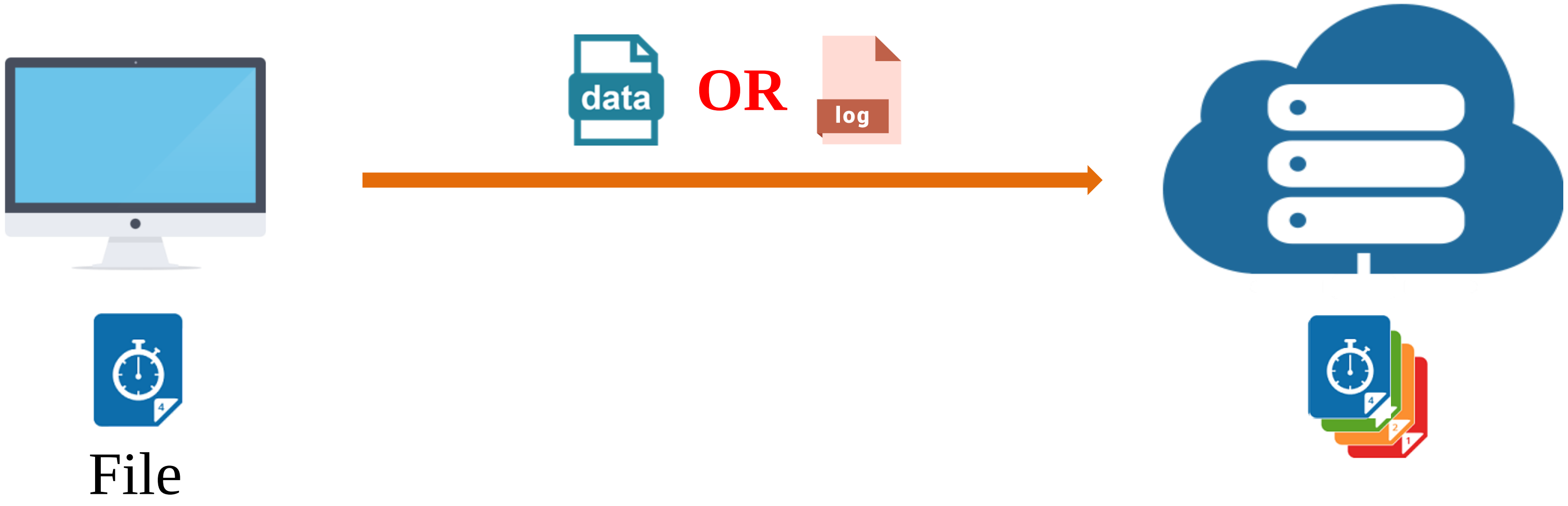
- Knockoff selectively represents a file as:
  - ❑ Normal file data (by value)
  - ❑ Logs of the nondeterminism needed to recompute the file (by operation)

Knockoff

- Knockoff selectively represents a file as:
  - ❑ Normal file data (by value)
  - ❑ Logs of the nondeterminism needed to recompute the file (by operation)

Knockoff

- Knockoff selectively represents a file as:
  - ❑ Normal file data (by value)
  - ❑ Logs of the nondeterminism needed to recompute the file (by operation)

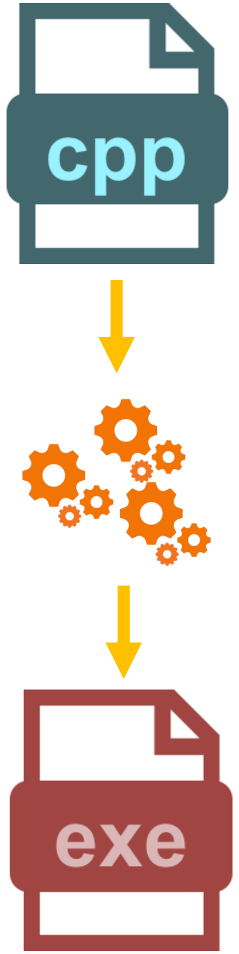


Knockoff

- Knockoff selectively represents a file as:
  - ❑ Normal file data (by value)
  - ❑ Logs of the nondeterminism needed to recompute the file (by operation)

An example log for compilation

	Log entry	Values
1	*open*	rc=3
2	*mmap*	rc=<addr>,file=<id,version>
3	*gettimeofday*	rc=0,time=<time>
4	*pthread_lock*	rc=0
5	*SIGCHILD*	
	…	…



An example log for compilation

	Log entry	Values
1	*open*	rc=3
2	*mmap*	rc=<addr>,file=<id,version>
3	*gettimeofday*	rc=0,time=<time>
4	*pthread_lock*	rc=0
5	*SIGCHILD*	
	…	…

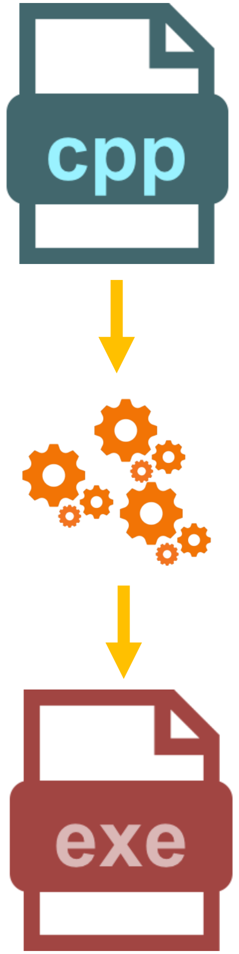
Return values from syscalls

Ordering of thread synchronization

Signals

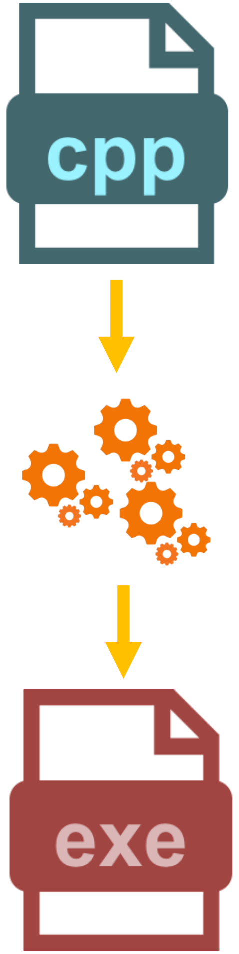
An example log for compilation

	Log entry	Values
1	*open*	rc=3
2	*mmap*	rc=<addr>,file=<id,version>
3	*gettimeofday*	rc=0,time=<time>
4	*pthread_lock*	rc=0
5	*SIGCHILD*	
	…	…

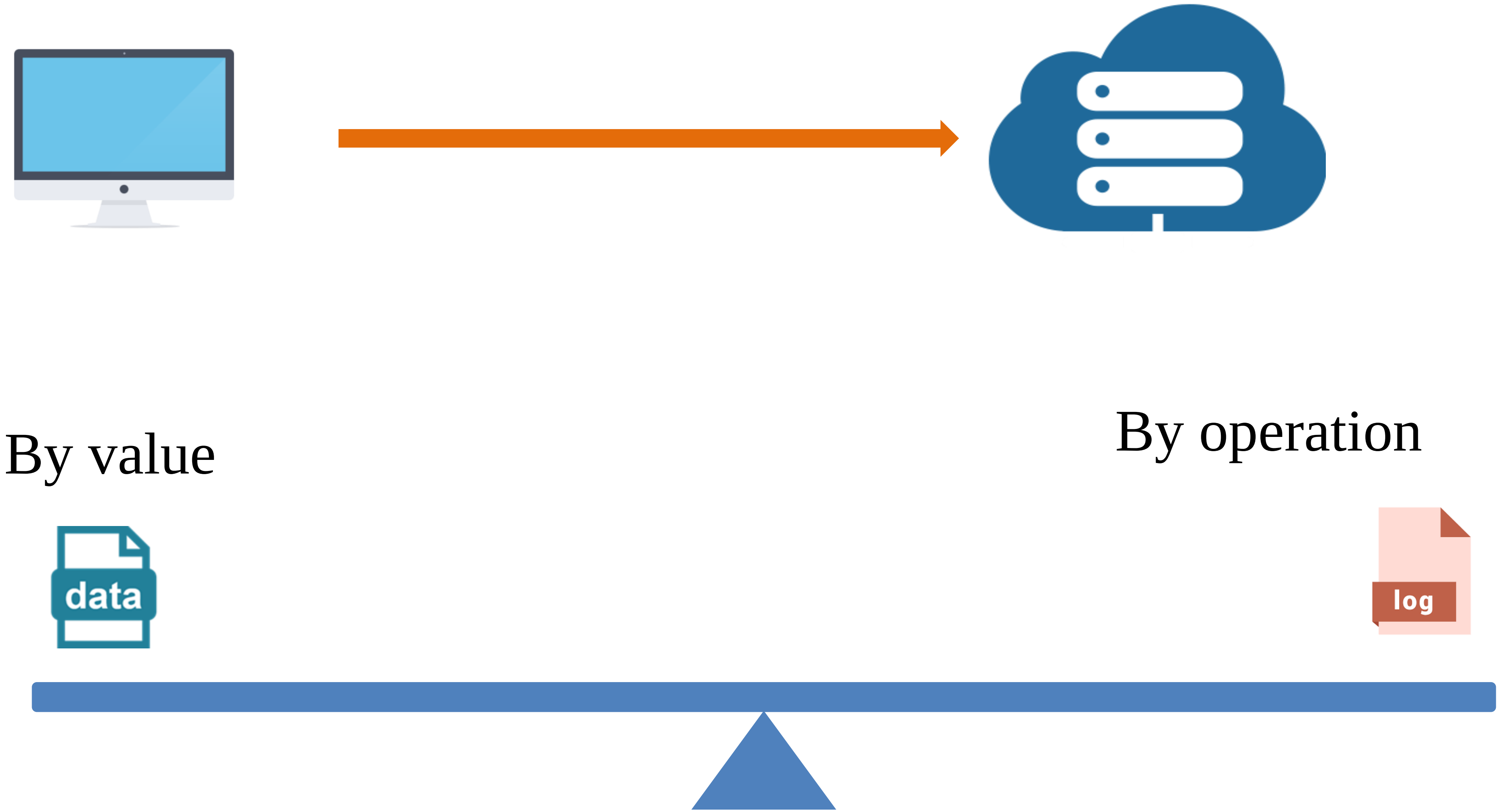


An example log for compilation

	Log entry	Values
1	*open*	rc=3
2	*mmap*	rc=<addr>,file=<id,version>
3	*gettimeofday*	rc=0,time=<time>
4	*pthread_lock*	rc=0
5	*SIGCHILD*	
	…	…



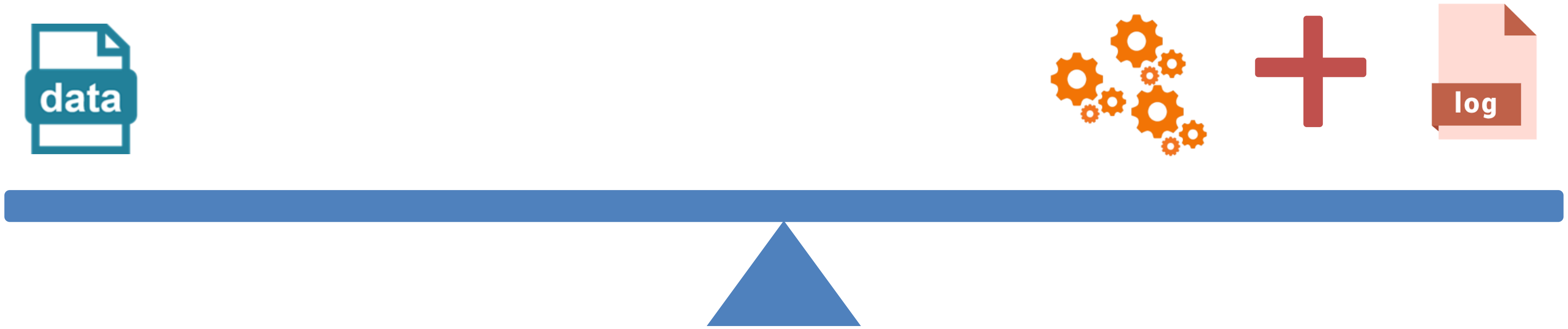
Writing files



Writing files

By value

By operation

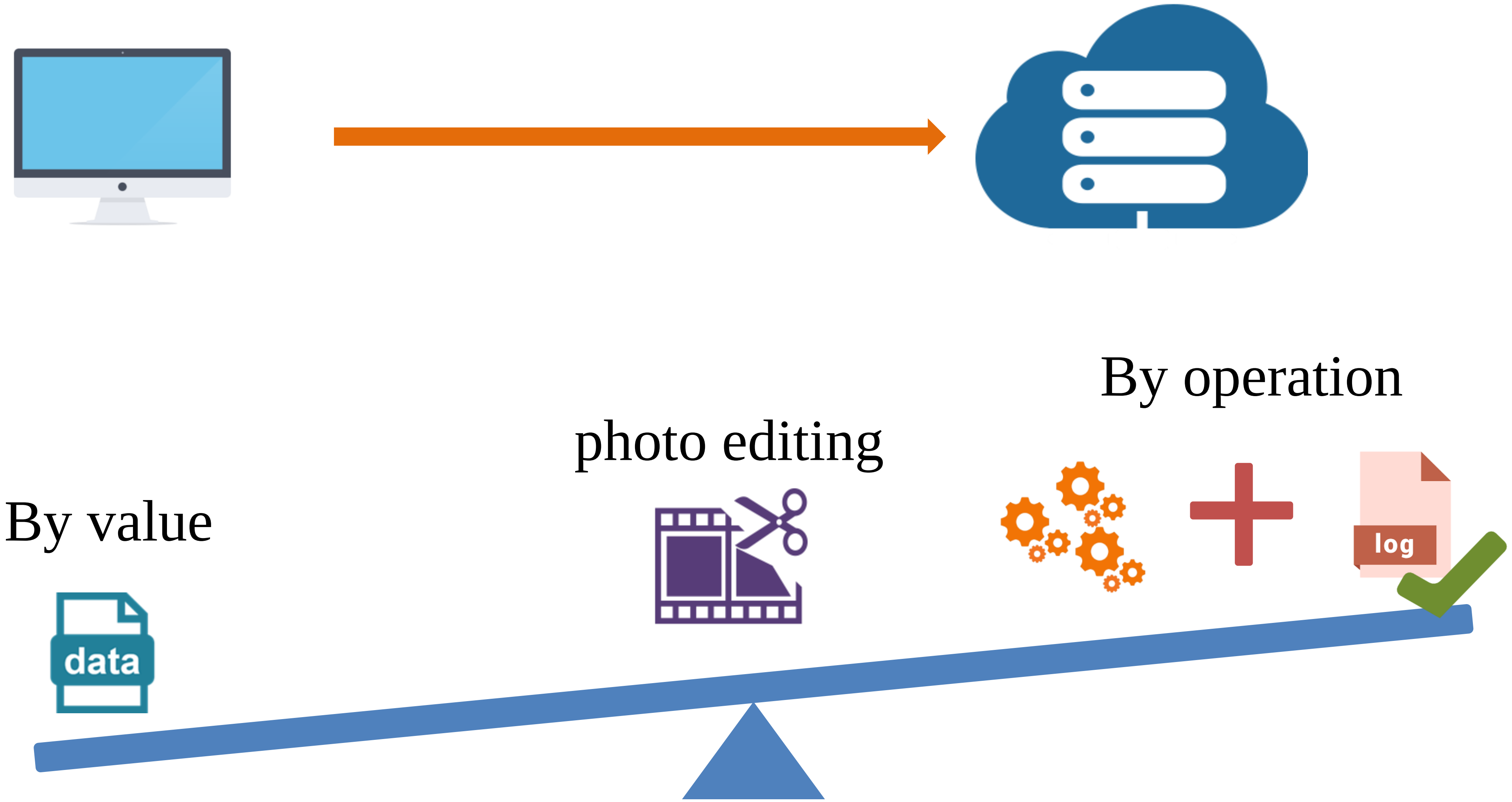


Writing files

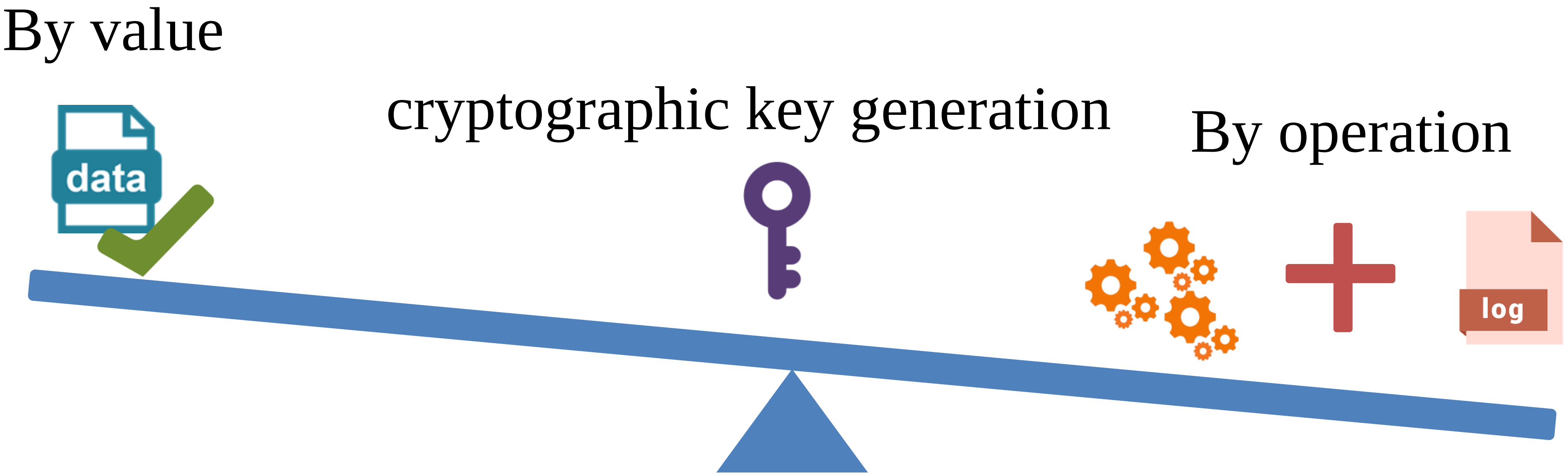
By value

By operation

Writing files



Writing files



Outline

- Introduction
- Writing files
- Storing files
- Evaluation

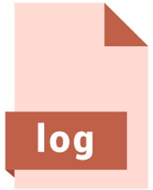
Storing files

- Store files by value or by operation?

- A tradeoff between latency and costs
  – Current versions: by value
  – Past versions: by value or by operation

Storing past versions

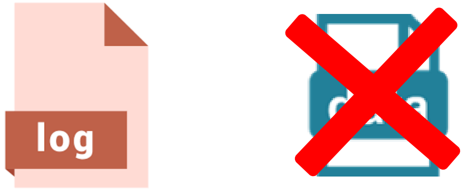
- Maximum materialization delay
  - Time bound for reconstructing any version



Regeneration time = **20s** < Materialization delay = **60s**

Storing past versions

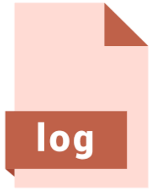
- Maximum materialization delay
  - Time bound for reconstructing any version



Regeneration time = **20s** < Materialization delay = **60s**

Storing past versions

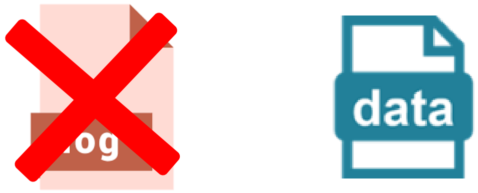
- Maximum materialization delay
  - Time bound for reconstructing any version



Regeneration time = **100s** > Materialization delay = **60s**

Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version



Regeneration time = **100s** > Materialization delay = **60s**

Storing past versions

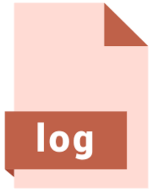
- Maximum materialization delay
  – Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = **20s**

<

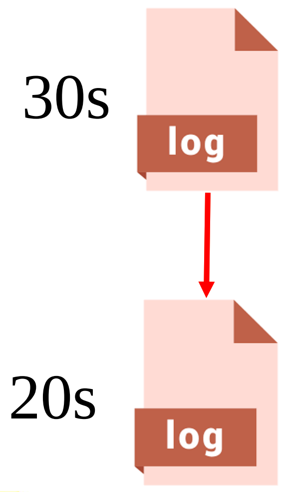
Materialization delay = **60s**

20s



Storing past versions

- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

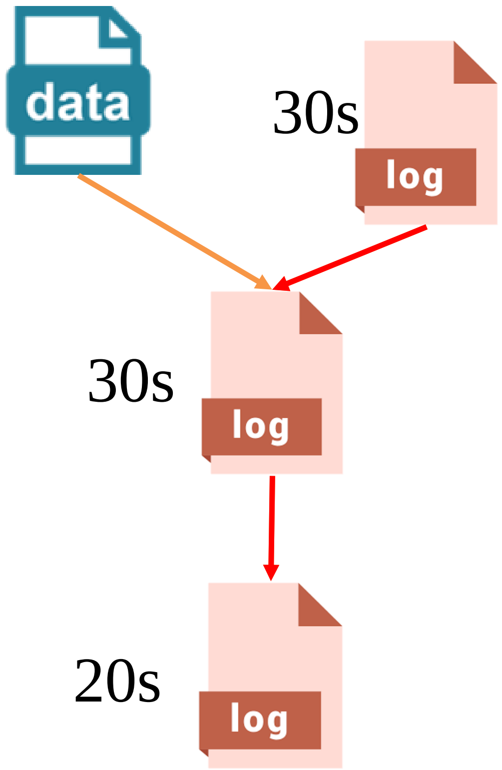


Total regeneration time = **50s**

$<$

Materialization delay = **60s**

Storing past versions



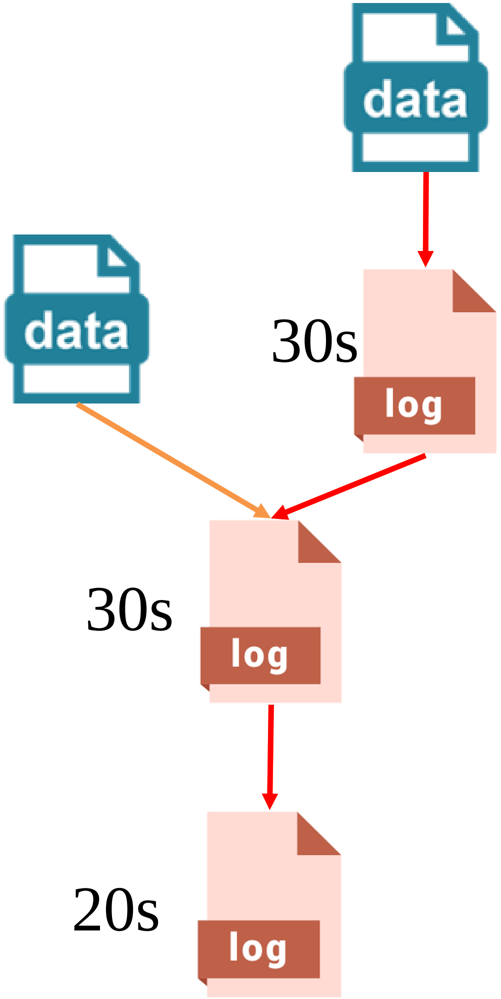
- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = **80s**

>

Materialization delay = **60**s

Storing past versions



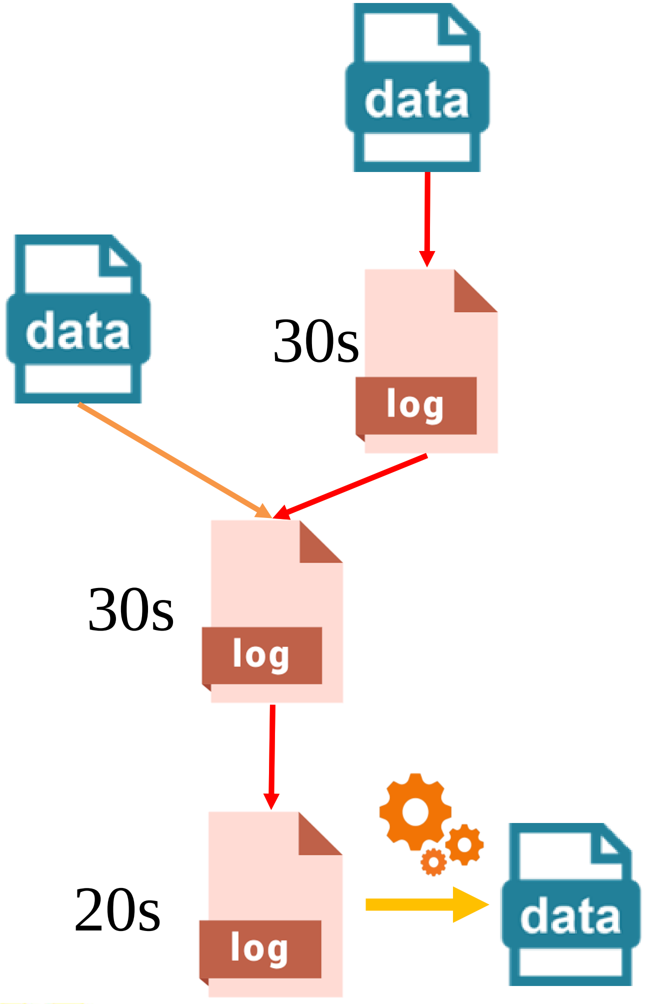
- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = **80s**

>

Materialization delay = **60**s

Storing past versions



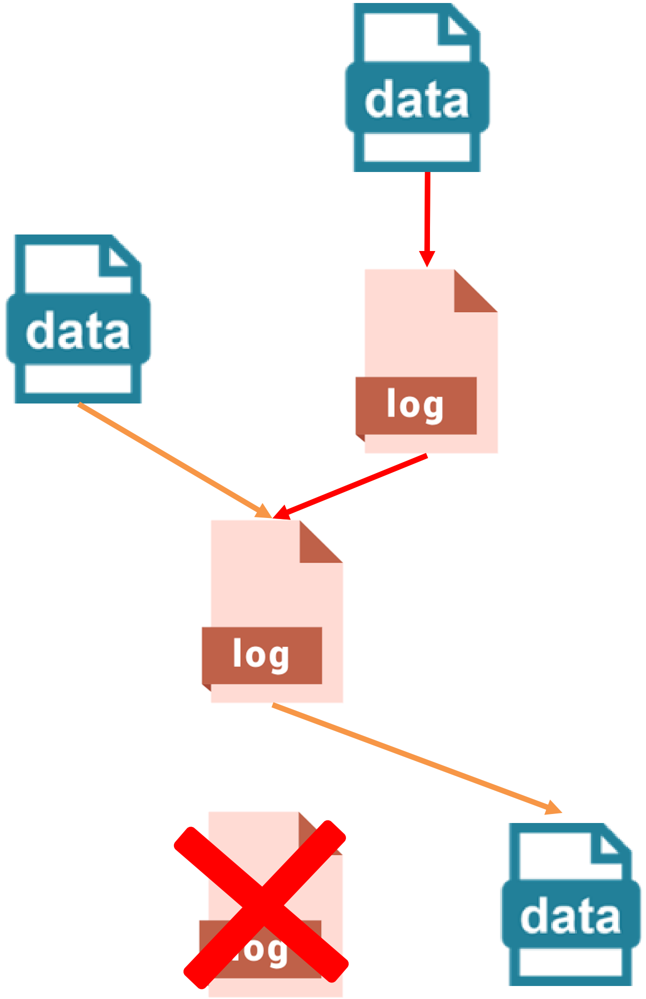
- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay

Total regeneration time = **80s**

>

Materialization delay = **60**s

Storing past versions

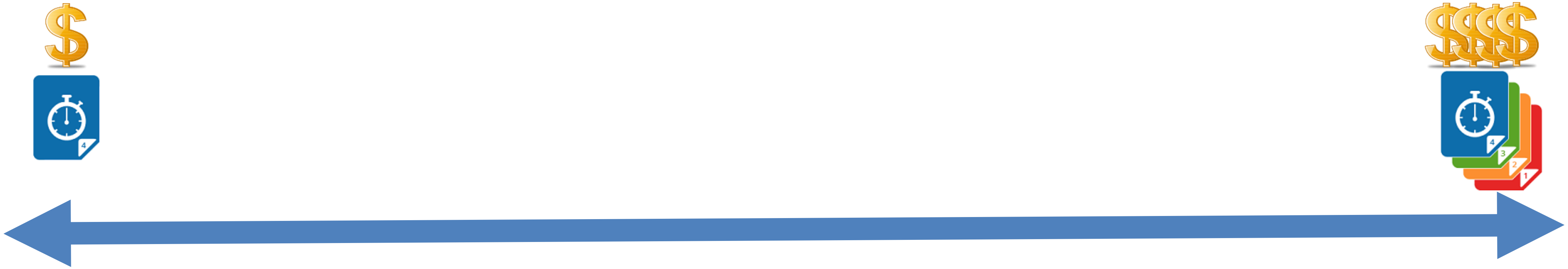


- Maximum materialization delay
  - Time bound for reconstructing any version
- Longest path > materialization delay
  - A greedy algorithm

Materialization delay = **60s**

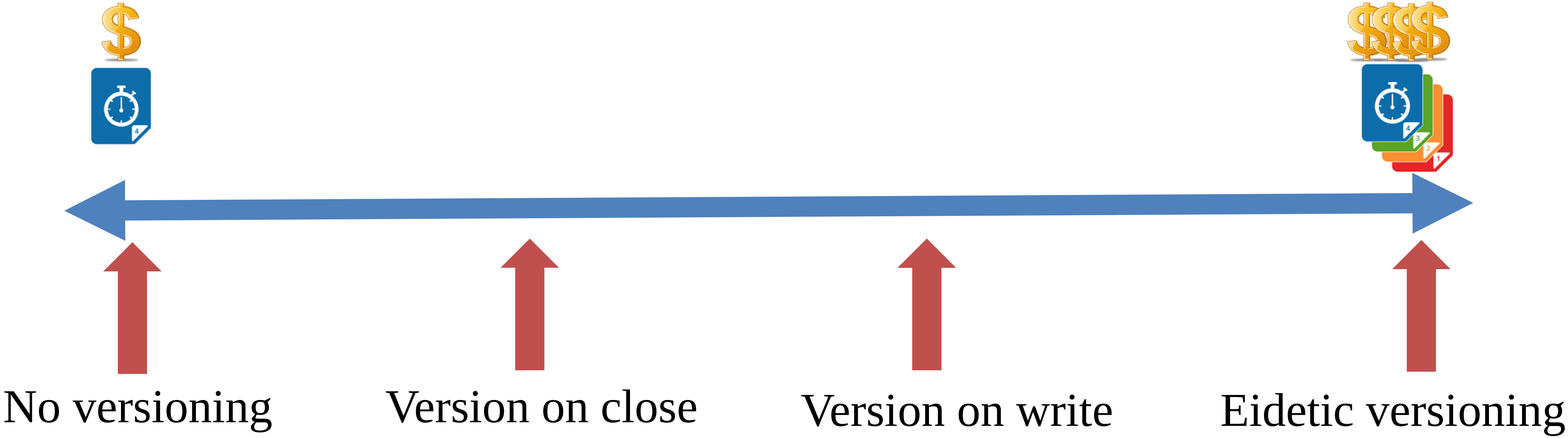
Storing past versions: versioning policies

- Frequency of versioning



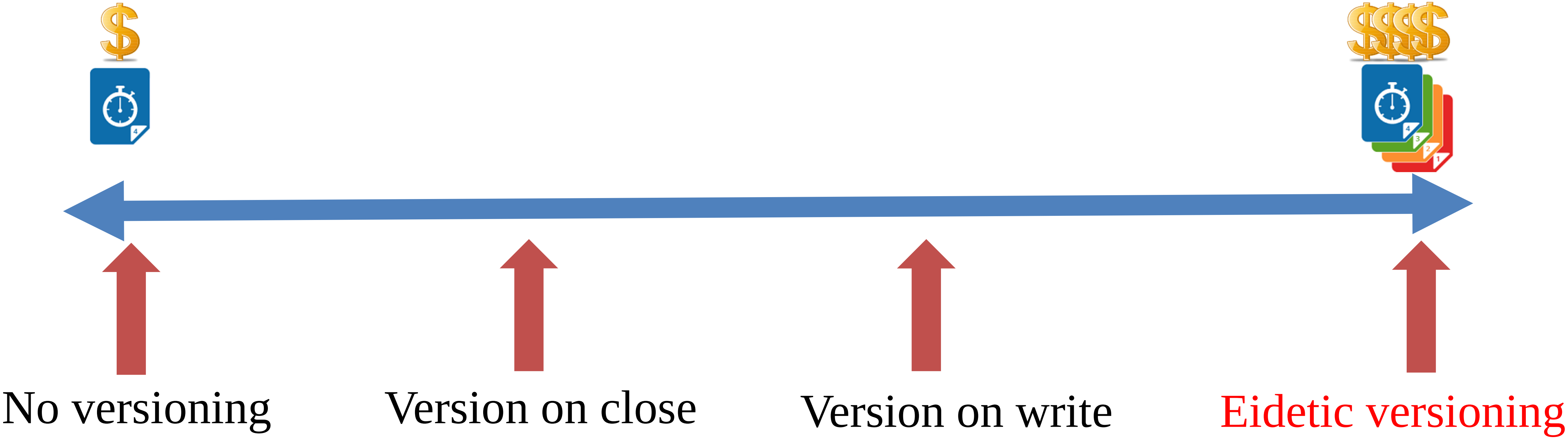
Storing past versions: versioning policies

- Frequency of versioning



Storing past versions: versioning policies

- Frequency of versioning



Memory-mapped files

Any past transient state in memory?

Optimization: log compression

- Chunk-based deduplication is effective for file data
  - Executions of the same application have similar patterns
  - Can it also be applied to computation (logs of nondeterminism)?

- Delta compression

Optimization: log compression

- Problem: a smattering of values differ in each log

	Log entry	Values
1	*open*	rc=3
2	*pthread_lock*	rc=0
3	*mmap*	rc=<addr>,file=<id,version>
4	*read*	rc=<size>,file=<id,version>
5	*gettimeofday*	rc=0,time=9.90
6	*write*	rc=<size>
7	*pthread_unlock*	rc=0
	…	…

	Log entry	Values
1	*open*	rc=3
2	*pthread_lock*	rc=0
3	*mmap*	rc=<addr>,file=<id,version>
4	*read*	rc=<size>,file=<id,version>
5	*gettimeofday*	rc=0,time=1.10
6	*write*	rc=<size>
7	*pthread_unlock*	rc=0
	…	…

Optimization: log compression

- Problem: a smattering of values differ in each log

	Log entry	Values
1	*open*	rc=3
2	*pthread_lock*	rc=0
7	*pthread_unlock*	rc=0
	…	…

	Log entry	Values
1	*open*	rc=3
2	*pthread_lock*	rc=0
7	*pthread_unlock*	rc=0
	…	…

Delta compression: 42% reduction

Outline

- Introduction
- Writing files
- Storing files
- Evaluation

Evaluation

- How much does Knockoff reduce bandwidth usage?
- How much does Knockoff reduce storage costs?
- What is Knockoff’s performance overhead?

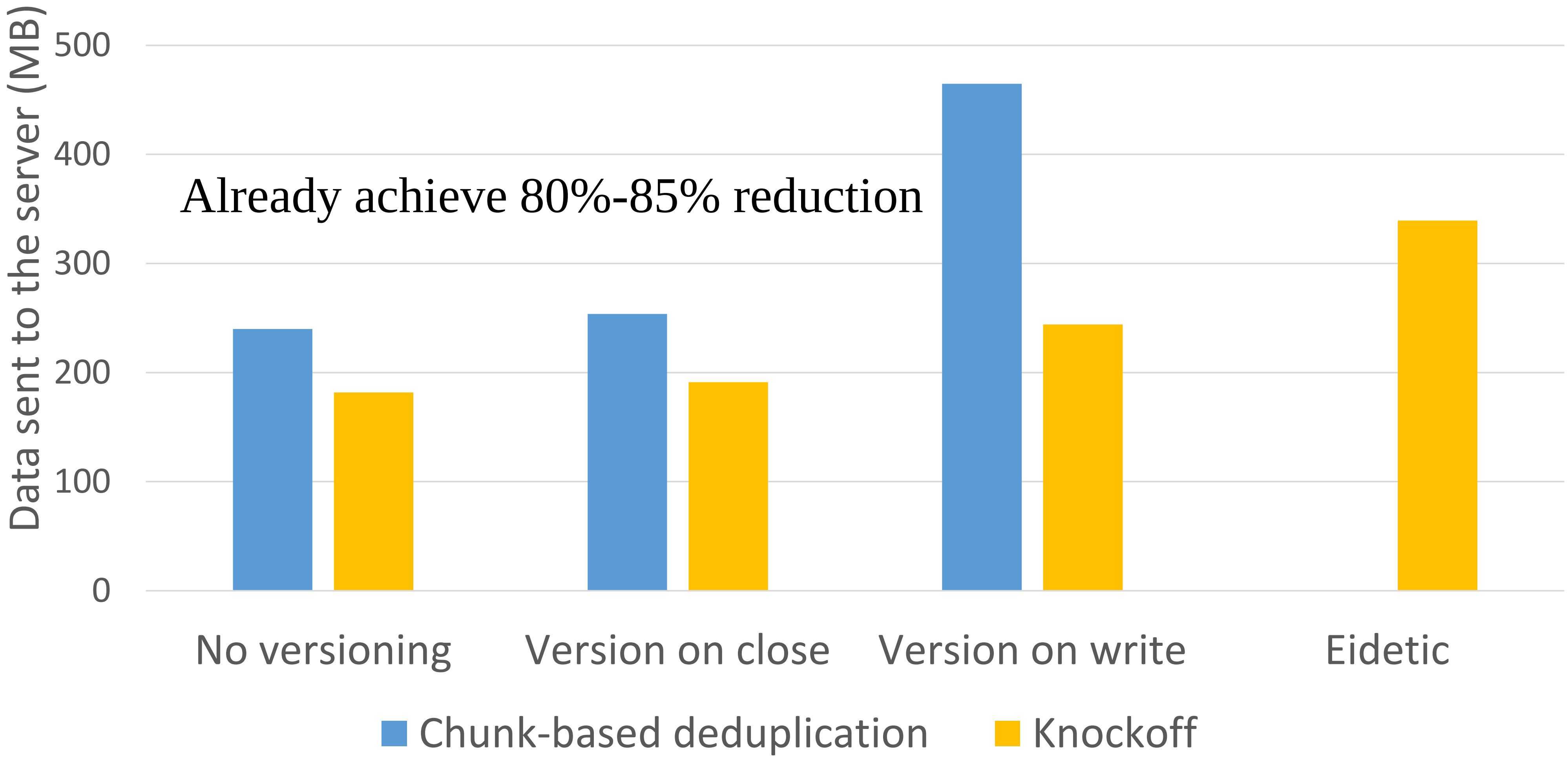
- For more experimental results, please refer to our paper

Experimental setup

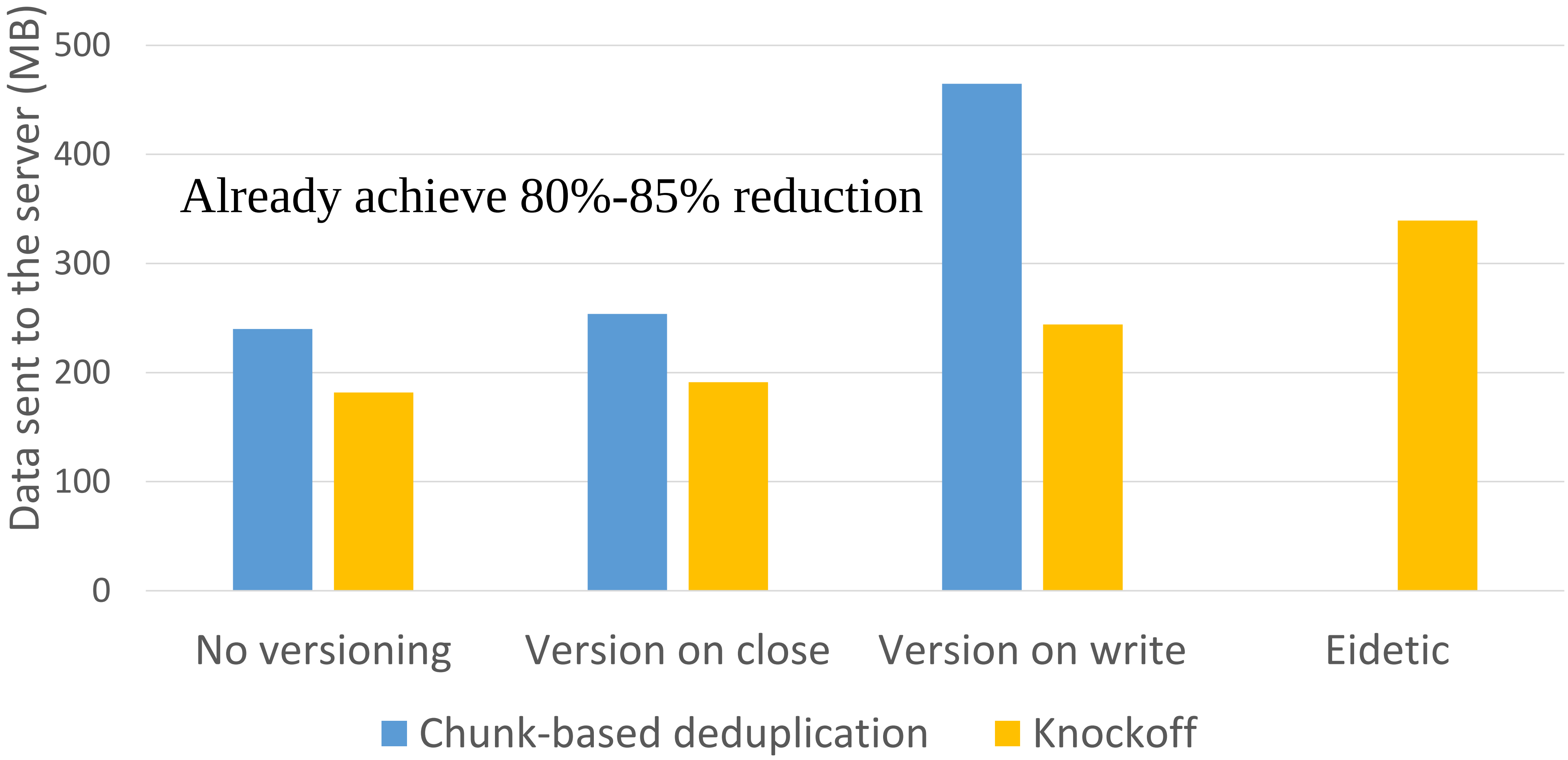
- User study
  - 8 participants performed several simple tasks in one hour
- 20-day study
  - A single-user longitudinal study
- A variety of programs used
  - Various Linux utilities, text editors and programming languages

Bandwidth usage: user study

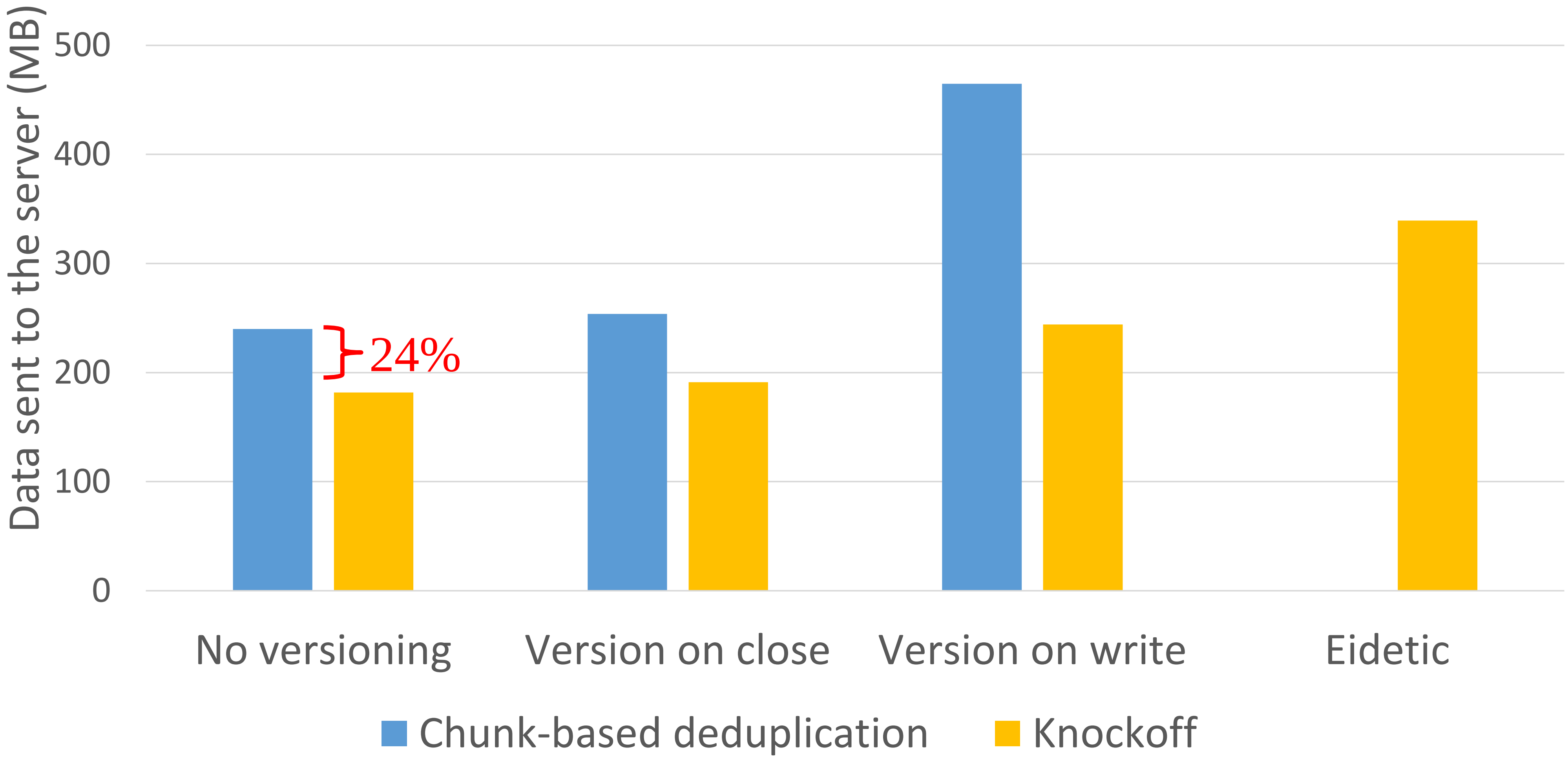
Bandwidth usage: user study



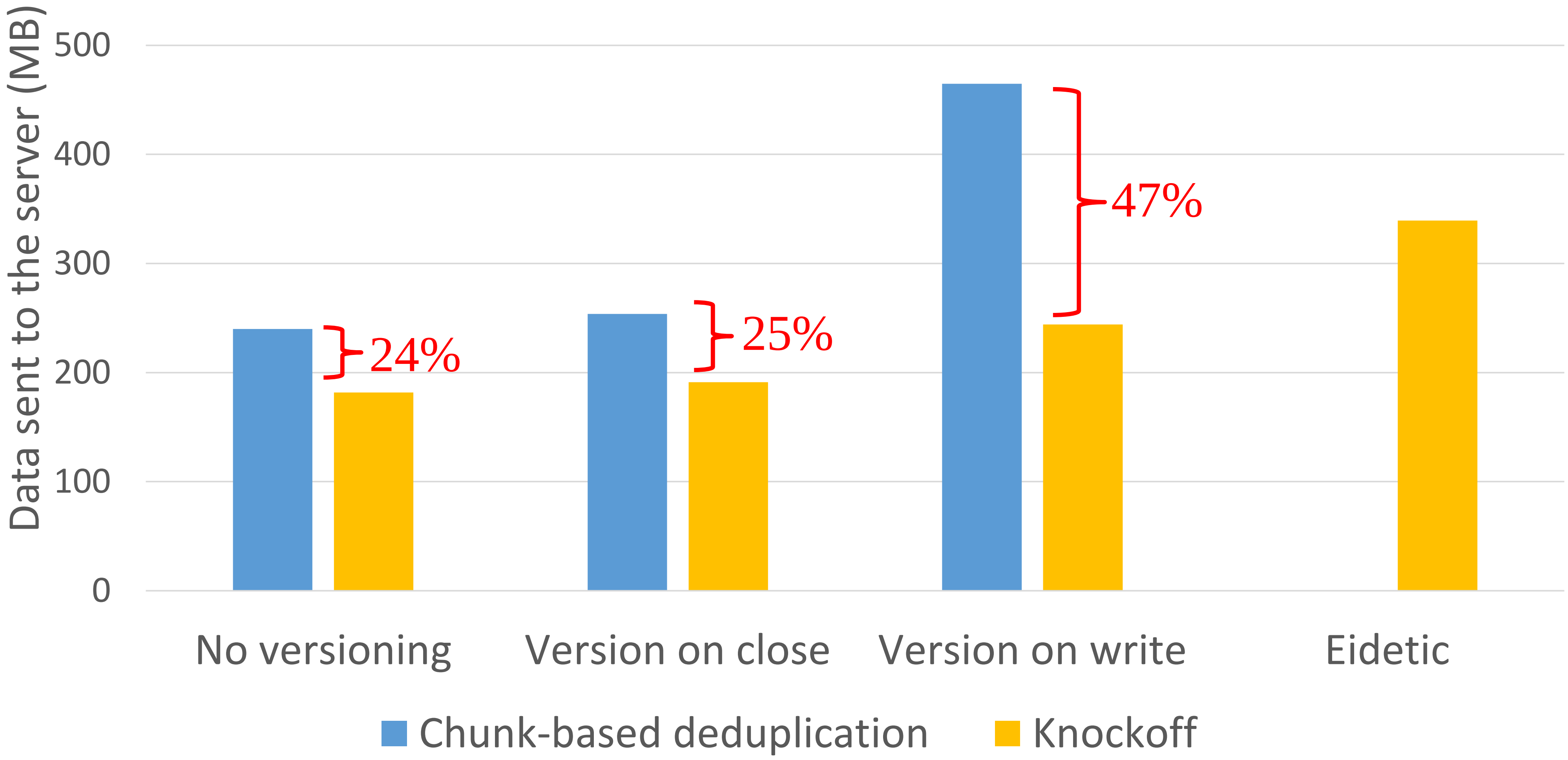
Bandwidth usage: user study



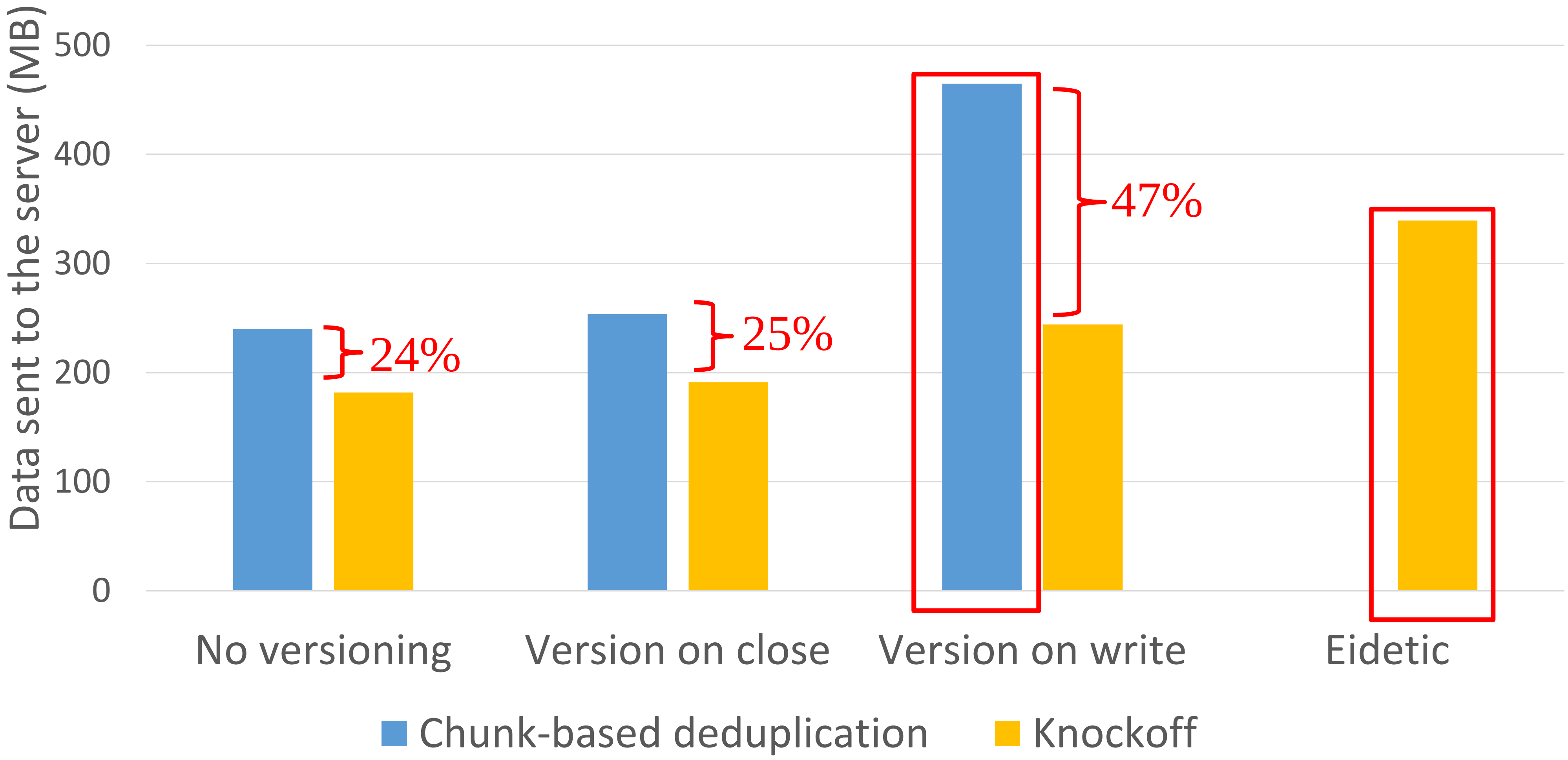
Bandwidth usage: user study



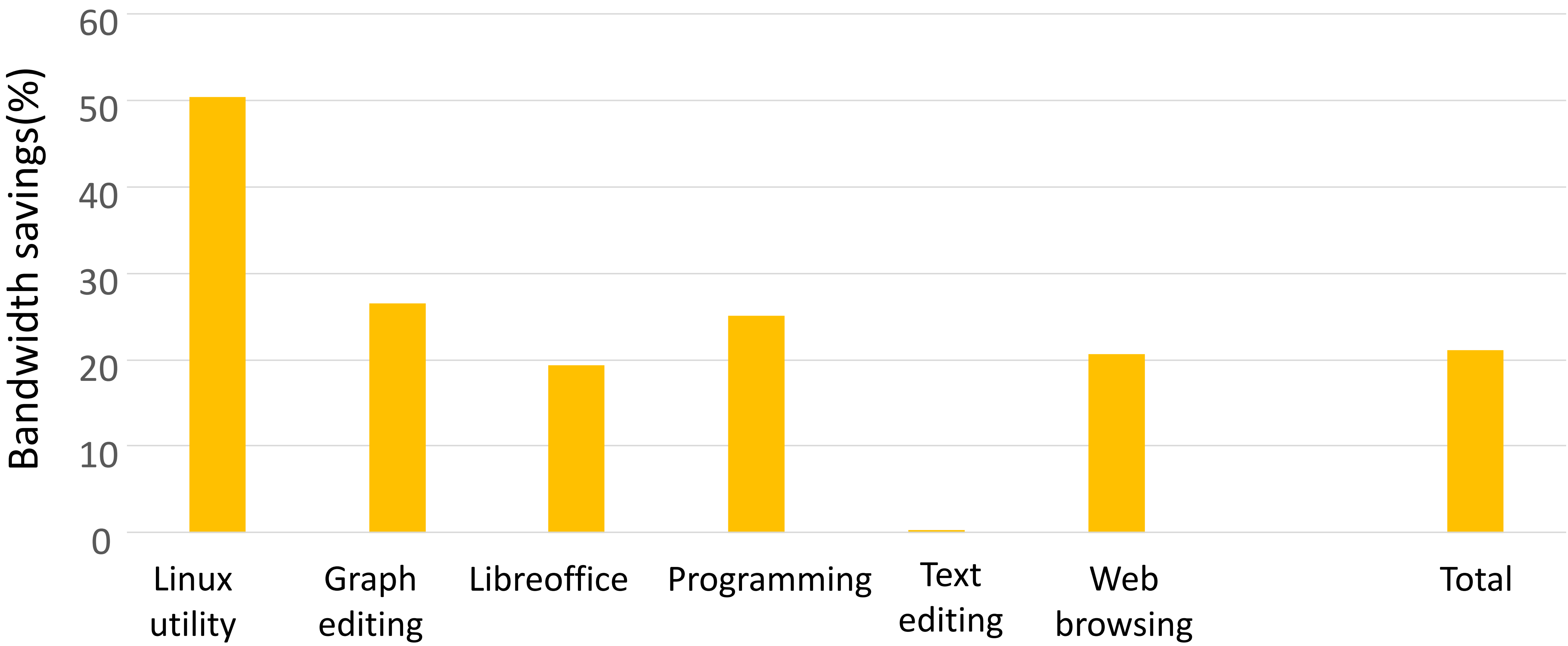
Bandwidth usage: user study



Bandwidth usage: user study

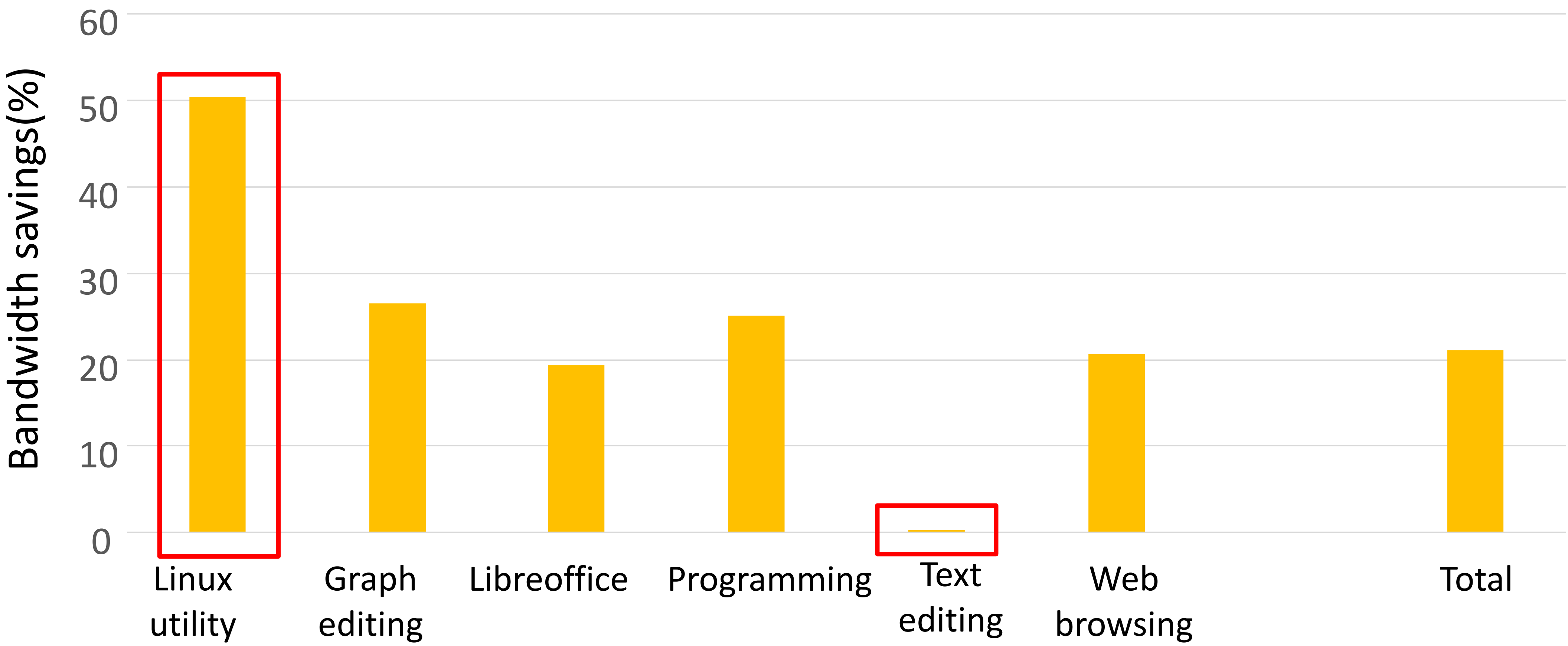


Variances across applications



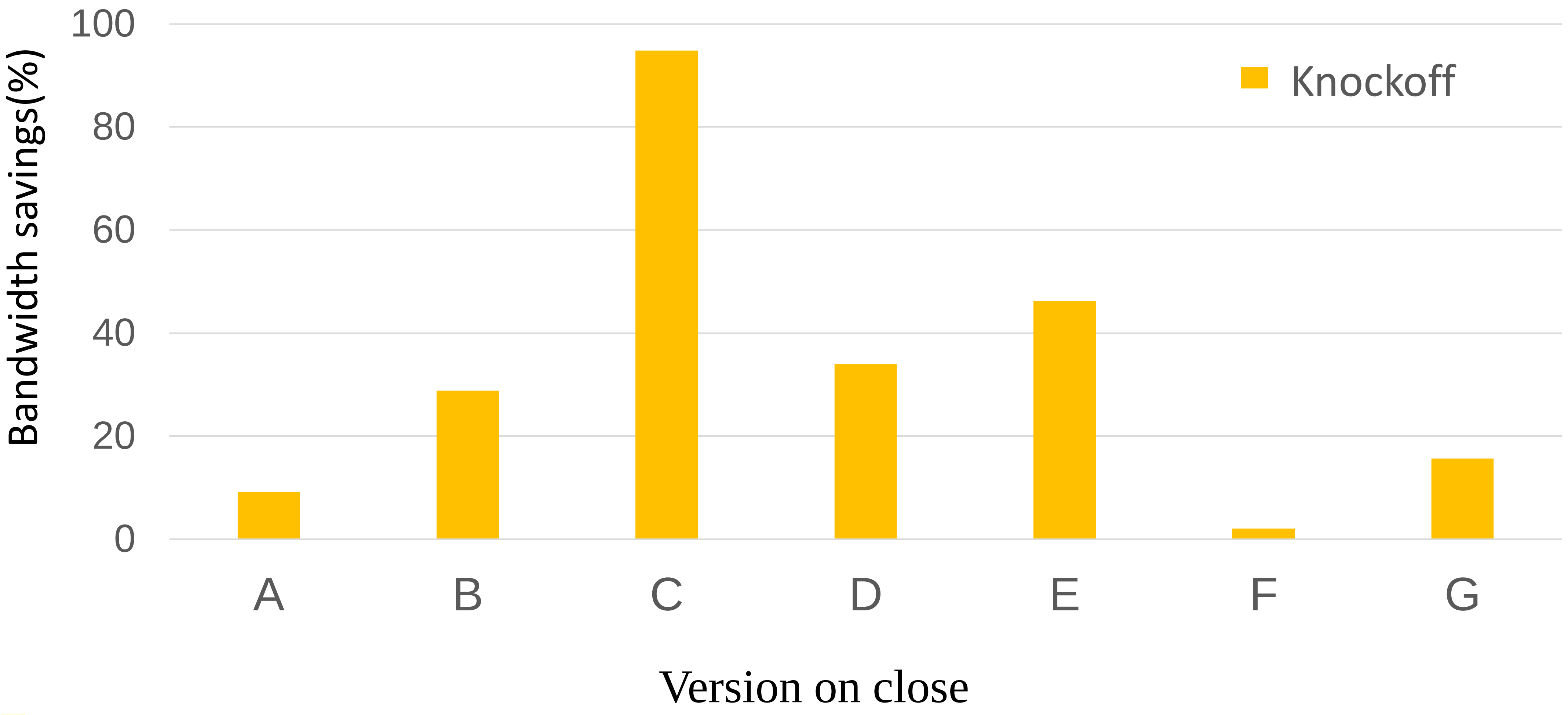
Version on close

Variances across applications

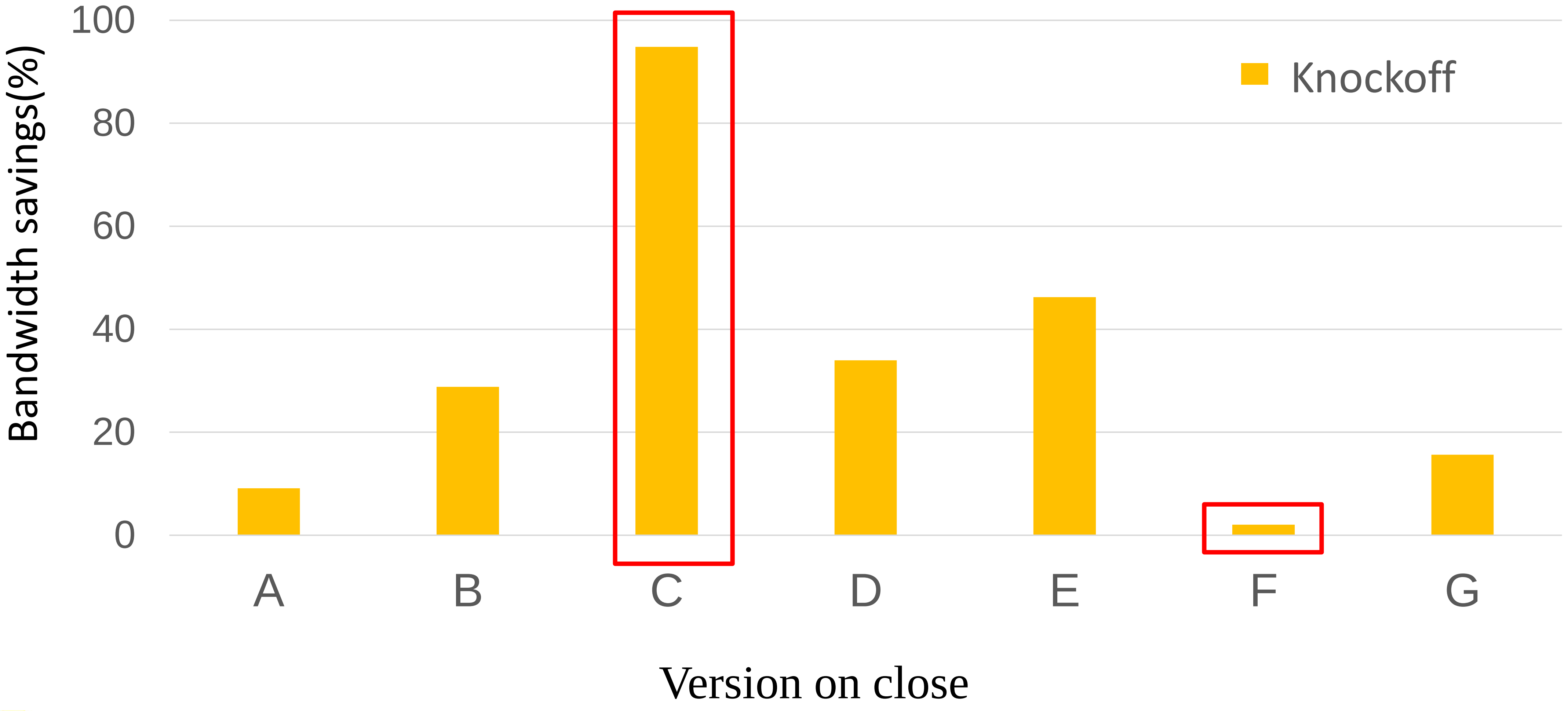


Version on close

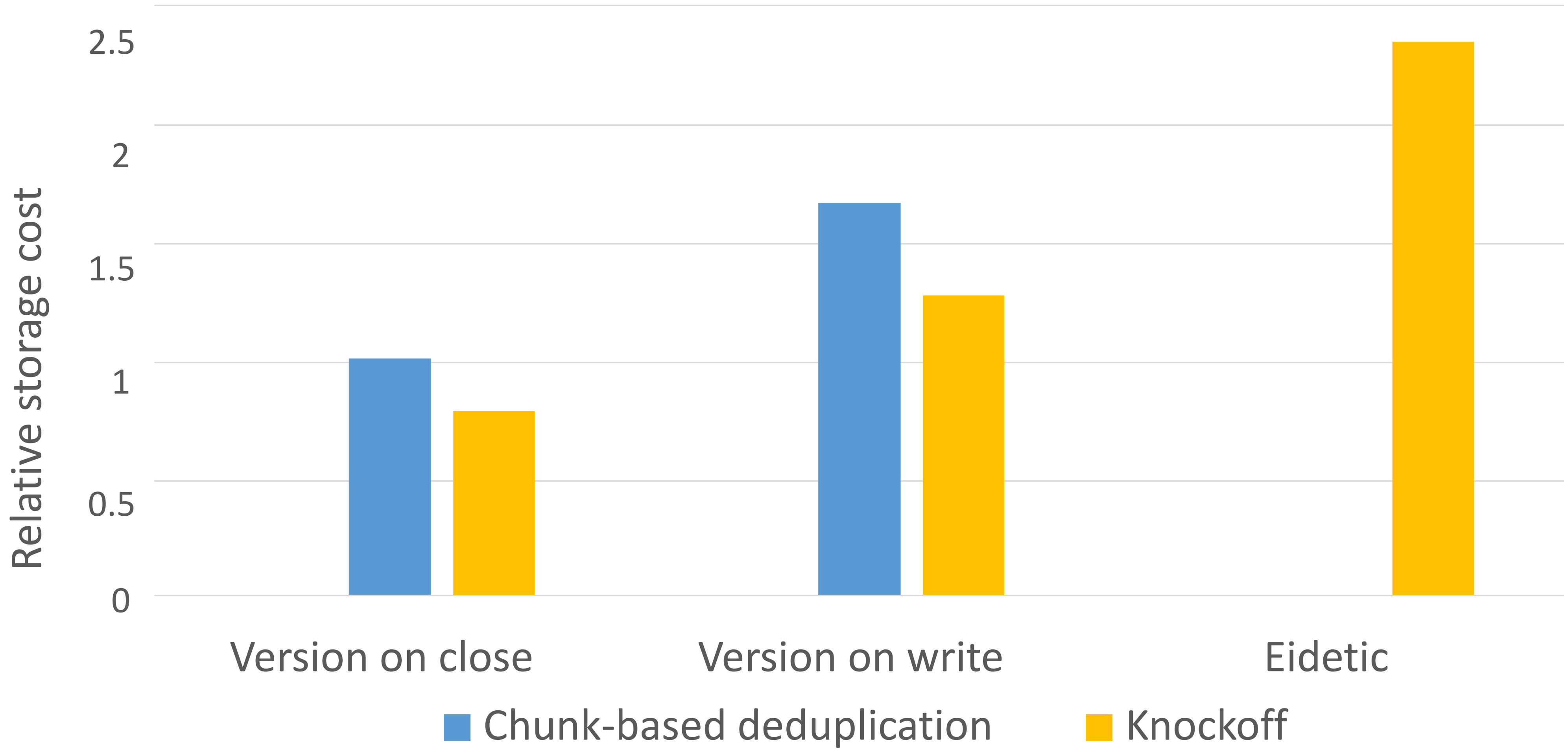
Variances across users



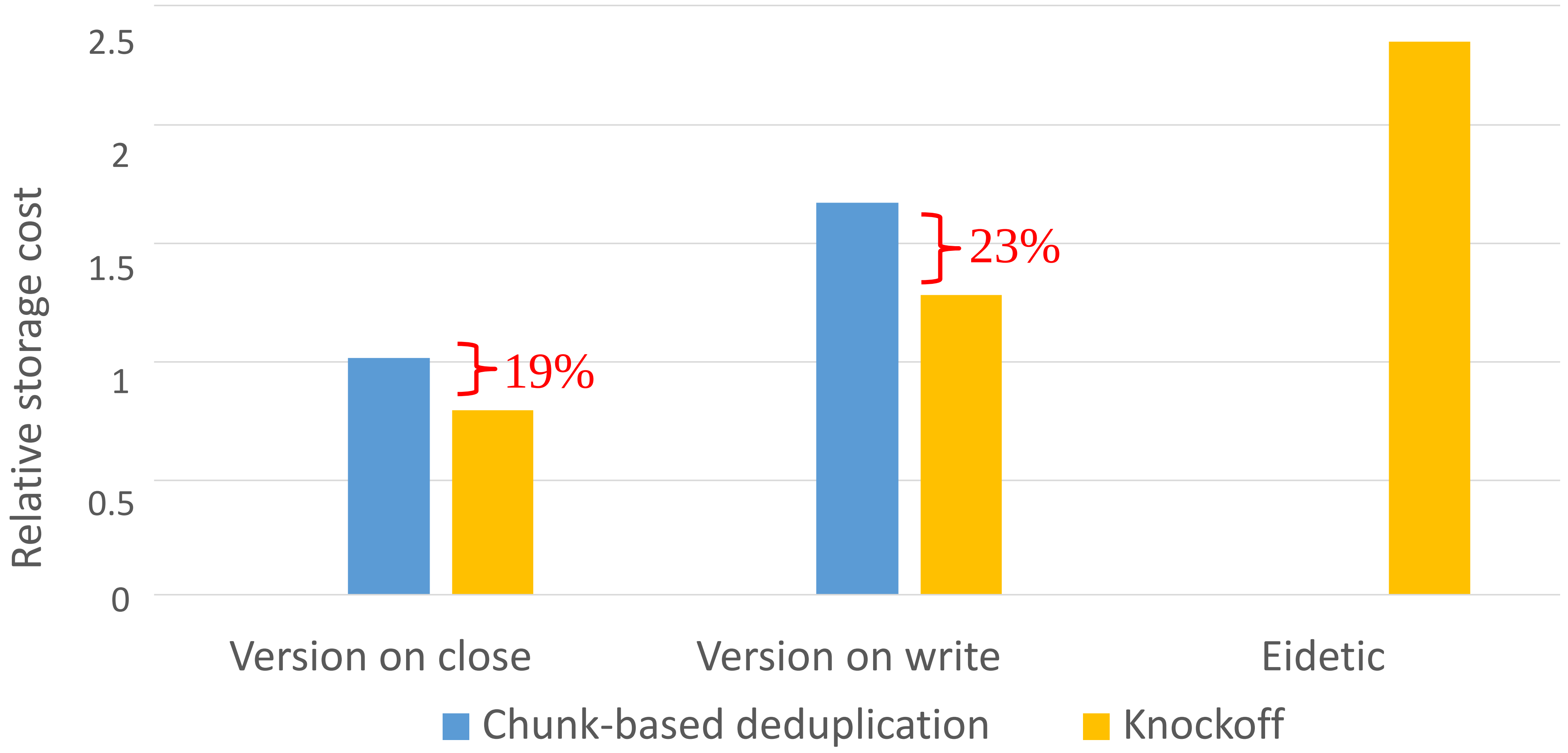
Variances across users



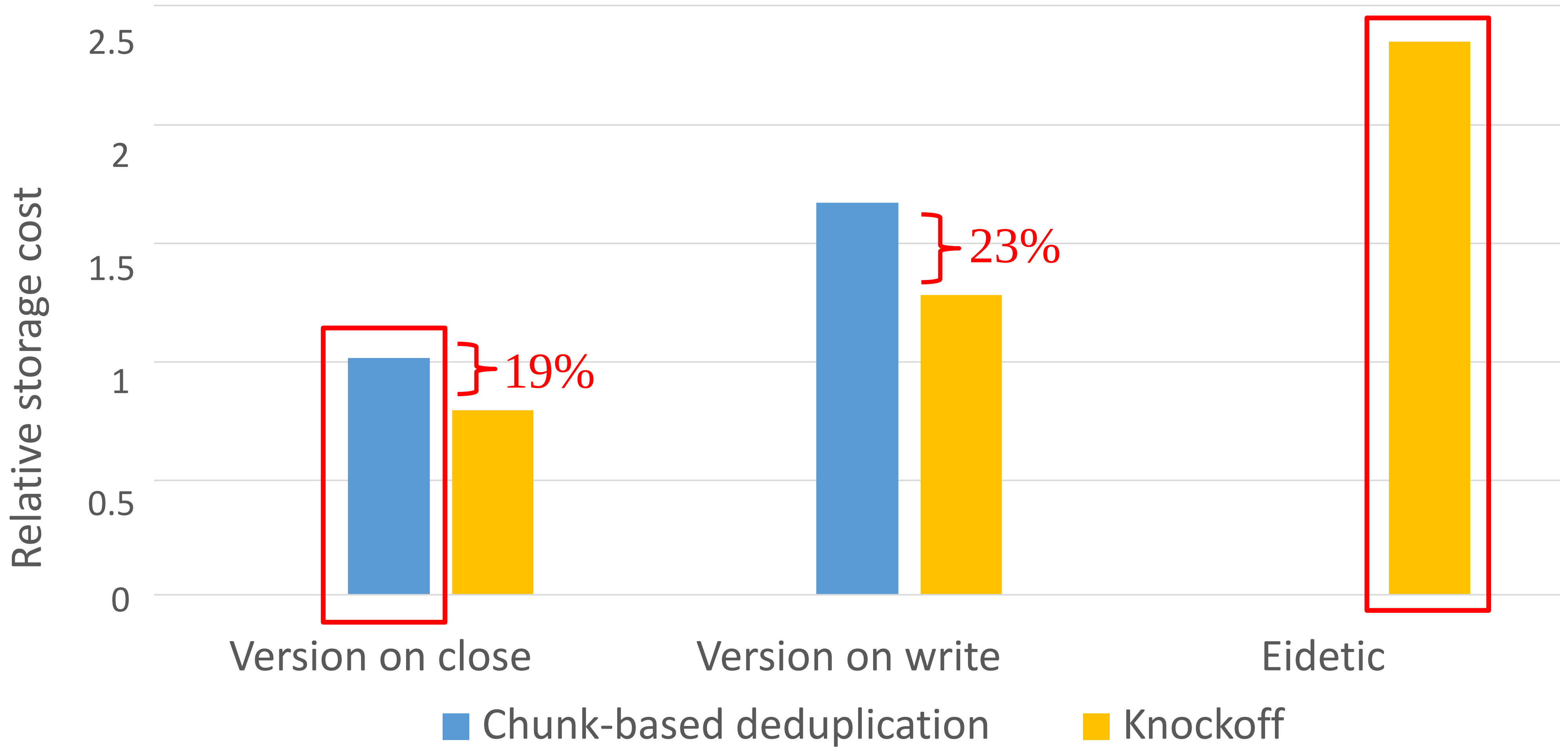
Relative storage costs for past versions



Relative storage costs for past versions

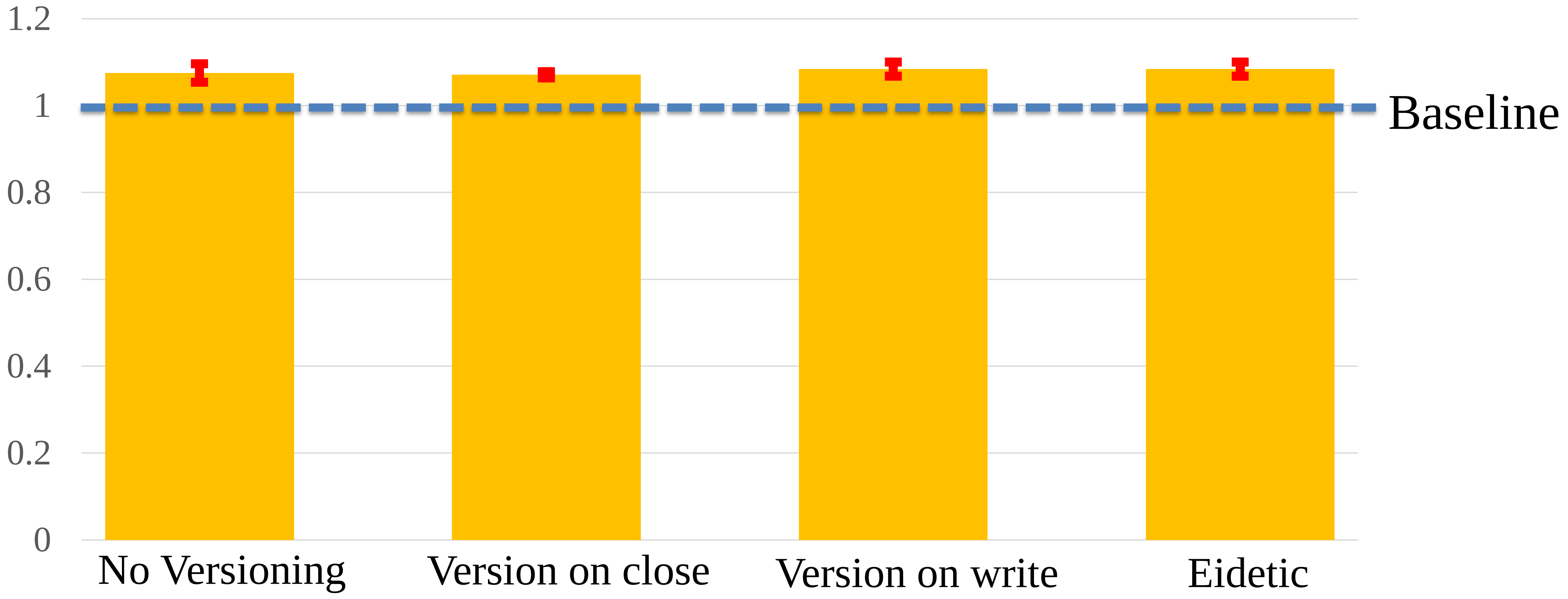


Relative storage costs for past versions



Performance overheads

- 7-8% performance overheads

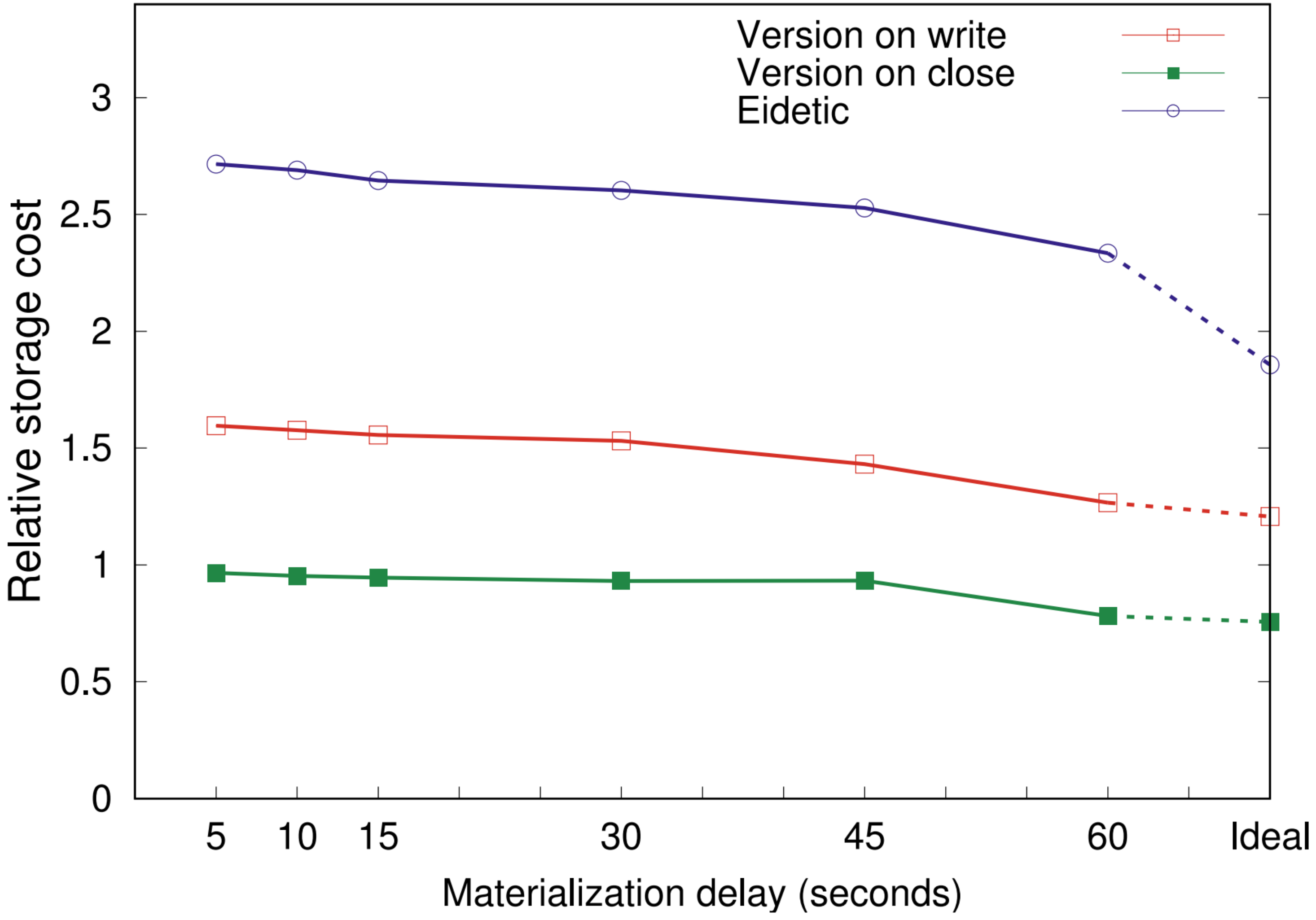


Conclusion

- A new dimension for reducing costs
- Selectively substitute computation for data
- A general-purpose system for deterministic recomputation
  - Reduces storage and communication costs for existing versioning policies
  - Enables eidetic versioning

Thank you!

Varying the materialization delay



Monetary costs

	Price($ per GB)	Knockoff savings					
		No version		Version on close		Version on write	
		20-day study	User study	20-day study	User study	20-day study	User study
4G network	4.50	21.0%	21.8%	21.2%	21.7%	22.9%	46.3%
Expensive ISP	0.20	20.3%	18.4%	20.5%	18.5%	22.0%	43.3%
Cheap ISP	0.05	18.1%	13.8%	18.2%	14.5%	19.2%	34.9%
Hypothetical ISP	0.005	8.2%	4.9%	8.4%	5.8%	8.2%	11.5%

Table 2: Relative cost savings from using Knockoff for different versioning policies. We show costs for a typical 4G cellular network, an expensive current ISP, a cheap current ISP, and a hypothetical ISP that is an order of magnitude cheaper than the cheap current ISP.

Workload characteristics

	20-day study	User study
Disk read (MB)	5473	2583
Disk write (MB)	6706	4339
File open count	261523	418594
Number of executions	3803	1146
Number of programs	75	63

Table 1: Workload characteristics