Application-aware I/O Scheduling in the Parallel File System Server Side

Francieli Zanon Boito

Philippe Navaux

GPPD - II - **Federal University of Rio Grande do Sul** (UFRGS), Brazil

Yves Denneulin

INRIA – LIG – **Grenoble University**, France

Application-aware **I/O Scheduling** in the Parallel File System Server Side

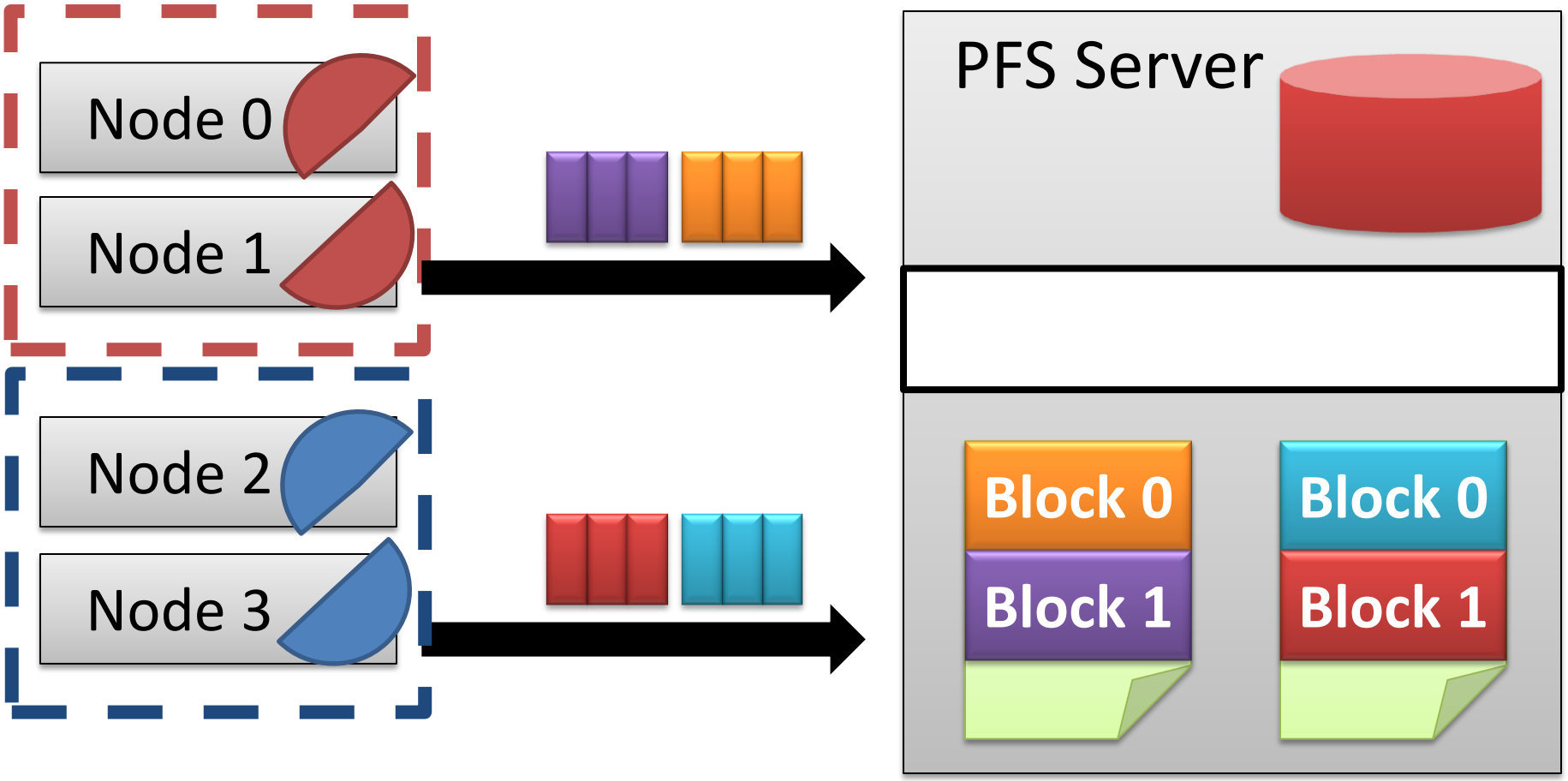
L I G

INSTITUTO DE INFORMÁTICA UFRGS

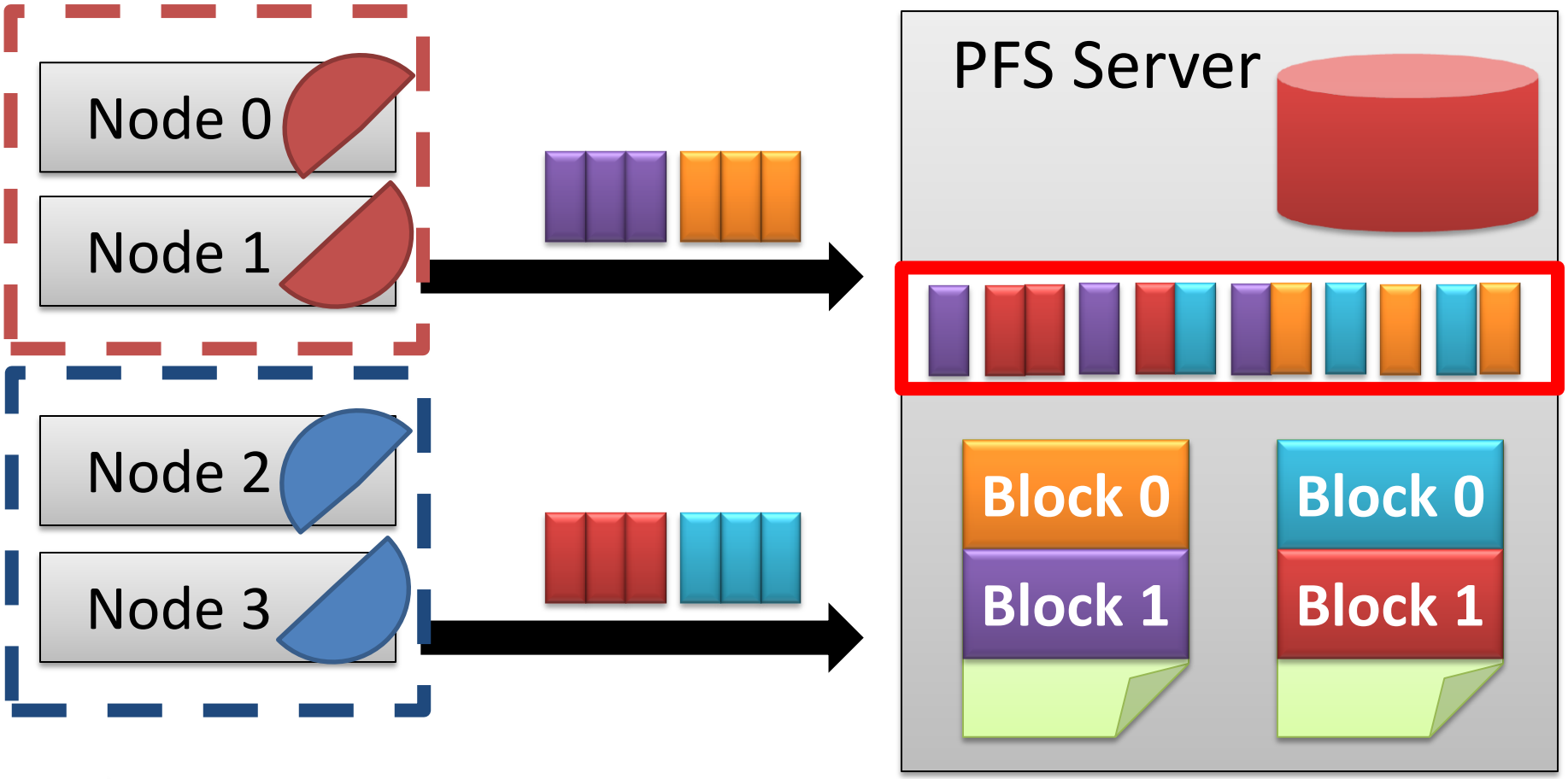
I/O Scheduling

- Optimizations adjust the **access pattern of applications**
  - Individually
- **Multiple-applications** Scenarios
  - Effect of **interference**

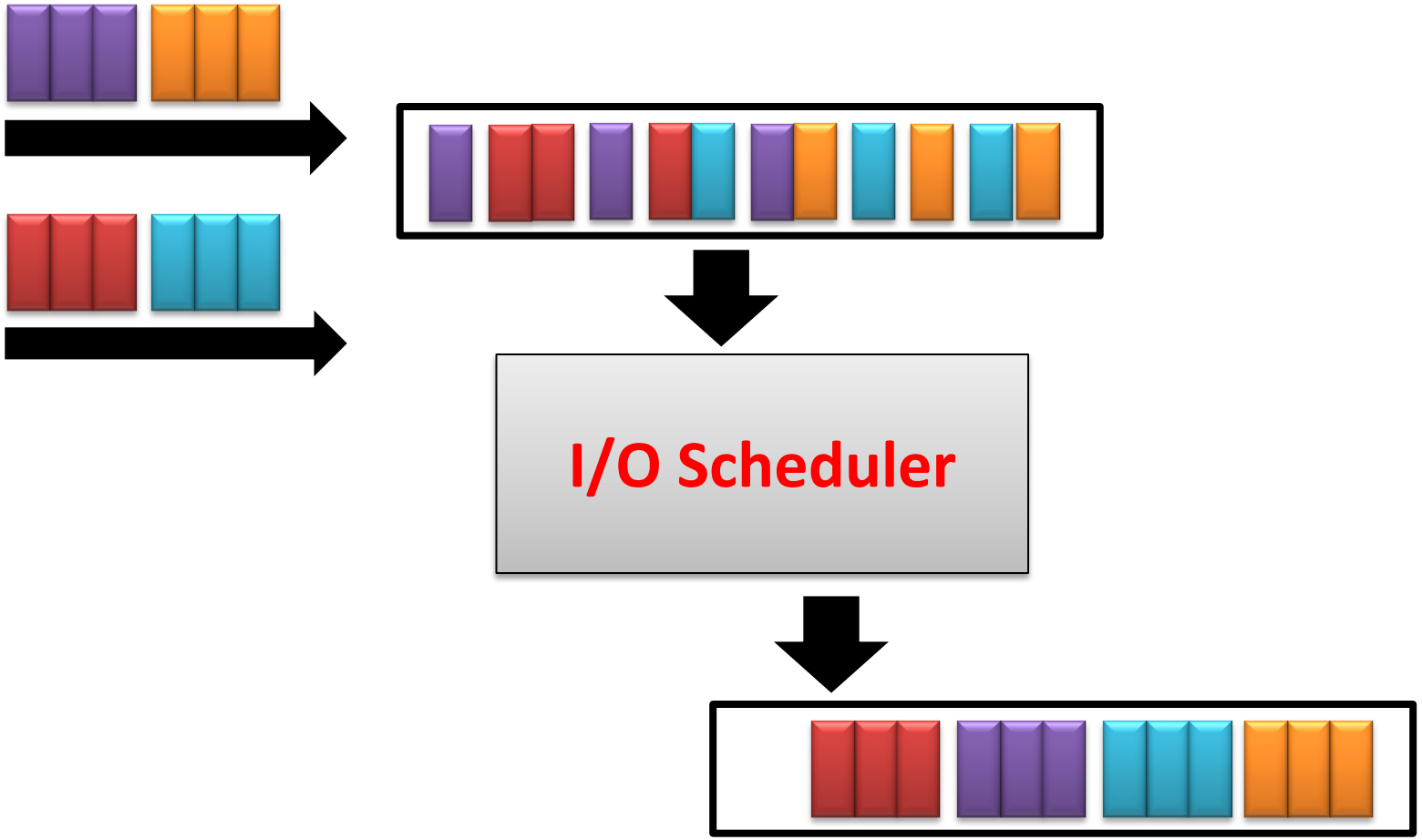
I/O Scheduling



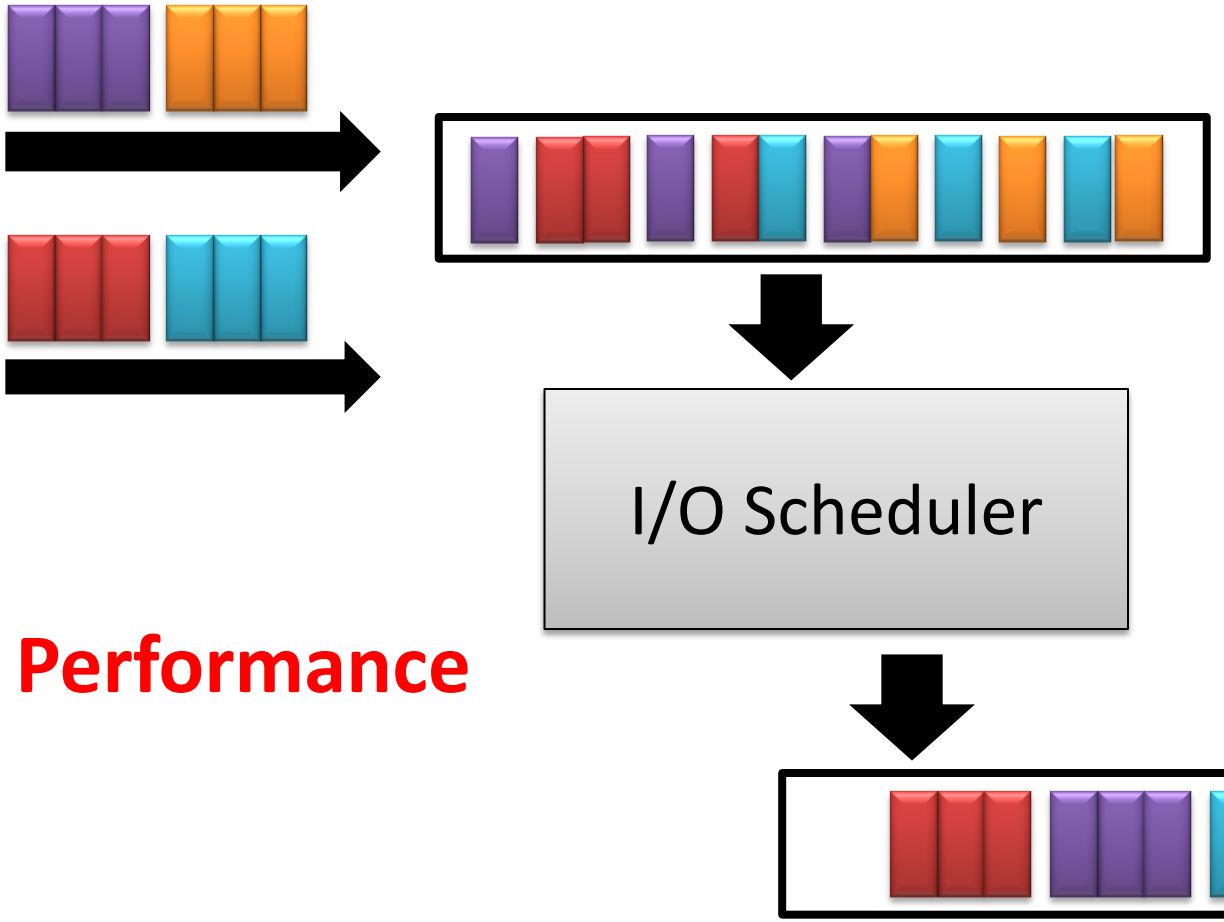
I/O Scheduling



I/O Scheduling



I/O Scheduling

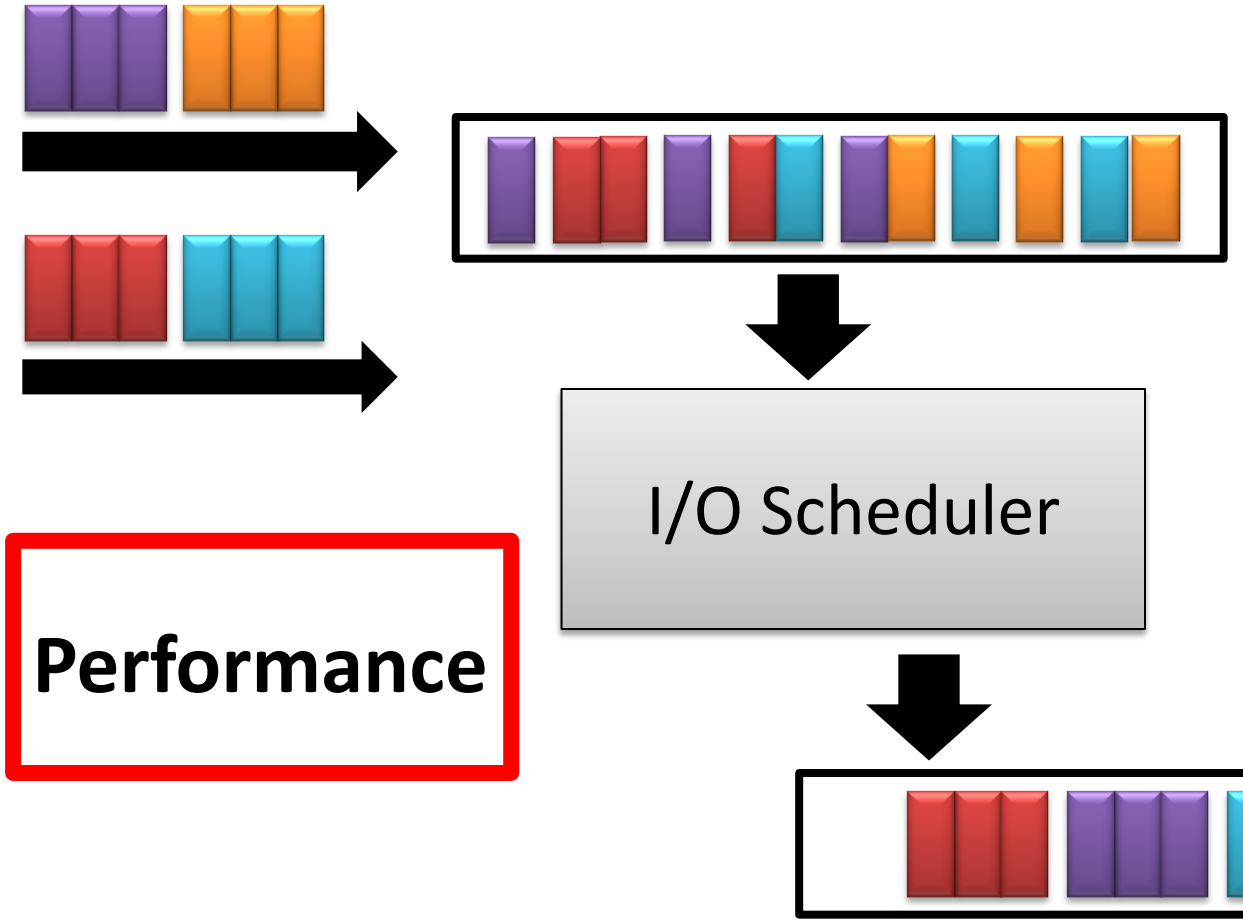


Fairness

Response Time

Performance

I/O Scheduling



Fairness

Response Time

Performance

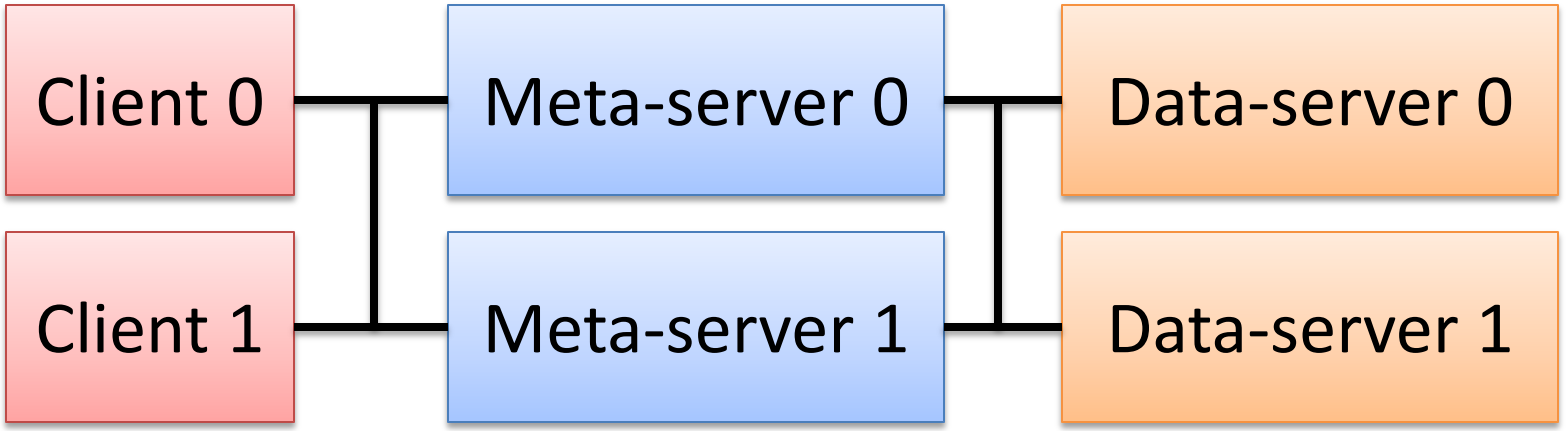
I/O Scheduling: LibaIOLi

- **aIOLi**: I/O Scheduling Framework
  - [Lebre et al. 2006]
  - Centralized file system

- **LibaIOLi**: library to use with **PFS servers**

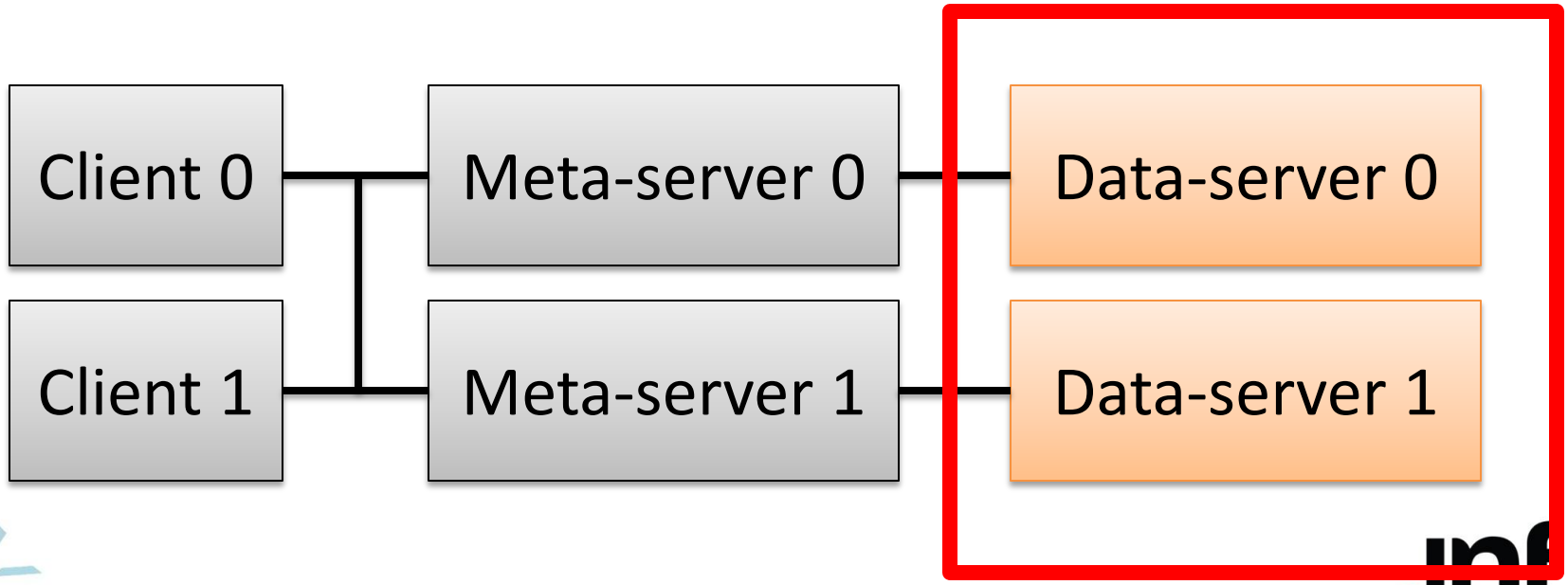
I/O Scheduling: LibaIOLi

- LibaIOLi + **dNFSp**
  - NFS-based **parallel file system**



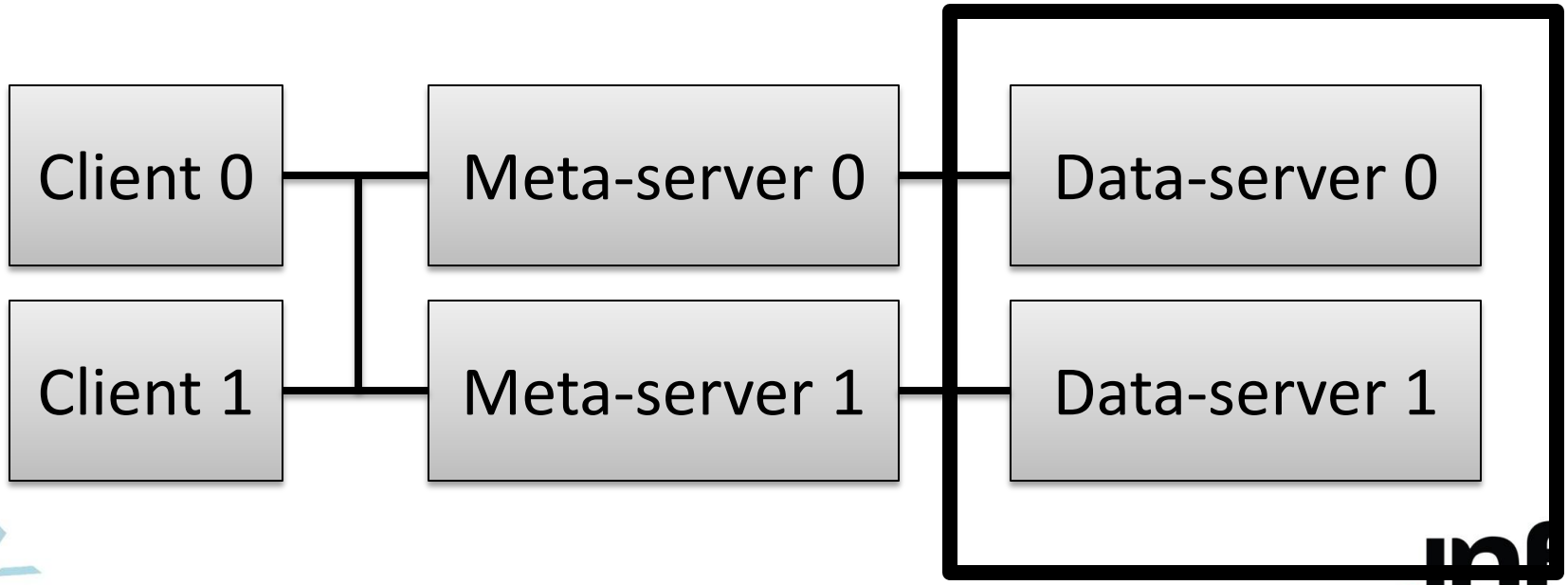
I/O Scheduling: LibaIOLi

- LibaIOLi + dNFSp
  - NFS-based parallel file system



I/O Scheduling: LibaIOLi

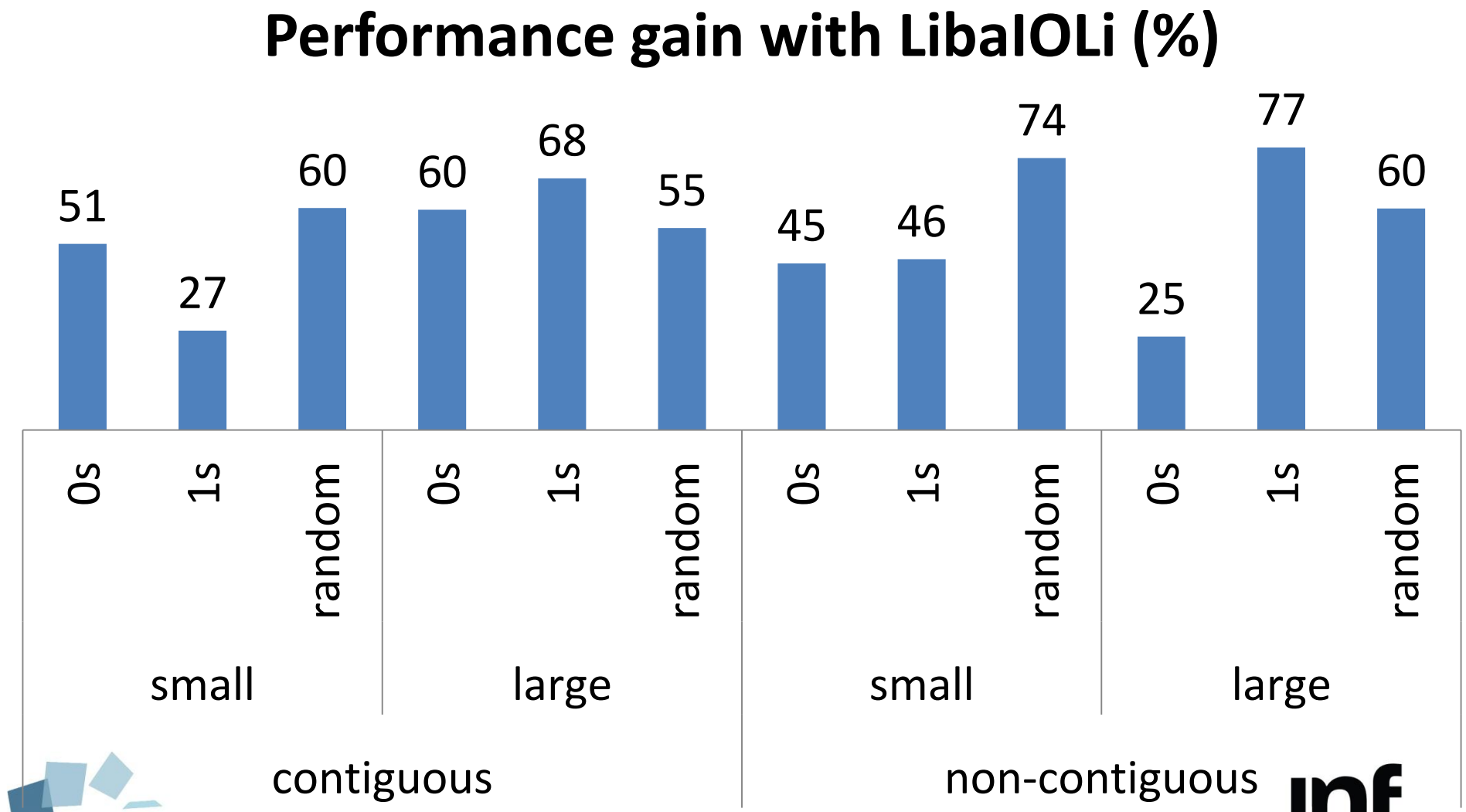
- Idea similar to **Lustre NRS**
  - But **more generic**



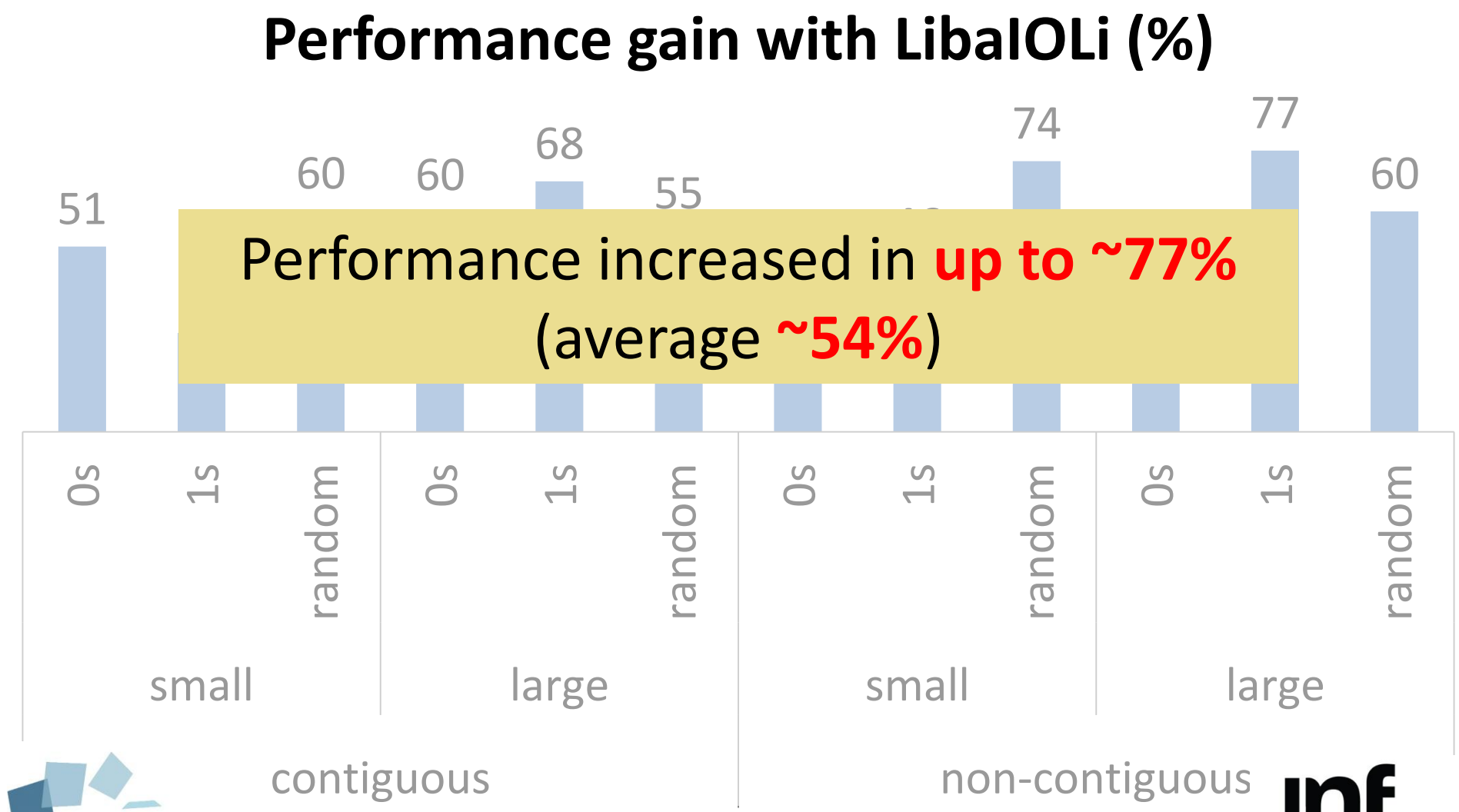
LibaIOLi + dNFSp Results

- MPI-IO Test
- Cluster Edel @ Grenoble.Grid5000
- dNFSp with 1 meta-server and 4 data-servers
- 32 clients – **single application**

LibaIOLi Results– Write Operations

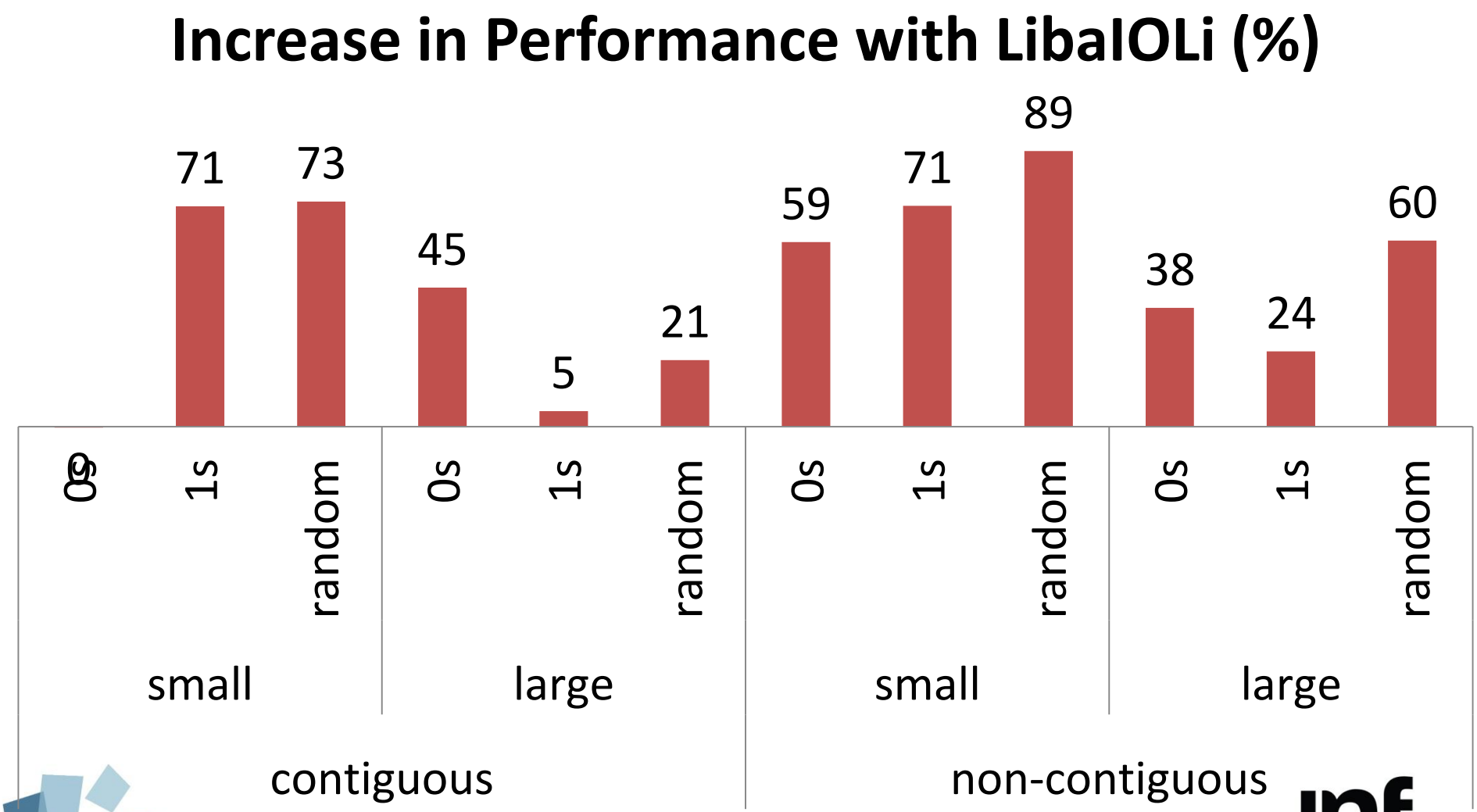


LibaIOLi Results – **Write Operations**



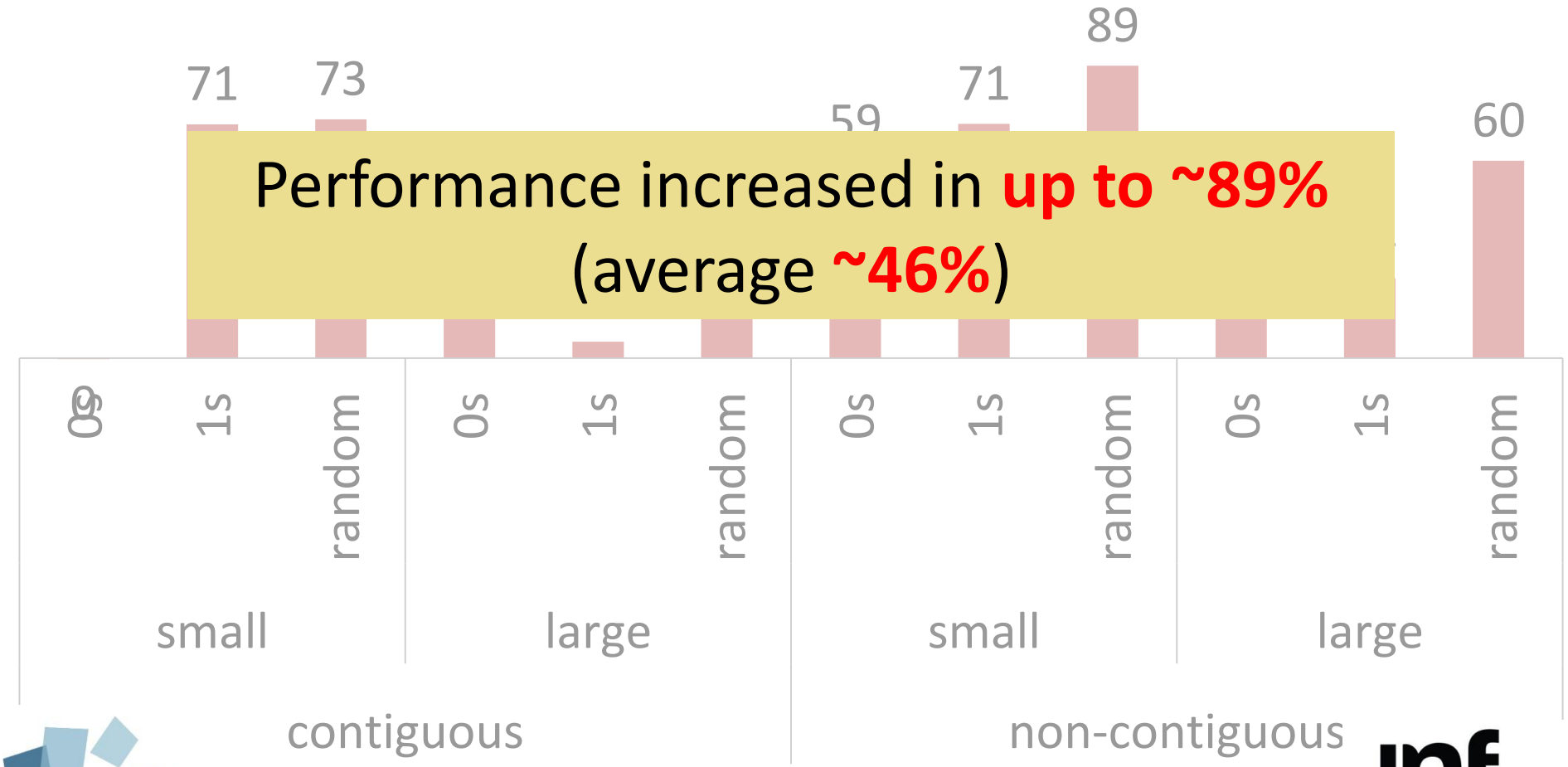
LibaIOLi Results – **Read Operations**

LibaIOLi Results – Read Operations



LibaIOLi Results – **Read Operations**

Increase in Performance with LibaIOLi (%)



Why does LibaIOLi **improve performance?**

I/O Scheduling with LibaIOLi

2 assumptions about performance:

1. Sequential is better than random

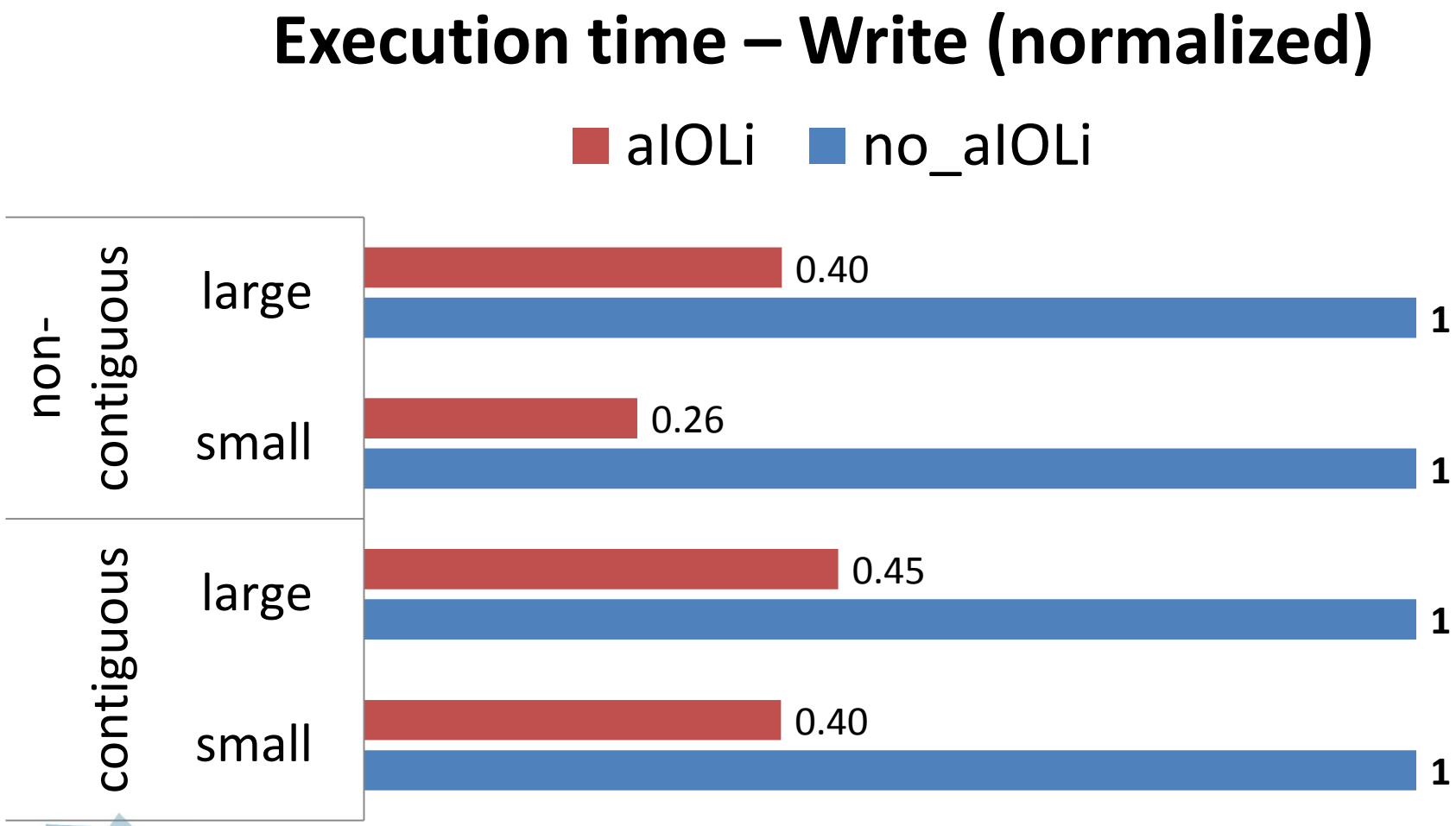
2. Large requests are better than small ones

I/O Scheduling with LibaIOLi

2 assumptions about performance:

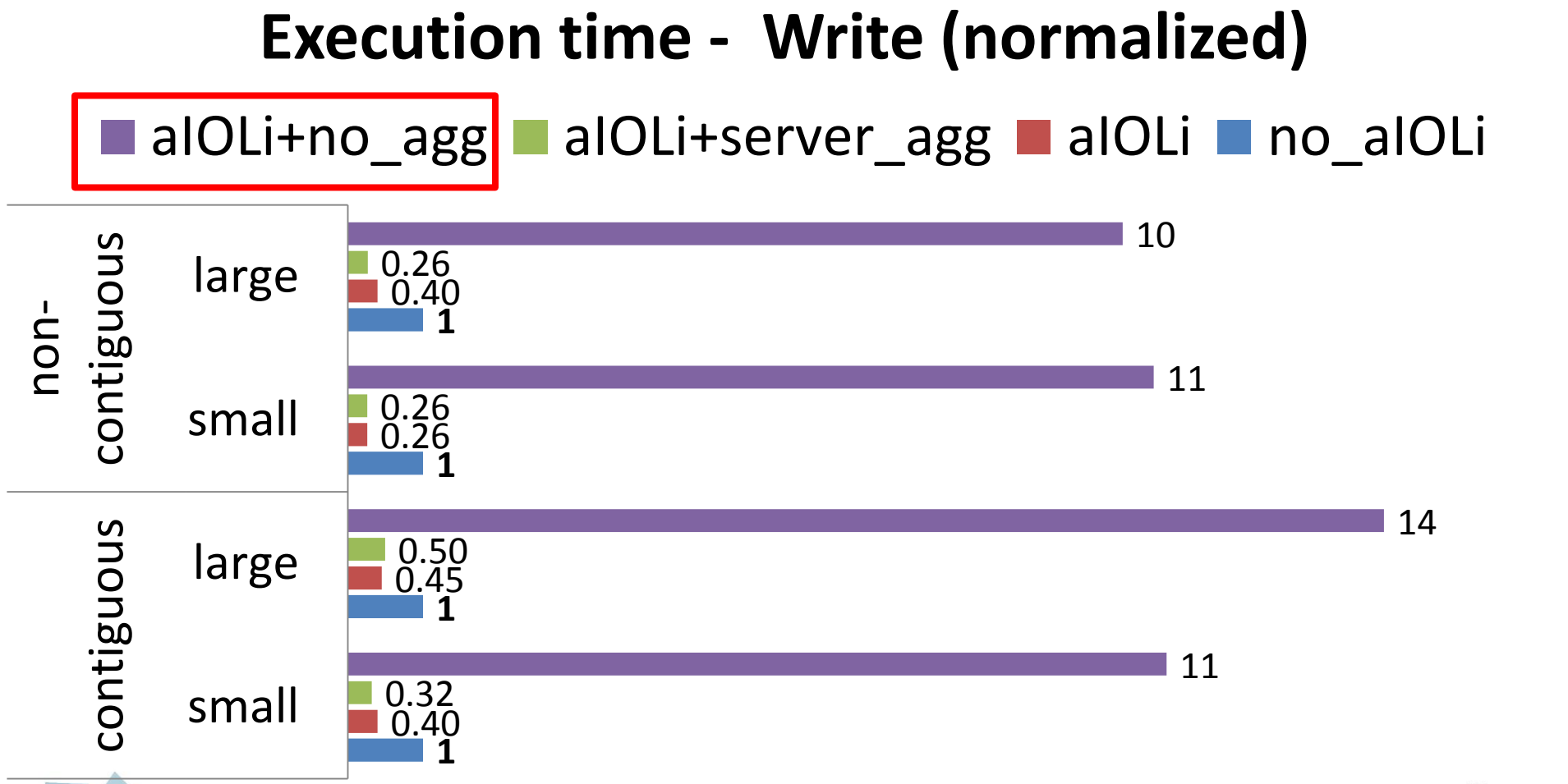
1. Sequential is better than random
   - **Reordering of requests**
2. Large requests are better than small ones
   - **Aggregation of requests**

LibalOLi - aggregations impact



LibaIOLi - aggregations impact

LibalOLi - aggregations impact



I/O Scheduling with LibaIOLi

2 assumptions about performance:

1. Sequential is better than random
   - Reordering of requests
2. Large requests are better than small ones
   - **Aggregation of requests**

I/O Scheduling with LibaIOLi

- aIOLi's approach for **larger aggregations**:
  - Wait (on specific conditions) for more contiguous requests

I/O Scheduling with LibaIOLi

- **Waiting conditions** for larger aggregations
  1. **Shift phenomena**

I/O Scheduling with LibaIOLi

- Waiting conditions for larger aggregations

1. **Shift phenomena**

I/O Scheduling with LibaIOLi

- Waiting conditions for larger aggregations

1. **Shift phenomena**

I/O Scheduling with LibaIOLi

- Waiting conditions for larger aggregations
  1. Shift phenomena **(Detection is not this fast)**

I/O Scheduling with LibaIOLi

- Waiting conditions for larger aggregations
  1. Shift phenomena
  2. **< Largest** aggregation performed

I/O Scheduling with LibaIOLi

- Waiting conditions for larger aggregations
  1. Shift phenomena
  2. < Largest aggregation performed

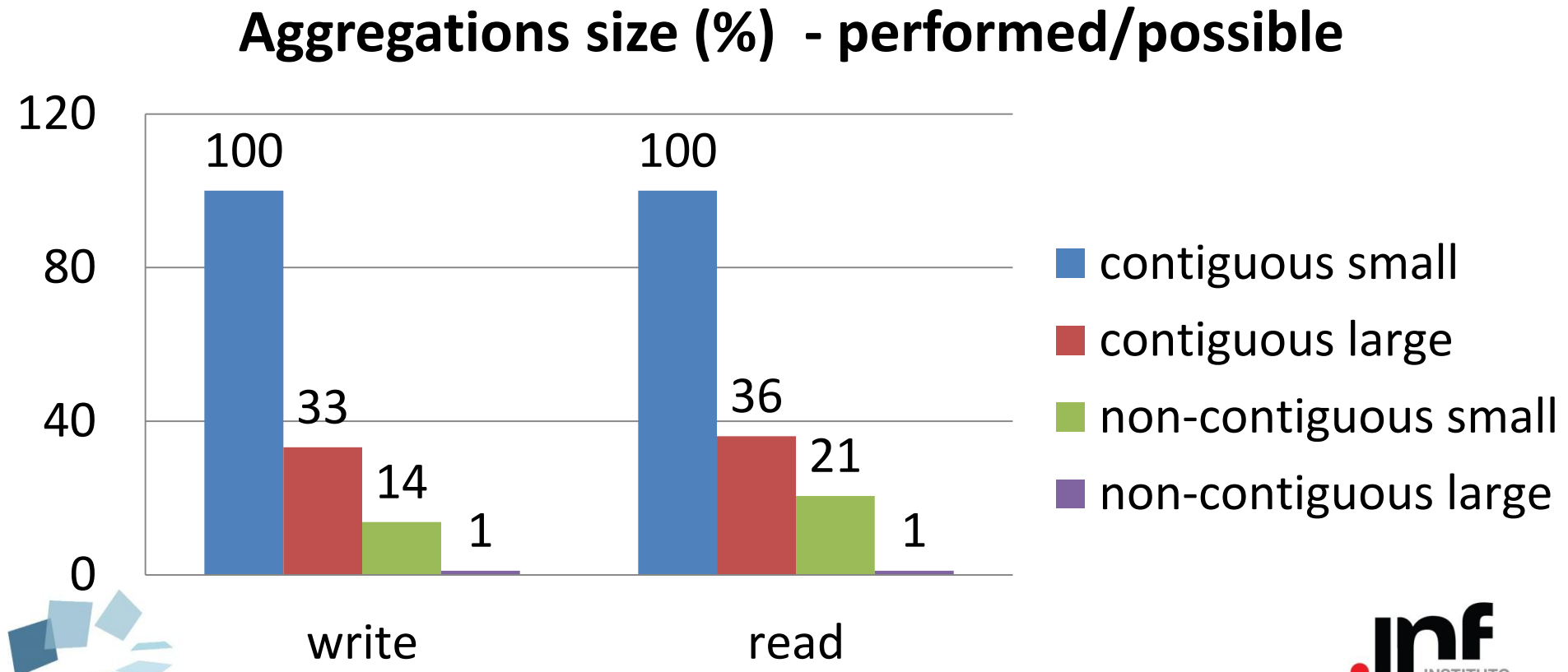
(it will just wait for a little time and move on)

I/O Scheduling with LibaIOLi

Average aggregation: 2.4 (write) or 2.8 (read) reqs

I/O Scheduling with LibaIOLi

Average aggregation: 2.4 (write) or 2.8 (read) reqs



So **how** could we

aggregate more?

Application-aware I/O Scheduling in the Parallel File System Server Side

L I G

INSTITUTO DE INFORMÁTICA UFRGS

Application-aware I/O Scheduling

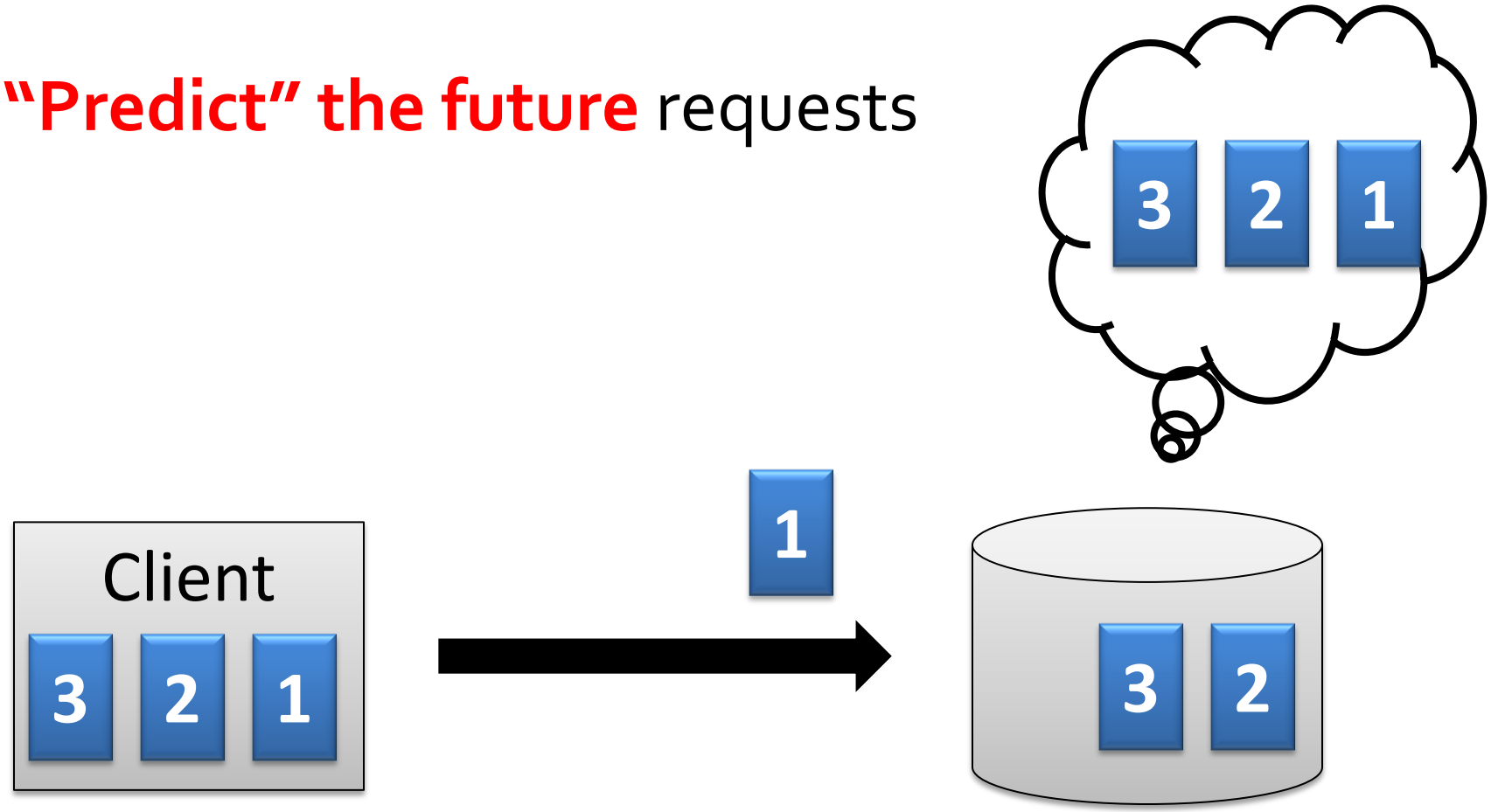
- LibaIOLi + **information about the application**
  - Scheduler takes better decisions
  - Better aggregations

Application-aware I/O Scheduling

- **"Predict" the future** requests

Application-aware I/O Scheduling

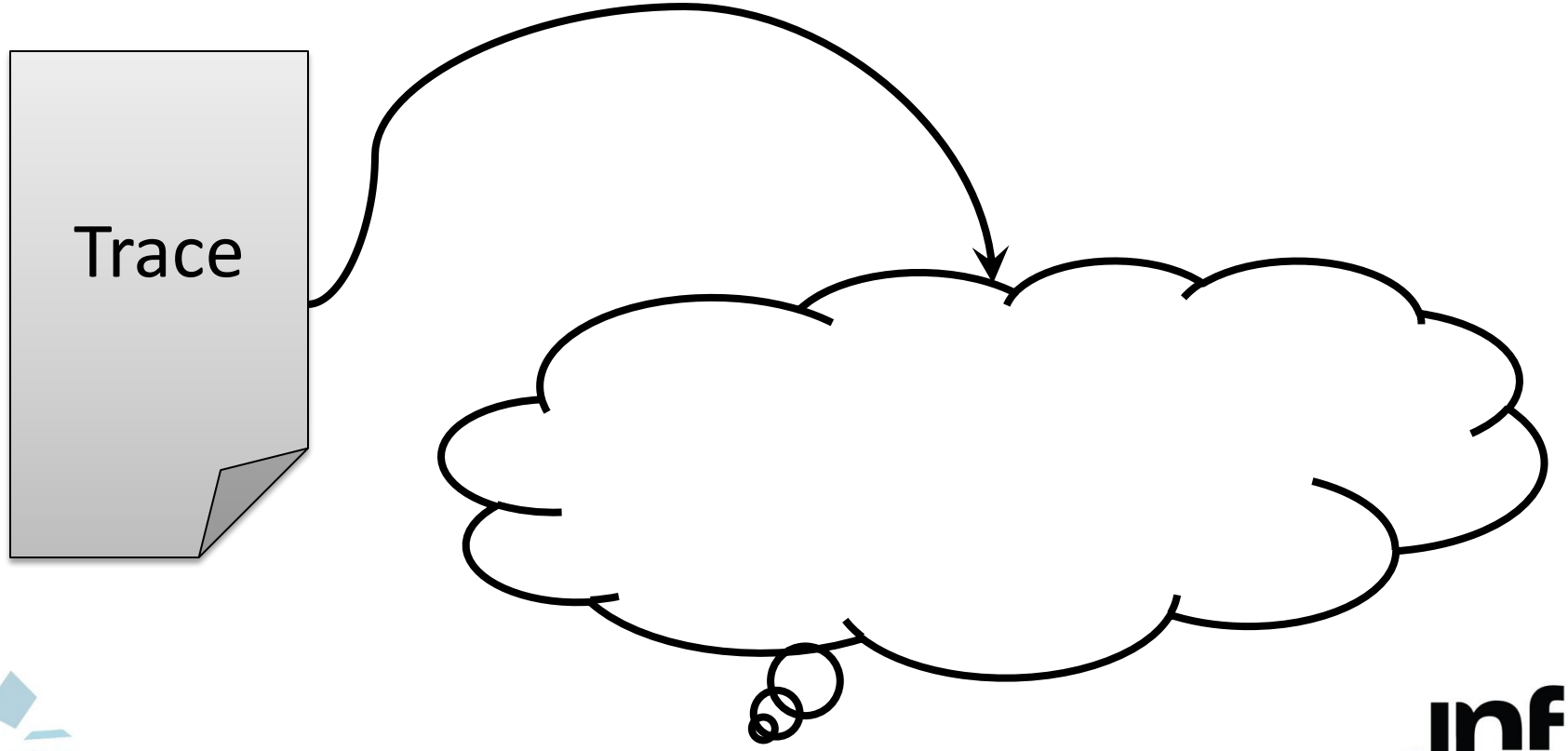
- **"Predict" the future** requests



The first attempt

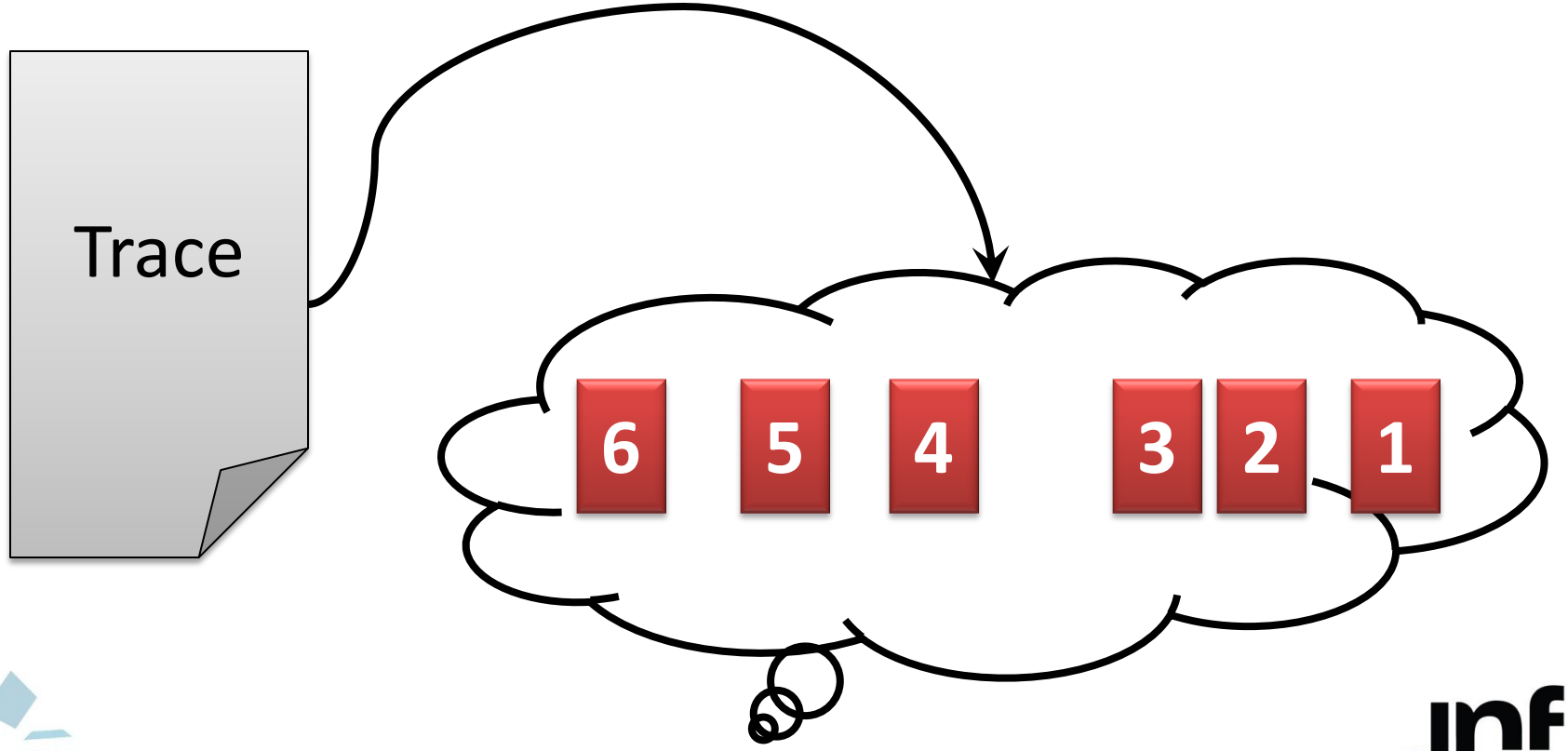
The first attempt

- "predict" = obtain from **traces**



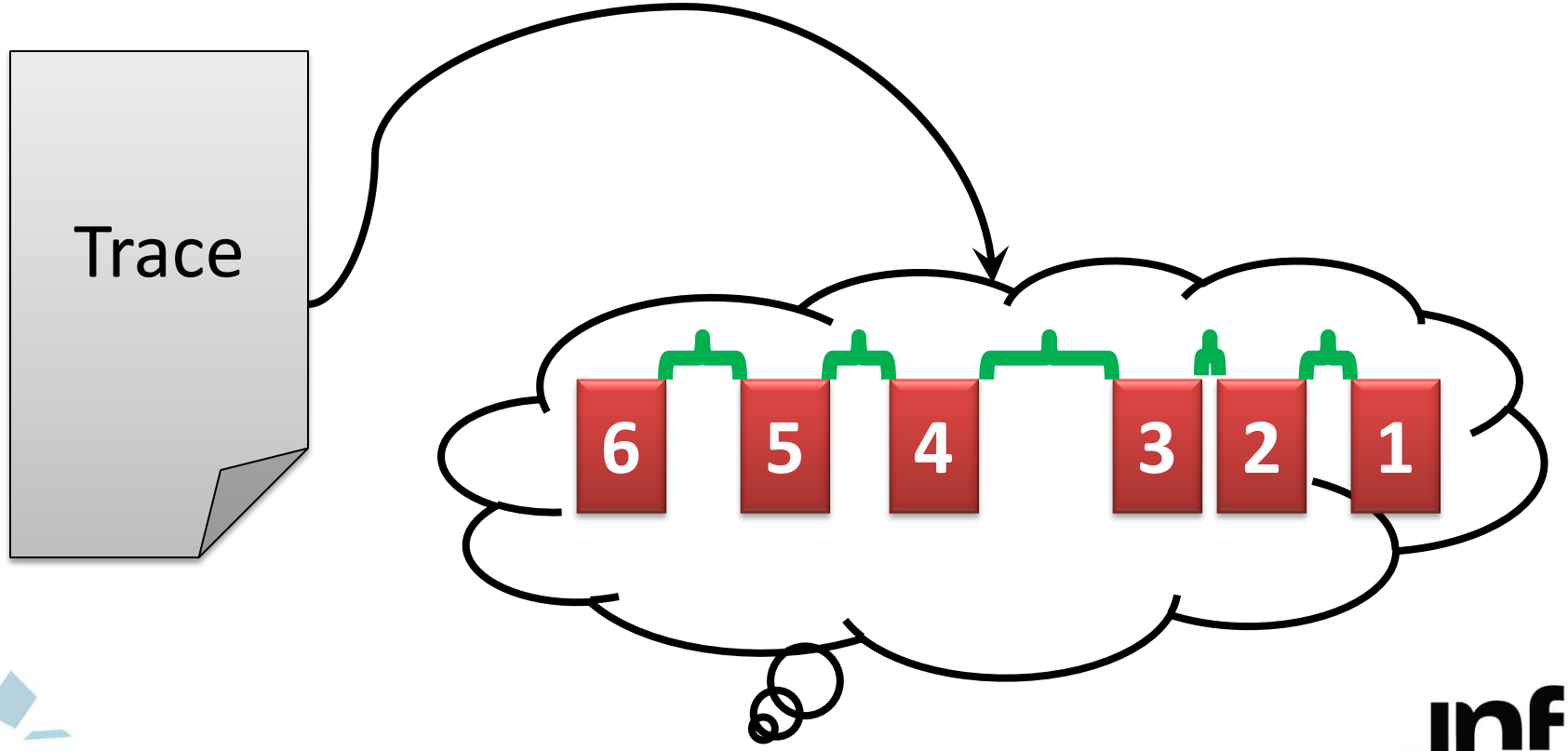
The first attempt

- **Traces -> requests**



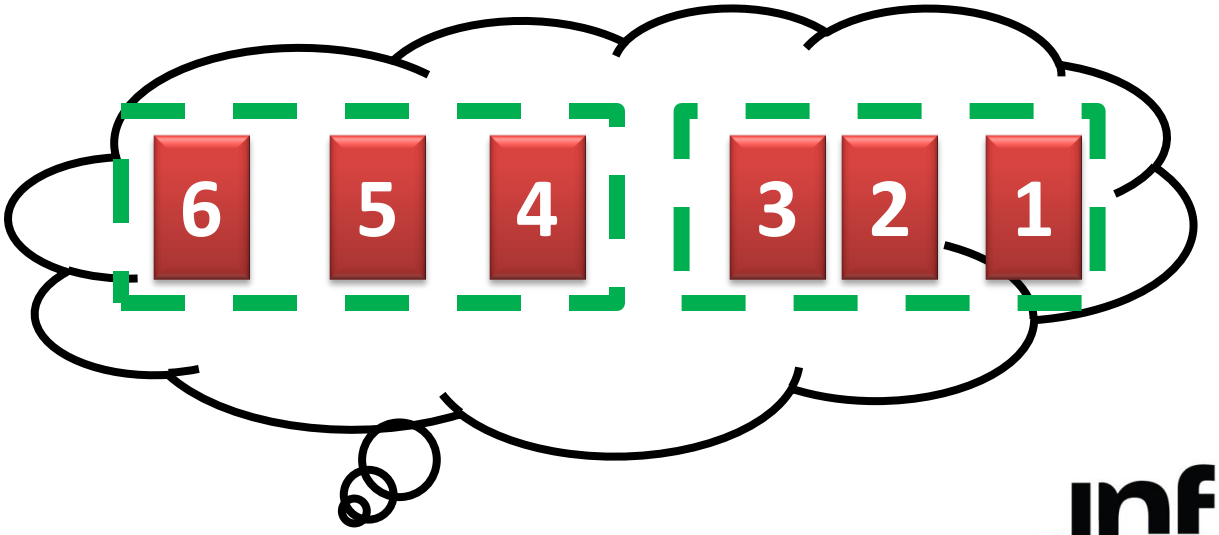
The first attempt

- Traces -> requests and **time between them**



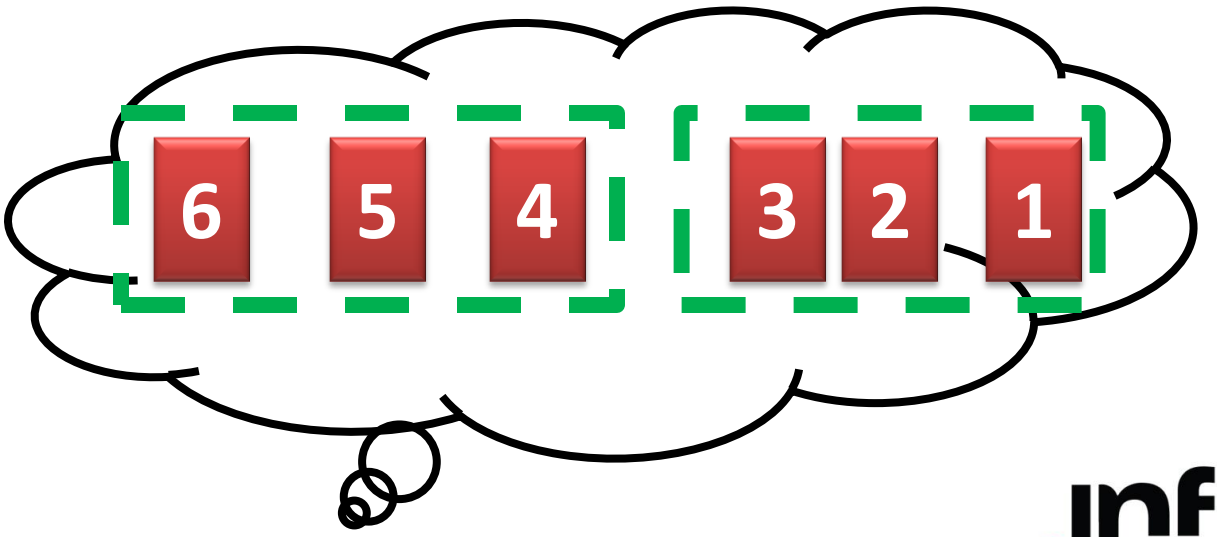
The first attempt

- **-> predicted aggregations**

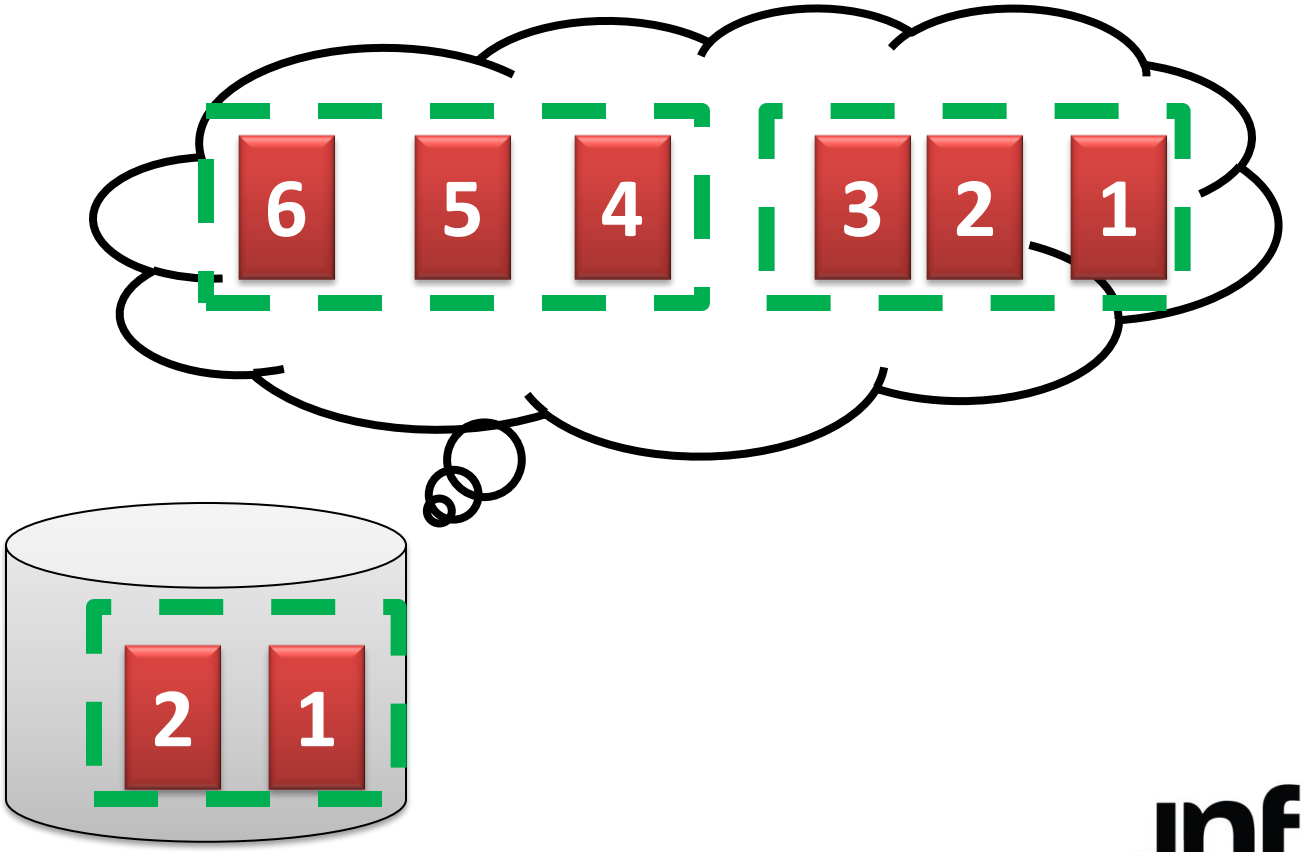


The first attempt

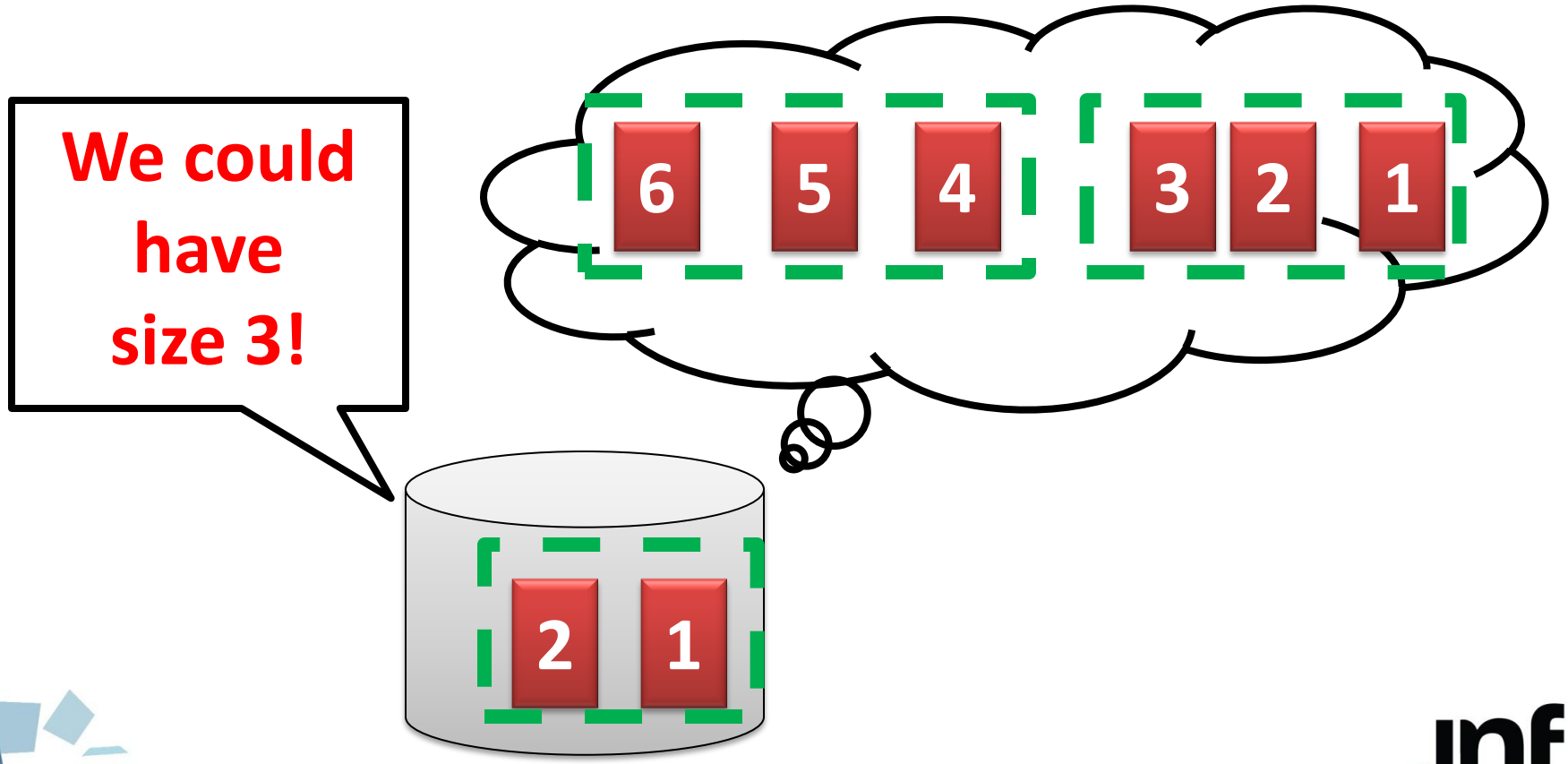
- -> predicted aggregations
  - Benchmarked **time to process a request** of size N
  - **Time between requests**



The first attempt



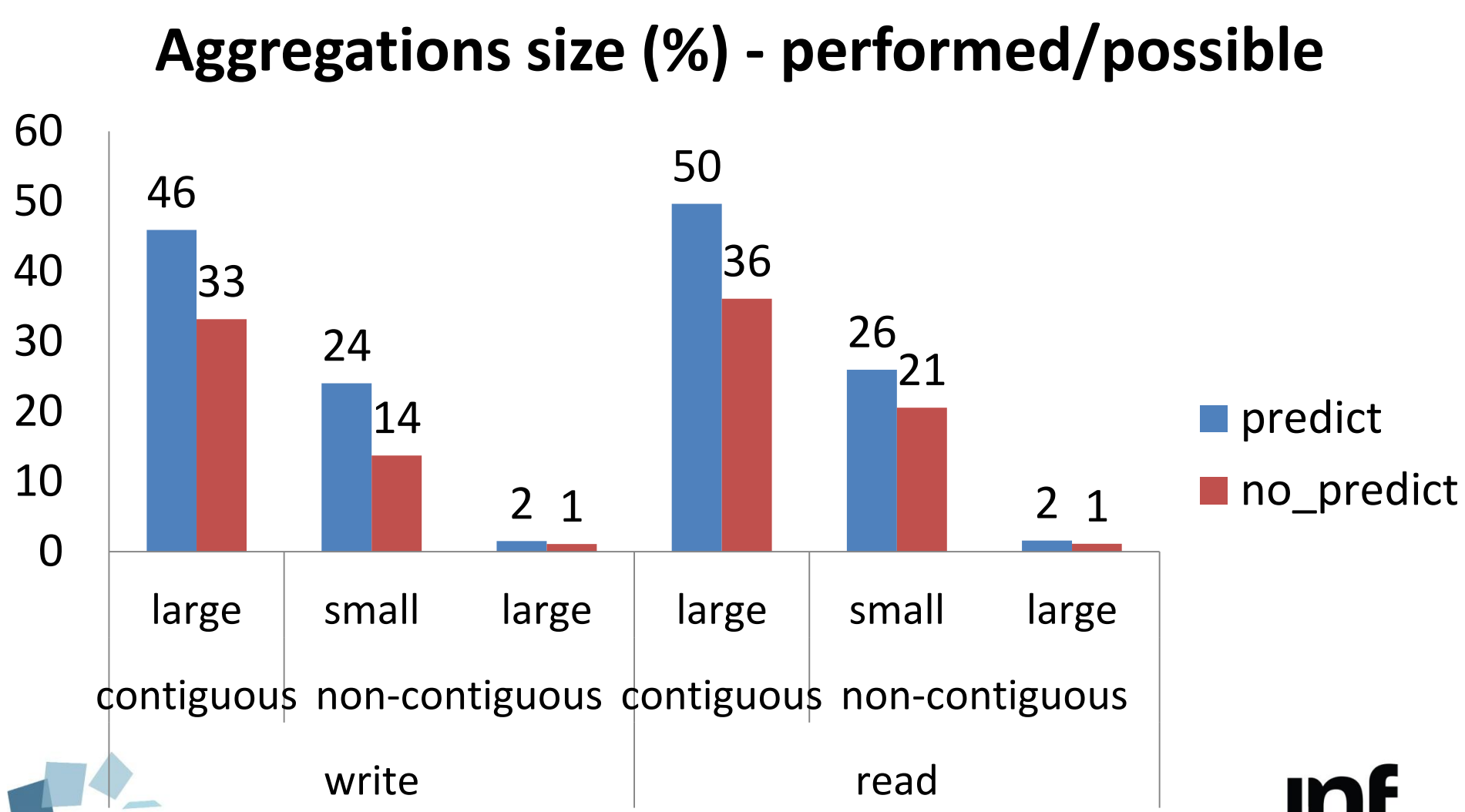
The first attempt



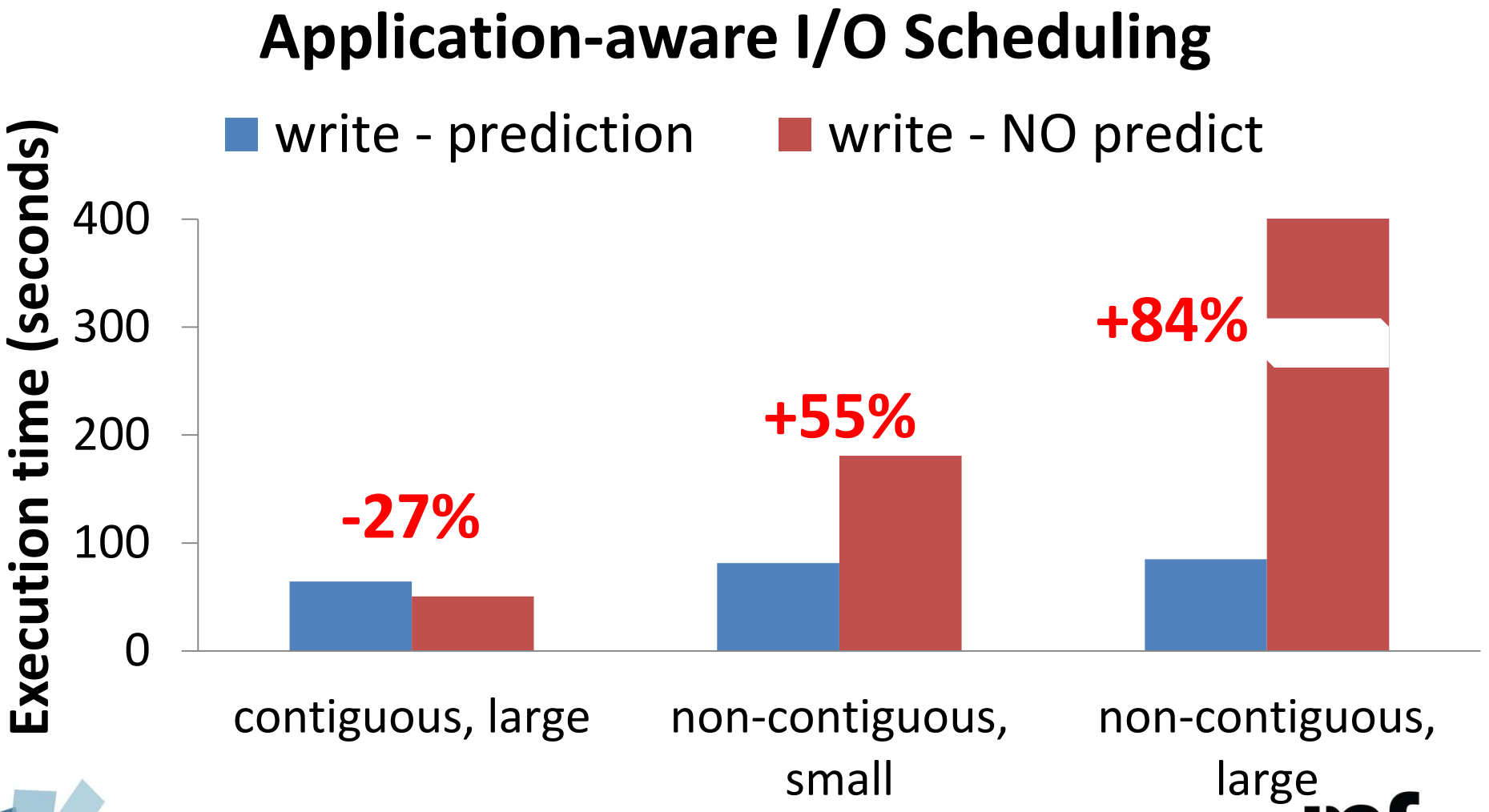
The first attempt

- **Aggregation size** goes to 3.8 (write) or 4.1 (read)
  - Increase of **58% (write) or 46% (read)**

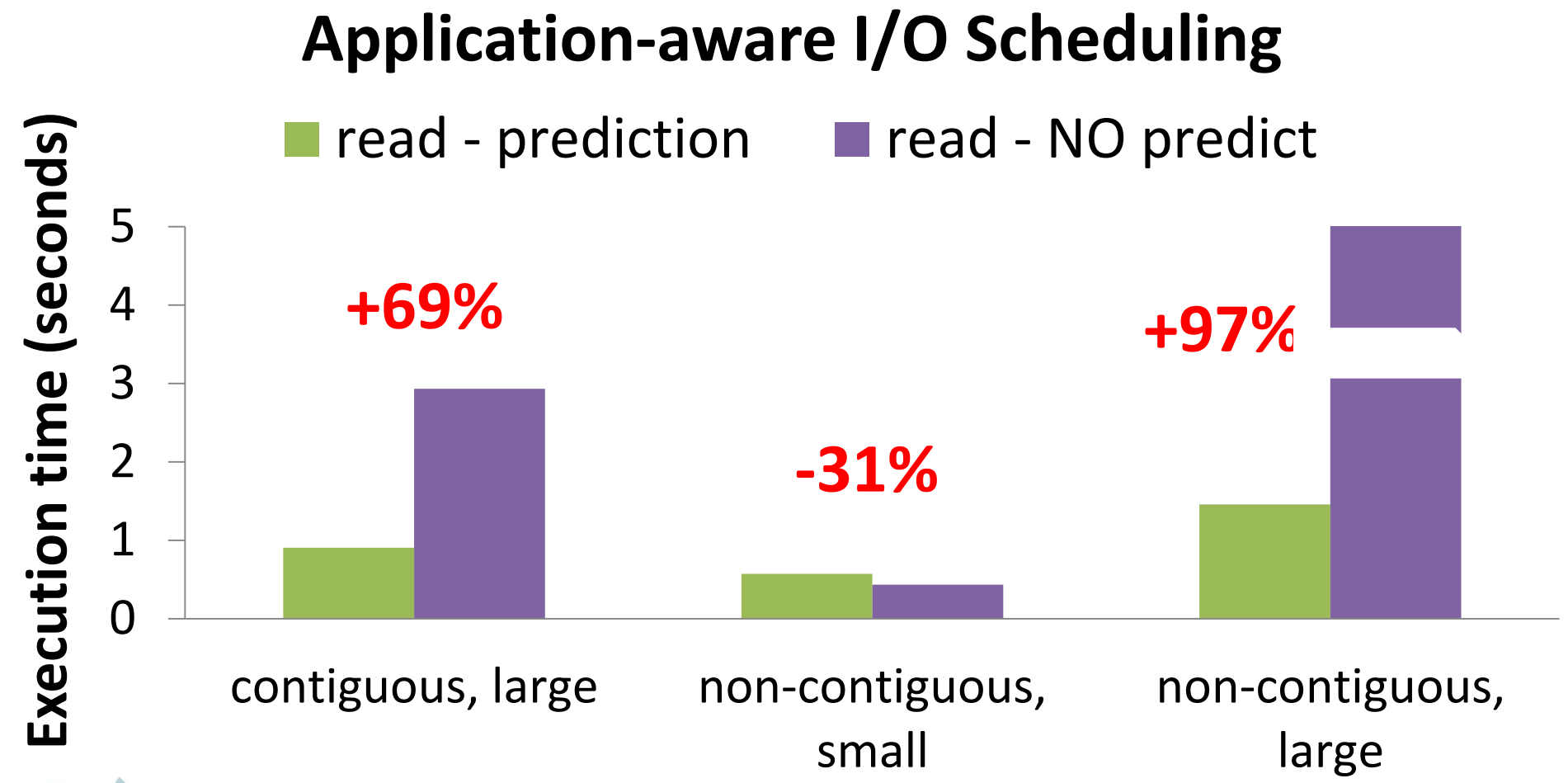
The first attempt



The first attempt



The first attempt



Application-aware I/O Scheduling in the Parallel File System Server Side

Summarizing...

Summarizing

I/O Scheduling with LibaIOLi (library for PFS)

Summarizing

I/O Scheduling with **LibaIOLi** (library for PFS)

Reordering and aggregation of requests

Summarizing

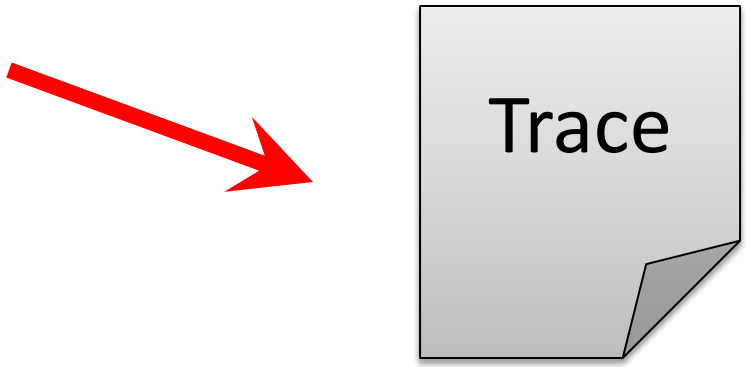
I/O Scheduling with **LibaIOLi** (library for PFS)

Reordering and **aggregation of requests**

Most of the increase in performance

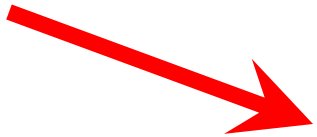
Summarizing

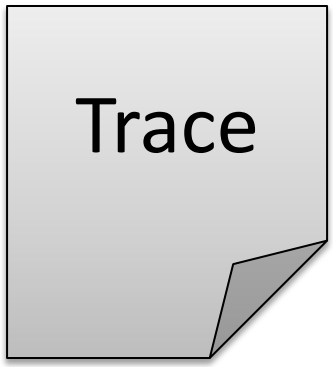
Application execution



Summarizing

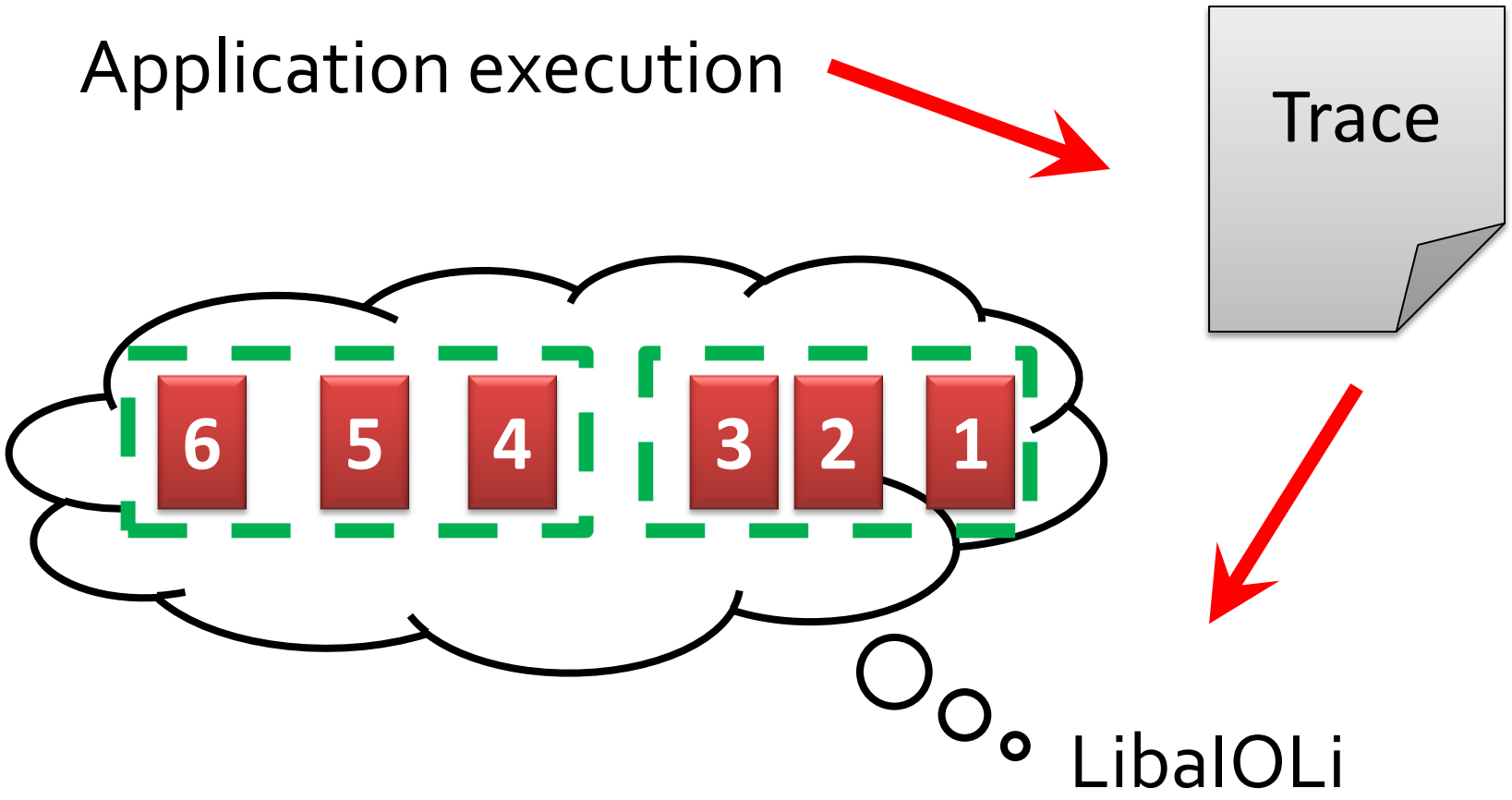
Application execution



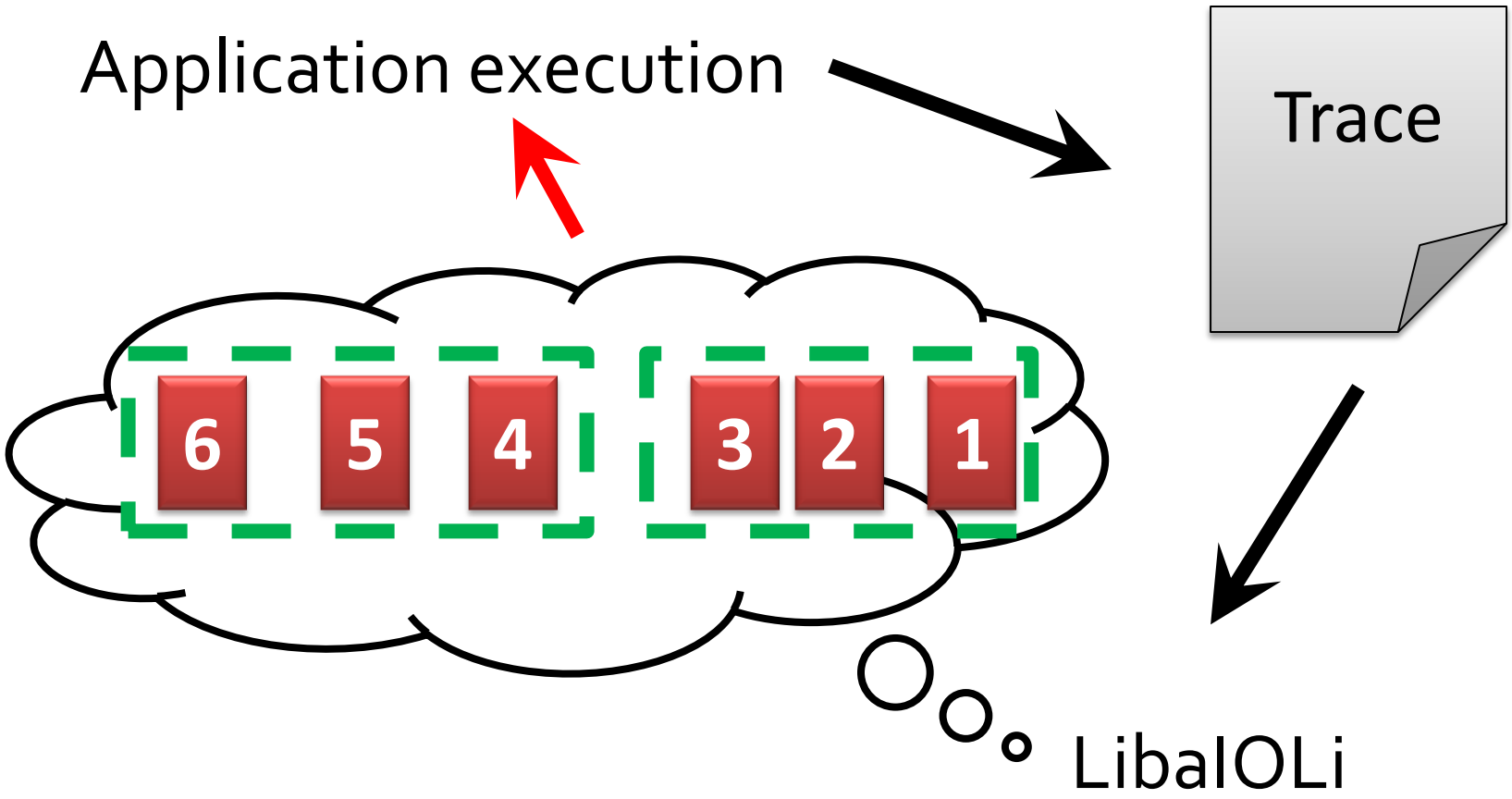


LibaIOLi

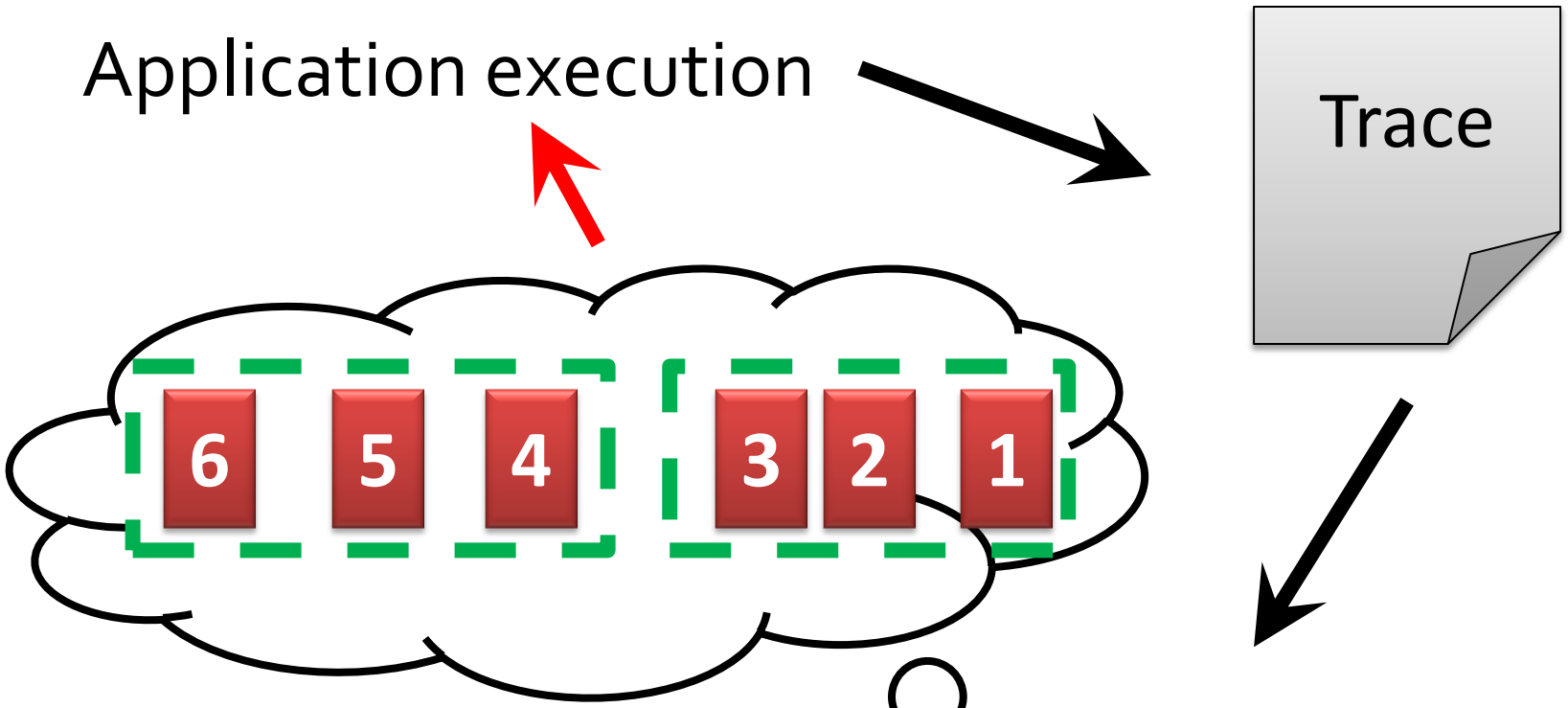
Summarizing



Summarizing



Summarizing

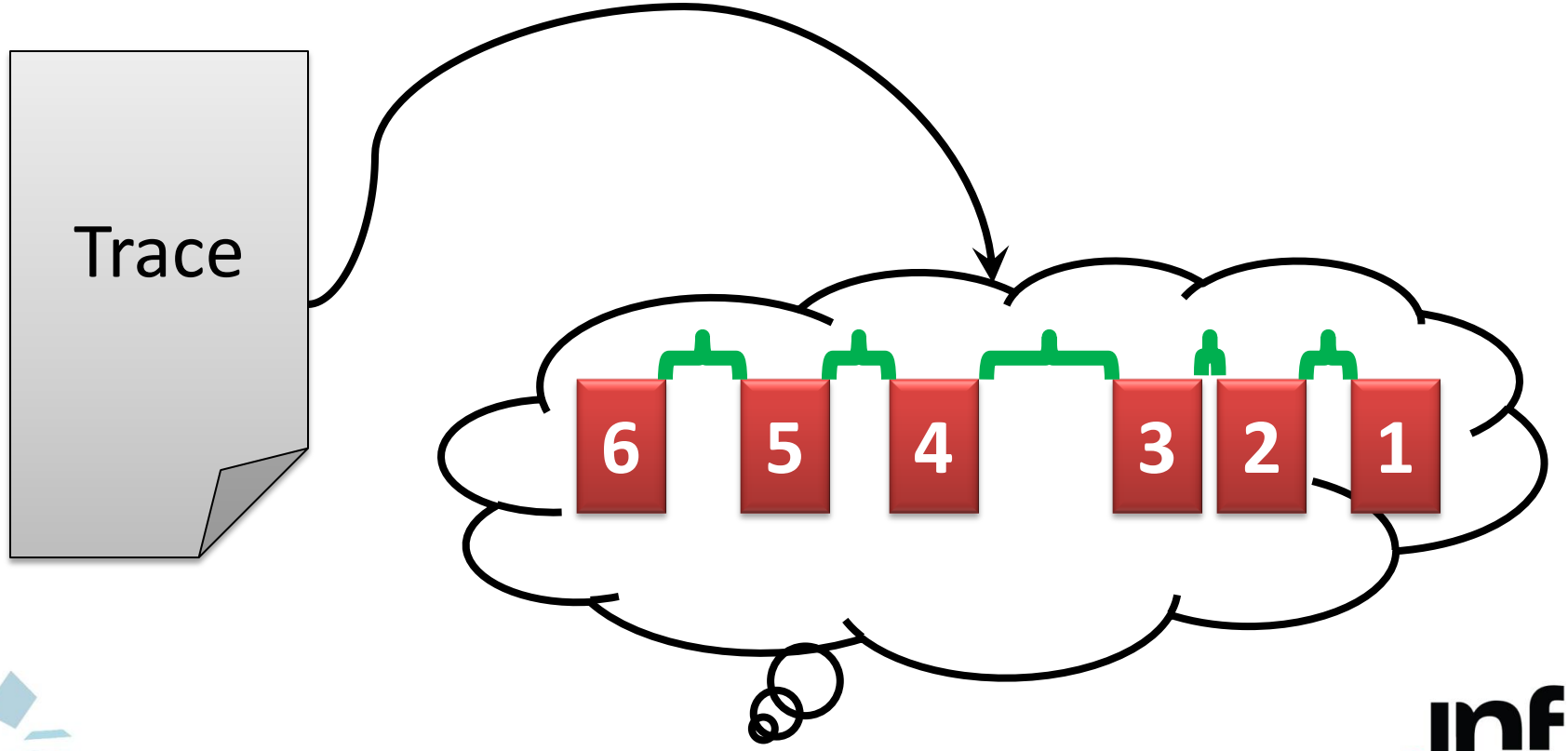


Up to ~97% better

What's next?

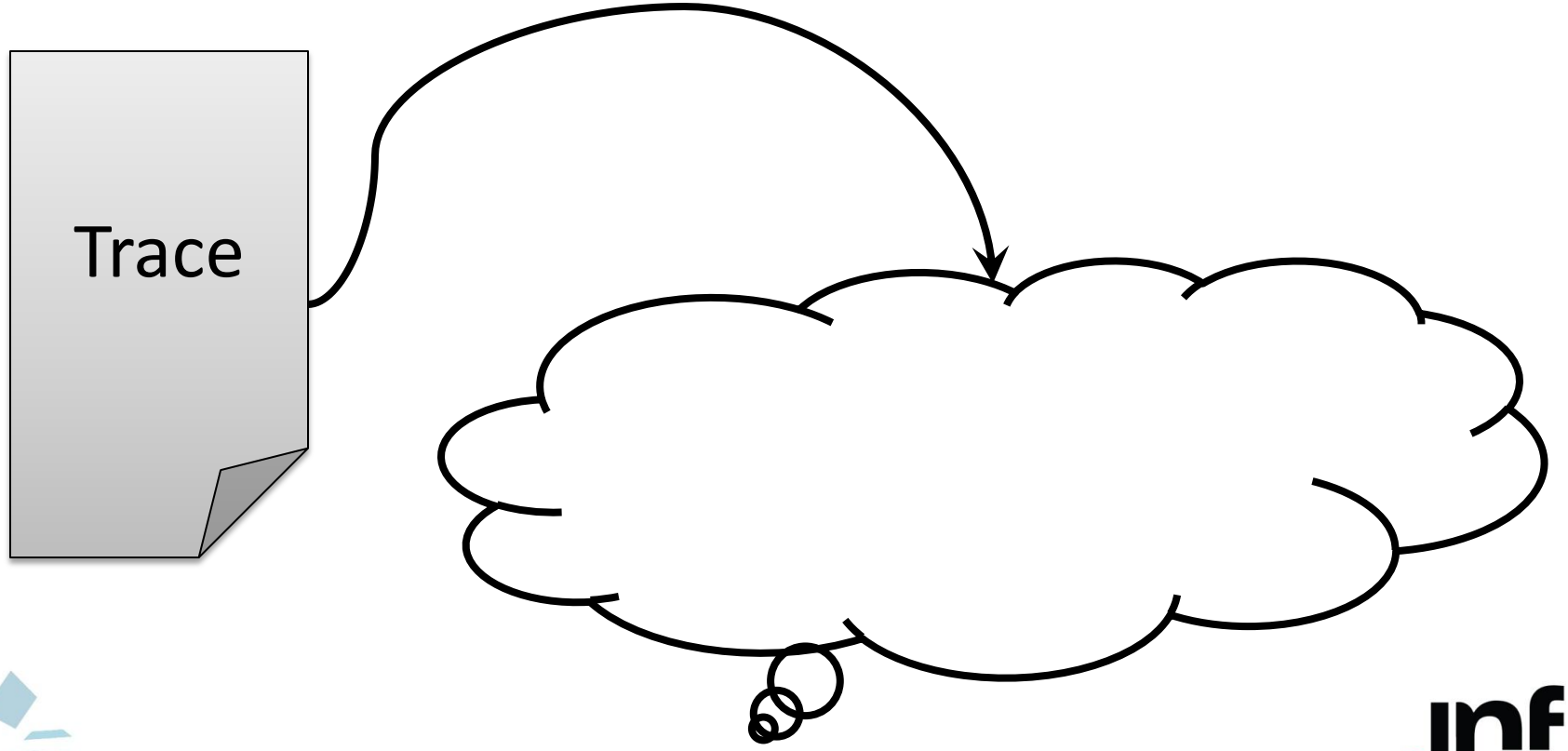
Future Work

- Detection of **access pattern**



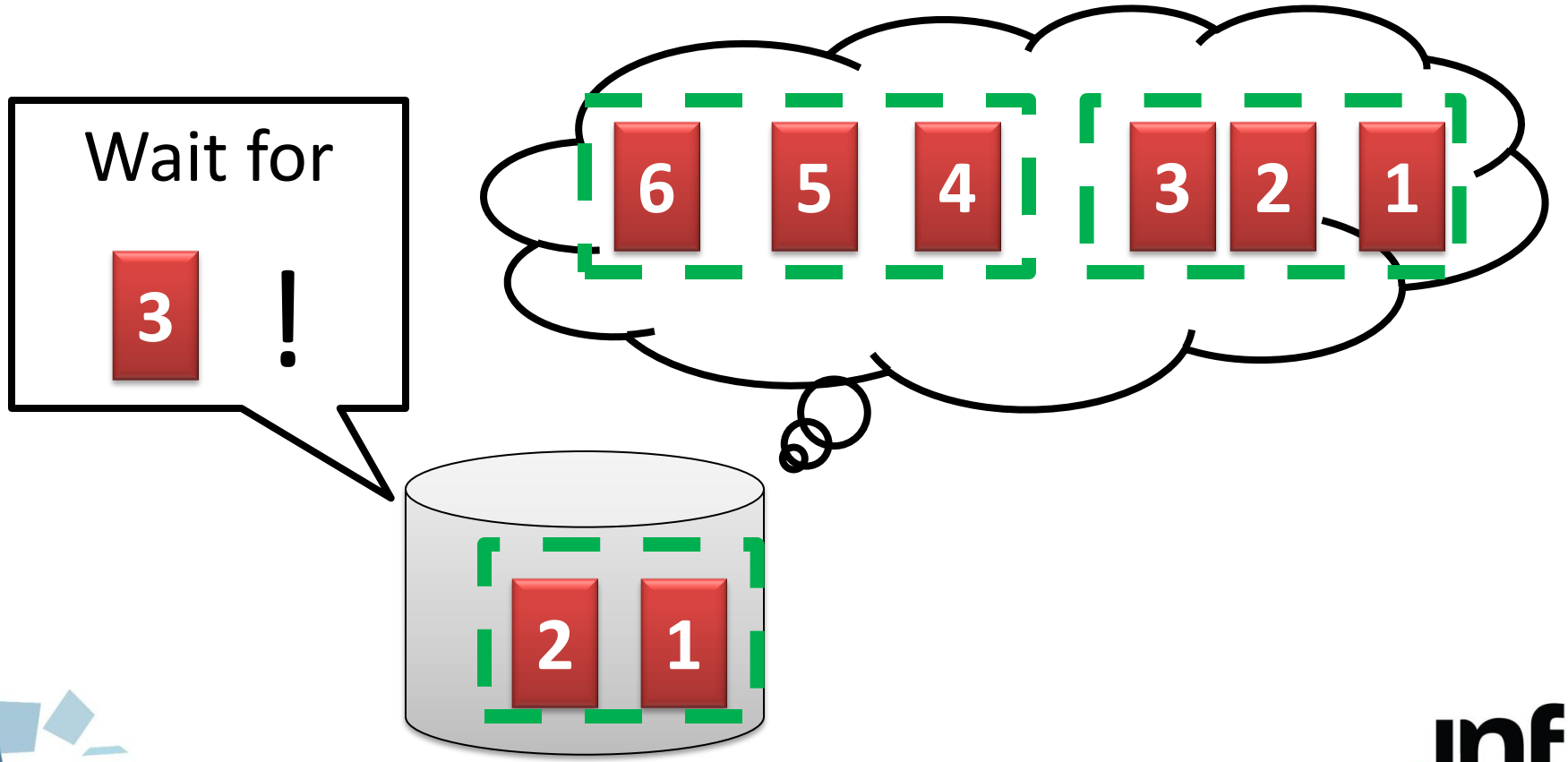
Future Work

- Use **Damaris** to obtain the information



Future Work

- **More "aggressive"** approach



Future Work

- **Further analysis**

- LibaIOLi with other PFS (PVFS, Lustre, … ?)

Application-aware I/O Scheduling in the Parallel File System Server Side

Thank you for your attention!

Francieli Zanon Boito

francieli.zanon@inf.ufrgs.br

GPPD - II - **Federal University of Rio Grande do Sul** (UFRGS), Brazil

INRIA – LIG – **Grenoble University**, France

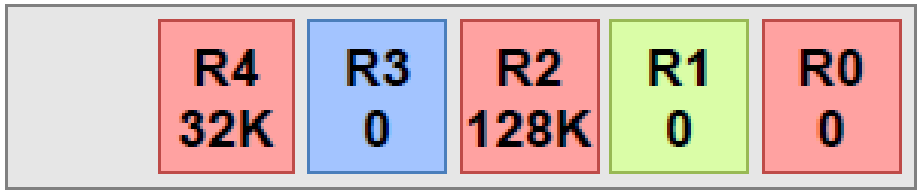
I/O Scheduling Example: aIOLi

[Lebre et al. 2006]

- Variation of **Multilevel Feedback** (MLF) algorithm

- Used in **Lustre NRS**

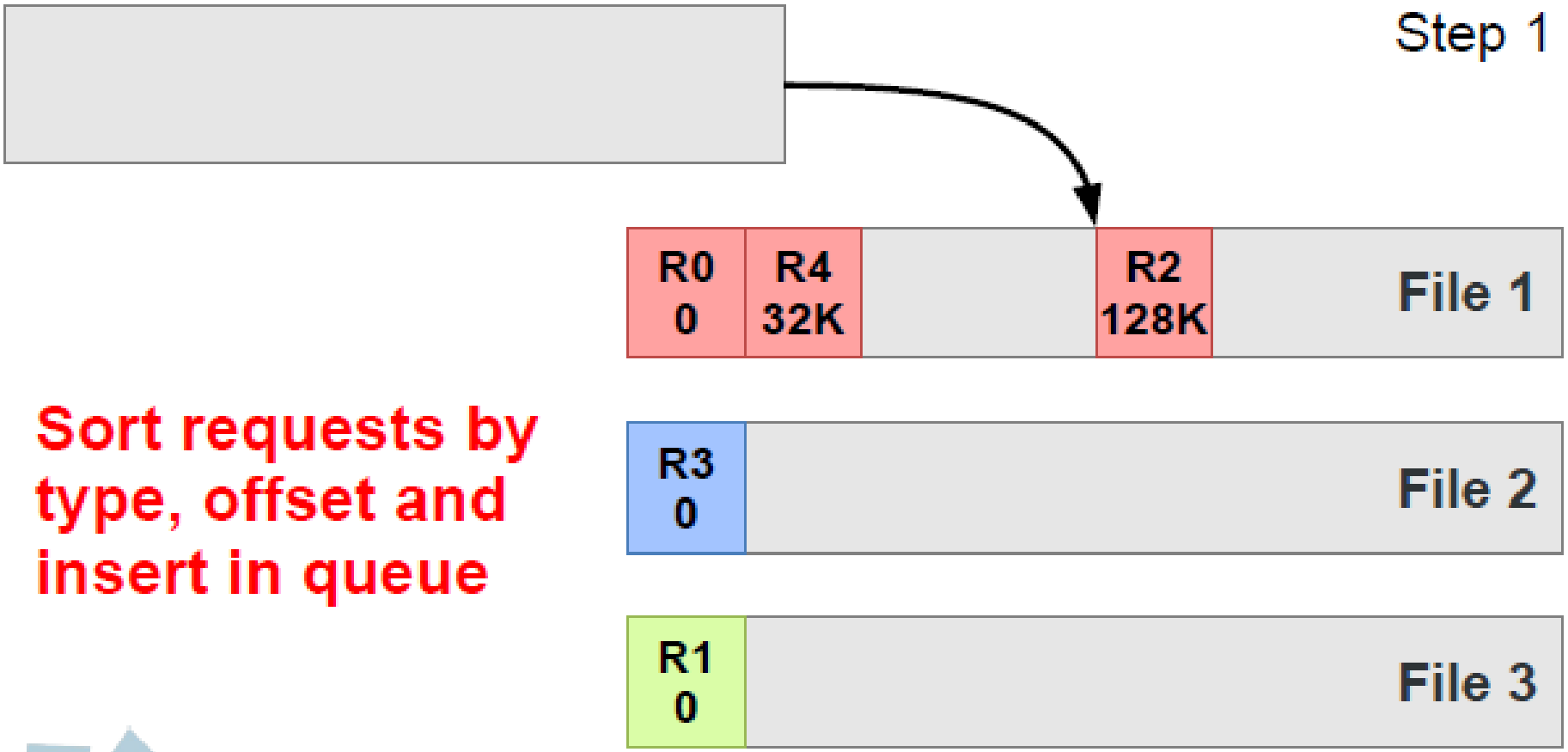
I/O Scheduling Example: aIOLi



Requests of 32KB offset

Step 1

I/O Scheduling Example: aIOLi

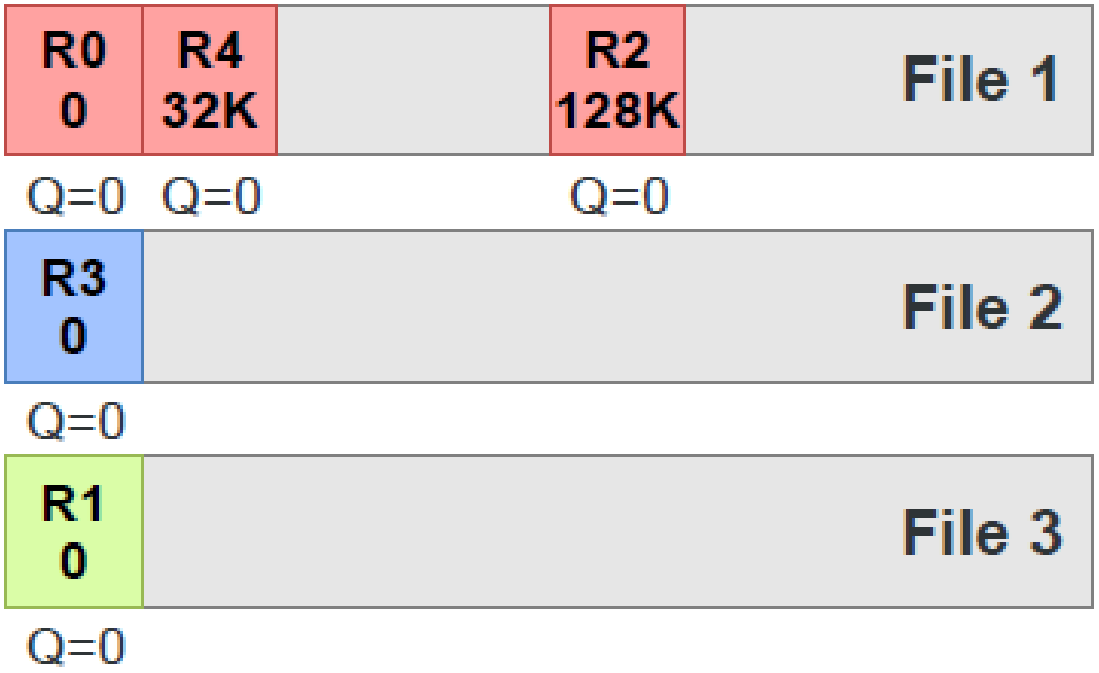


Sort requests by type, offset and insert in queue

I/O Scheduling Example: aIOLi

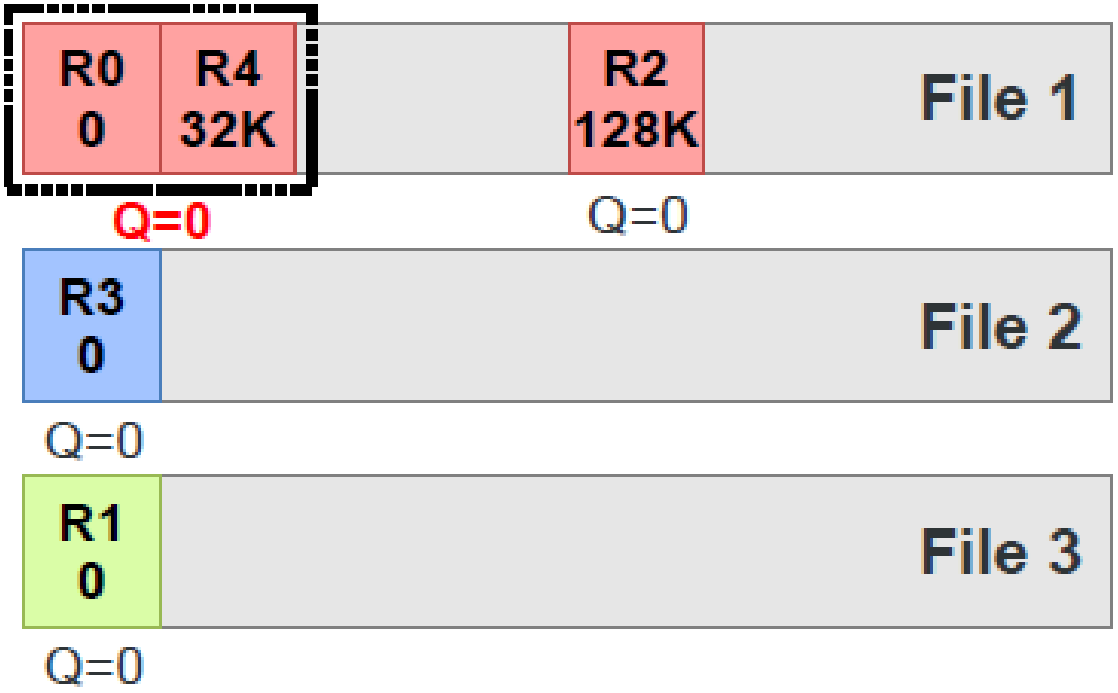
Step 1

Quantum = 0



I/O Scheduling Example: aIOLi

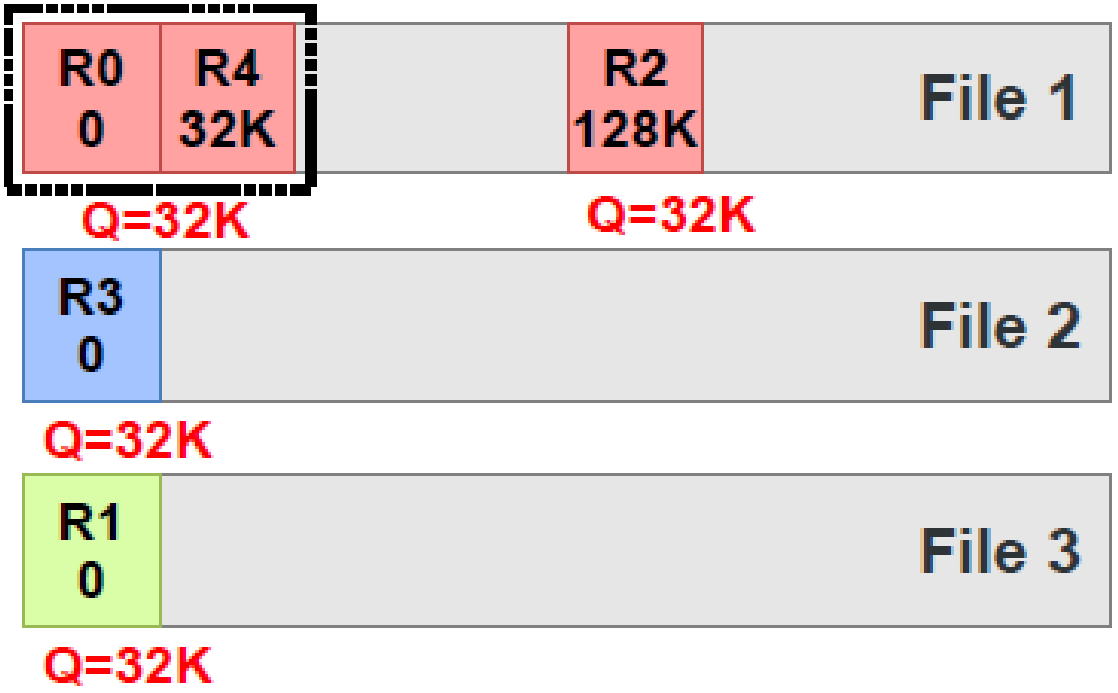
Step 1



Perform aggregations

I/O Scheduling Example: aIOLi

Step 1



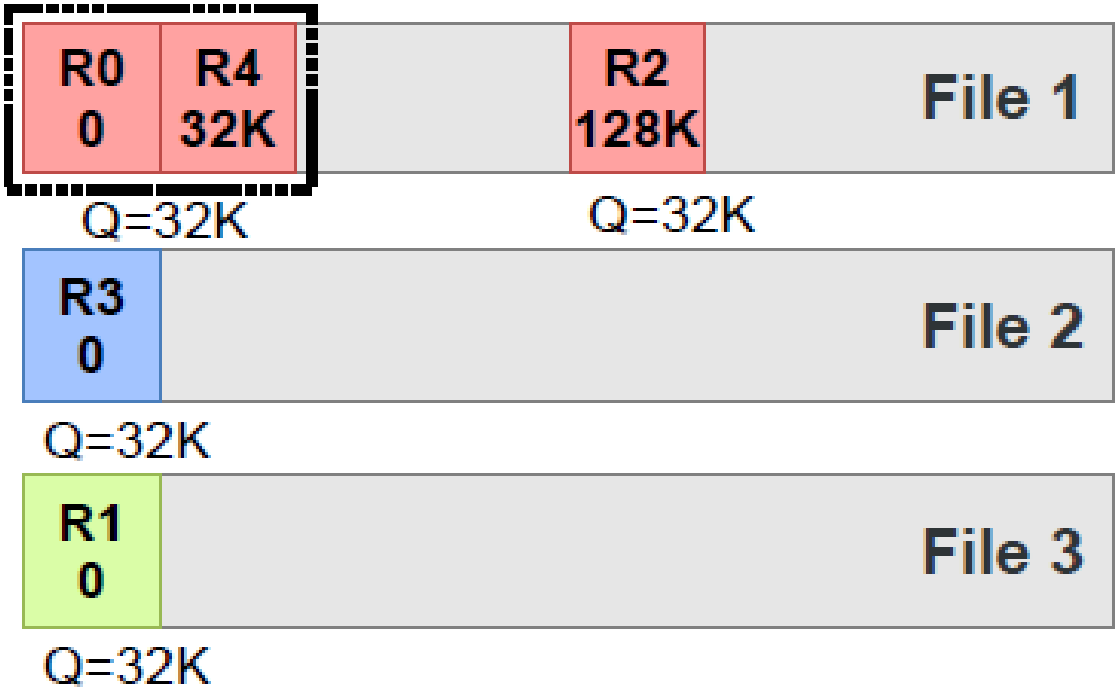
Quantum is increased by a fixed value

I/O Scheduling Example: aIOLi

Step 1

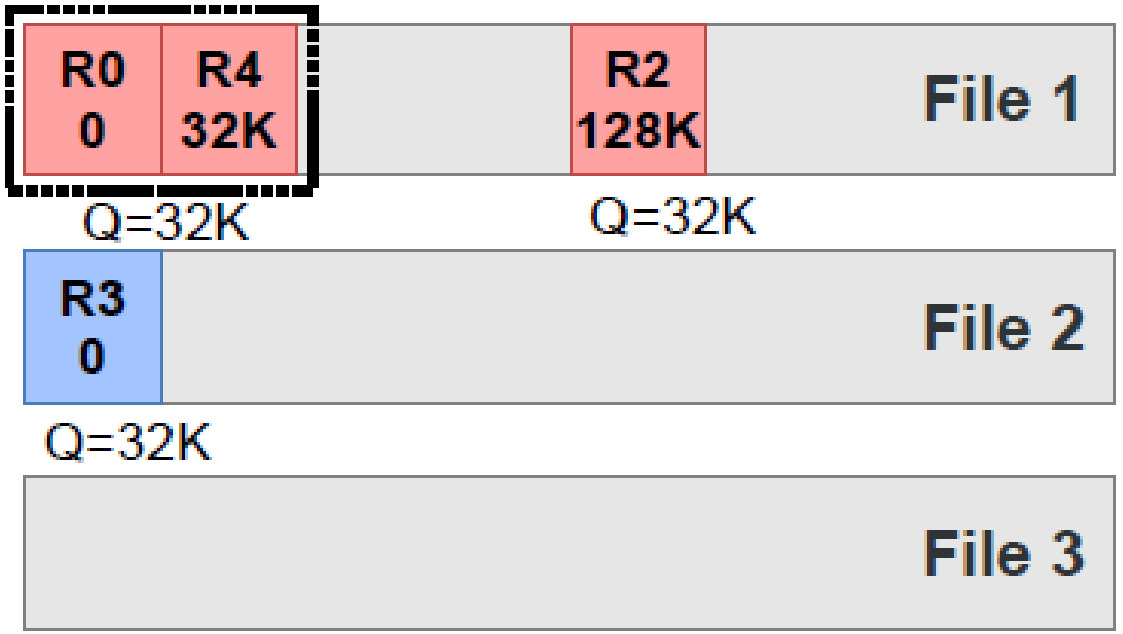
Select request

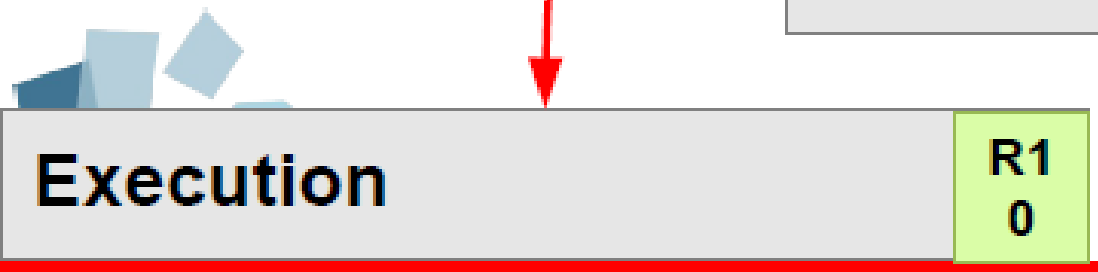
- offset order
- FIFO
- quantum enough



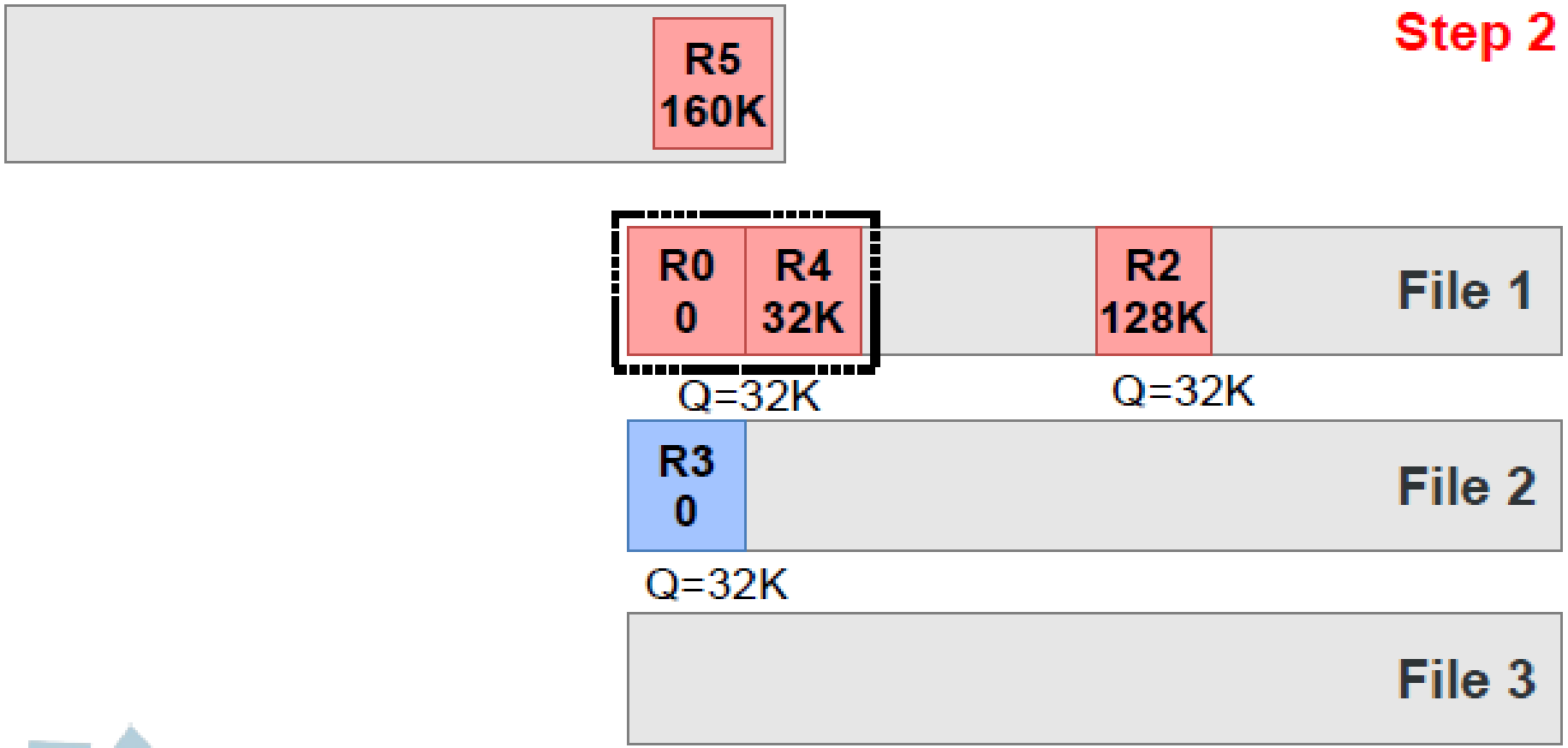
I/O Scheduling Example: aIOLi

Step 1

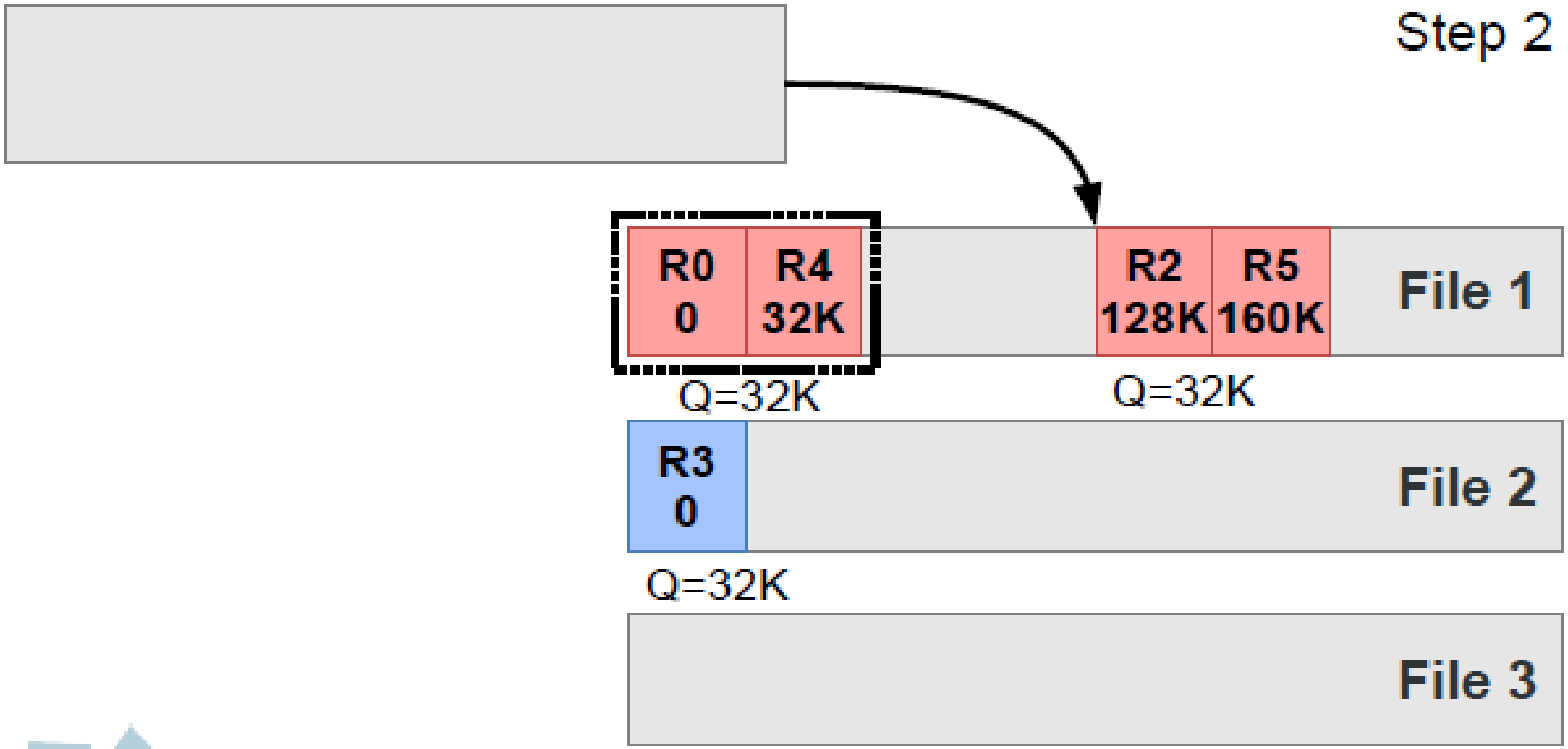




I/O Scheduling Example: aIOLi

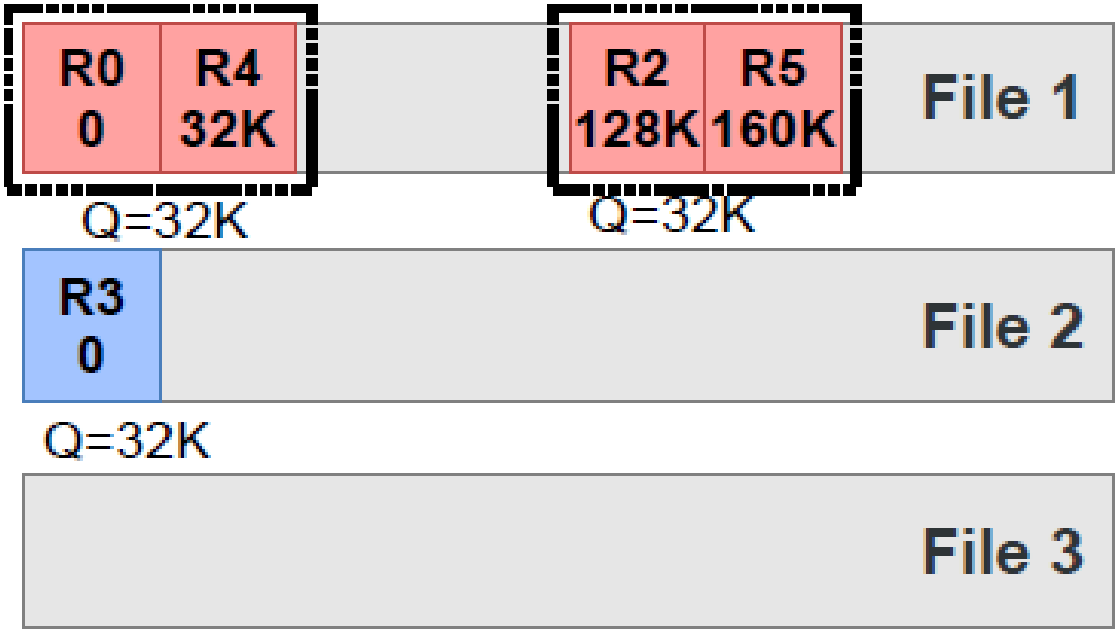


I/O Scheduling Example: aIOLi



I/O Scheduling Example: aIOLi

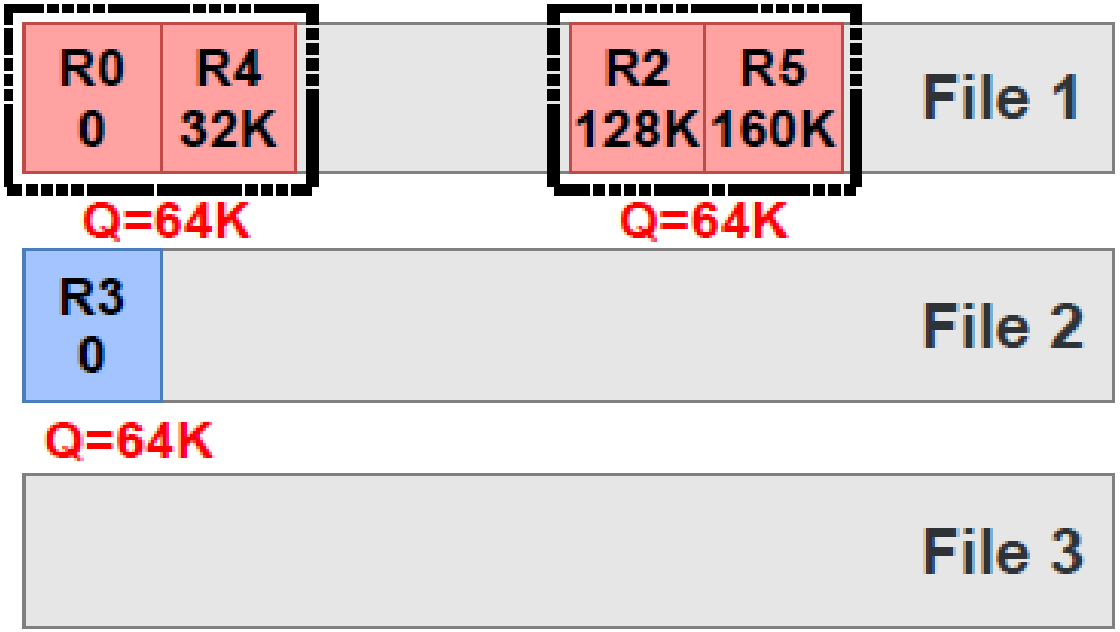
Step 2



Execution | R1 0

I/O Scheduling Example: aIOLi

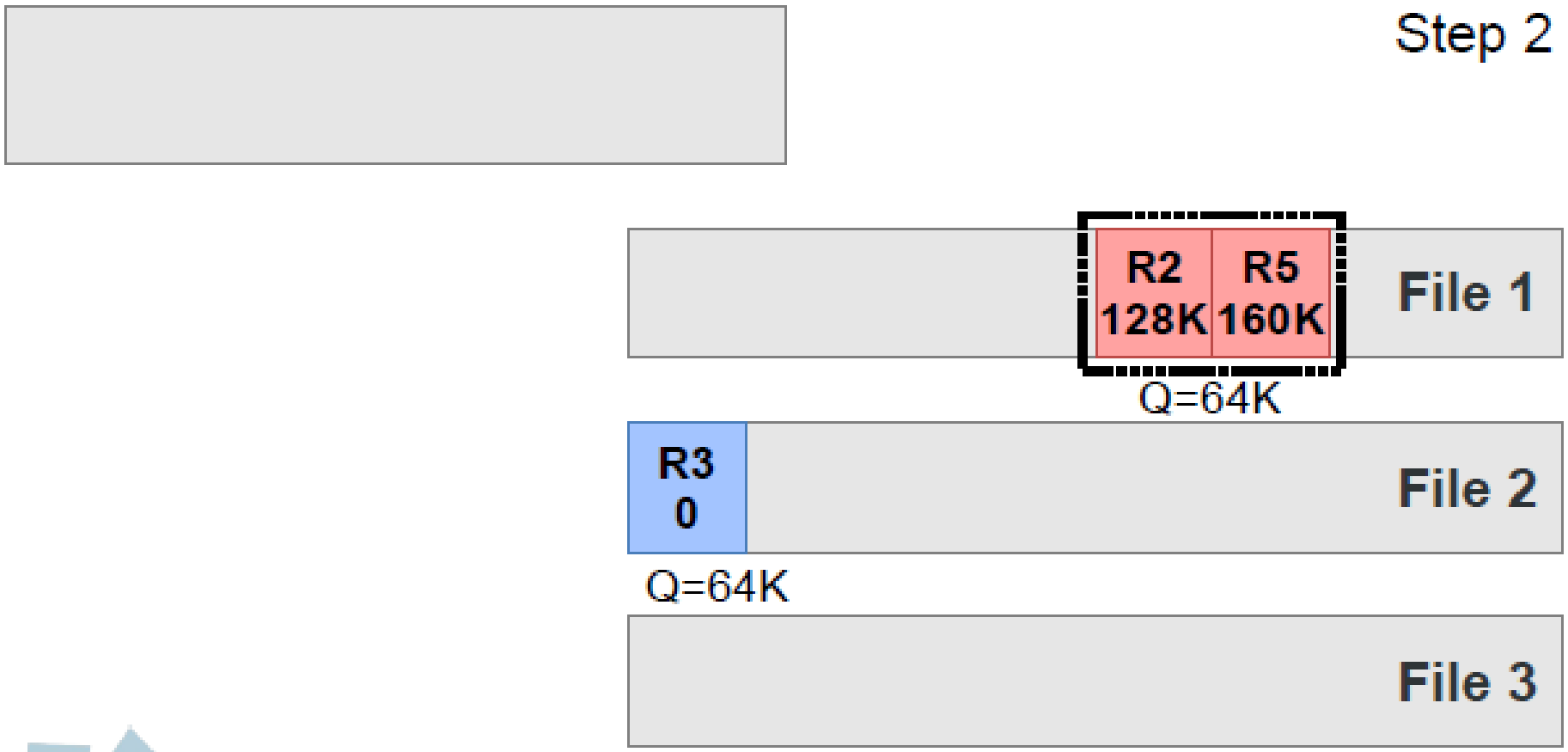
Step 2



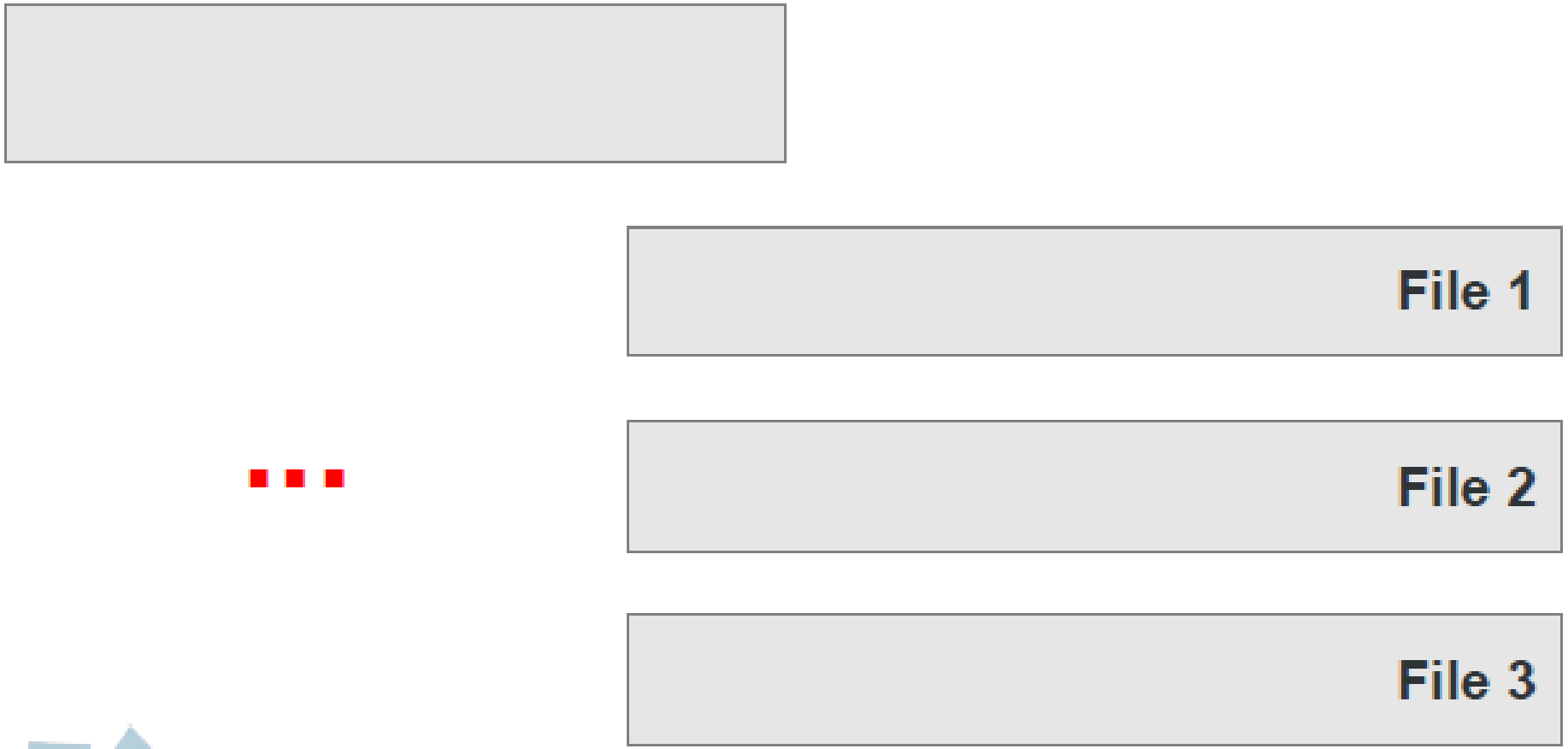
Execution

R1
0

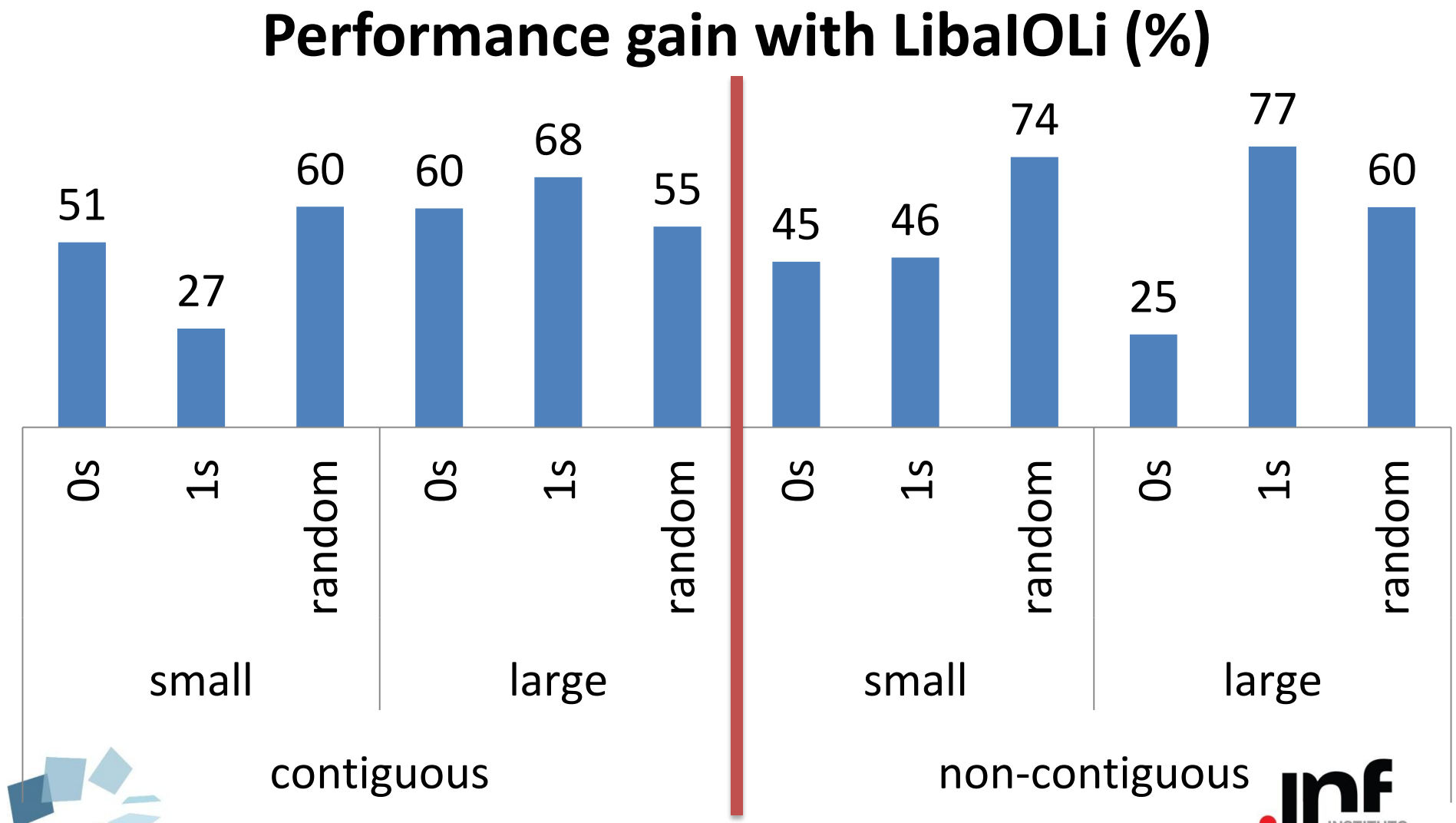
I/O Scheduling Example: aIOLi



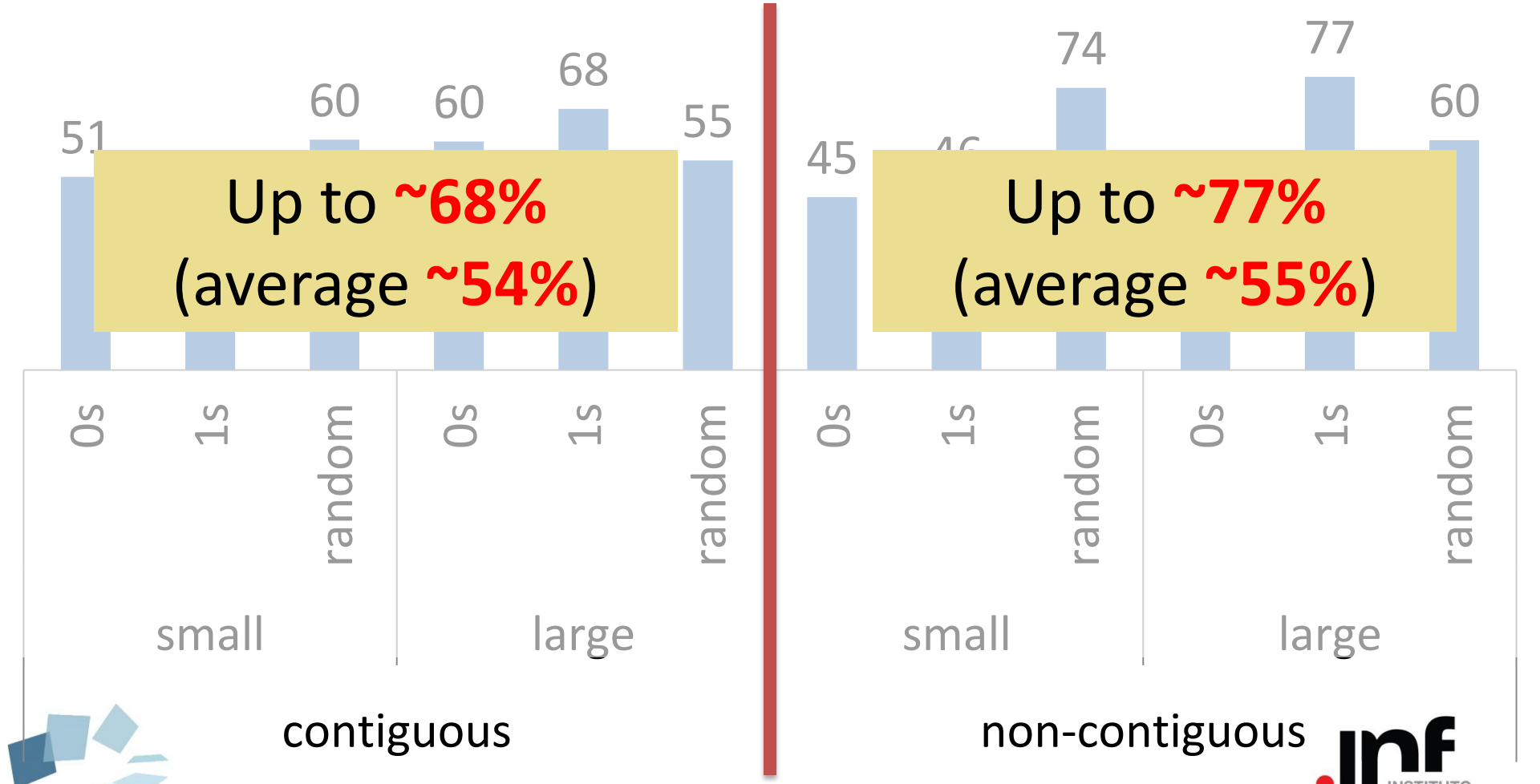
I/O Scheduling Example: aIOLi



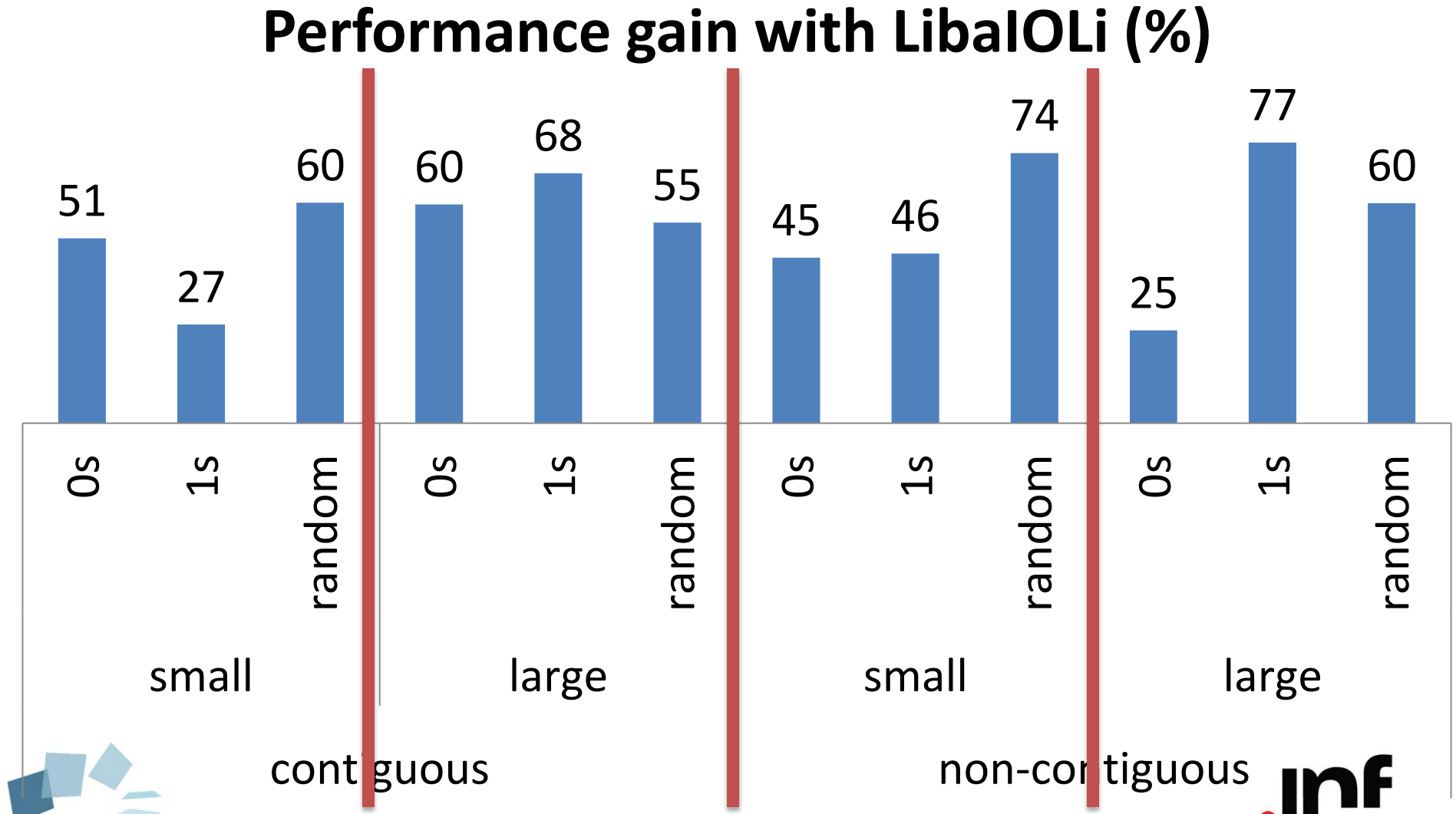
Preliminary Results – Write Operations



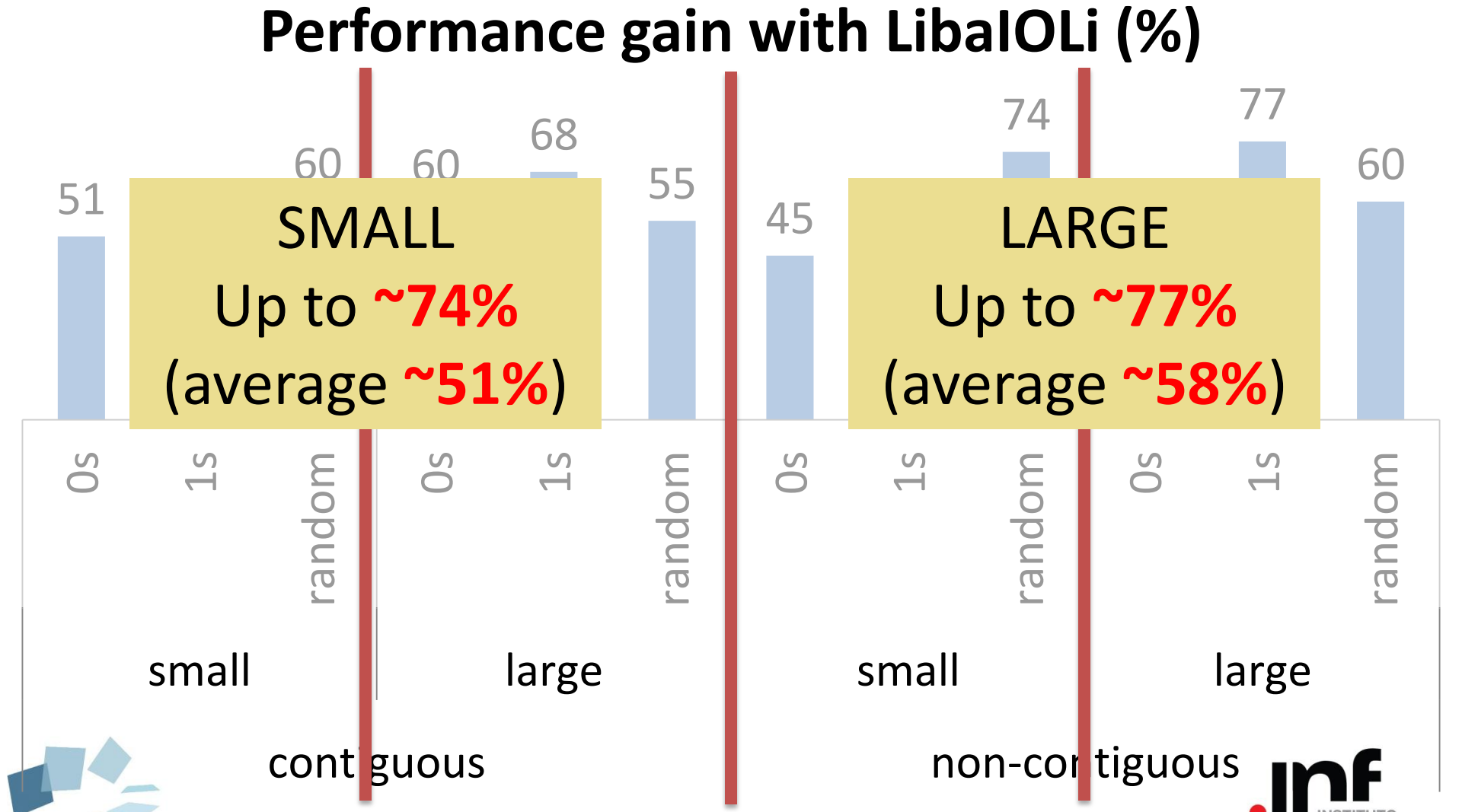
Preliminary Results – Write Operations



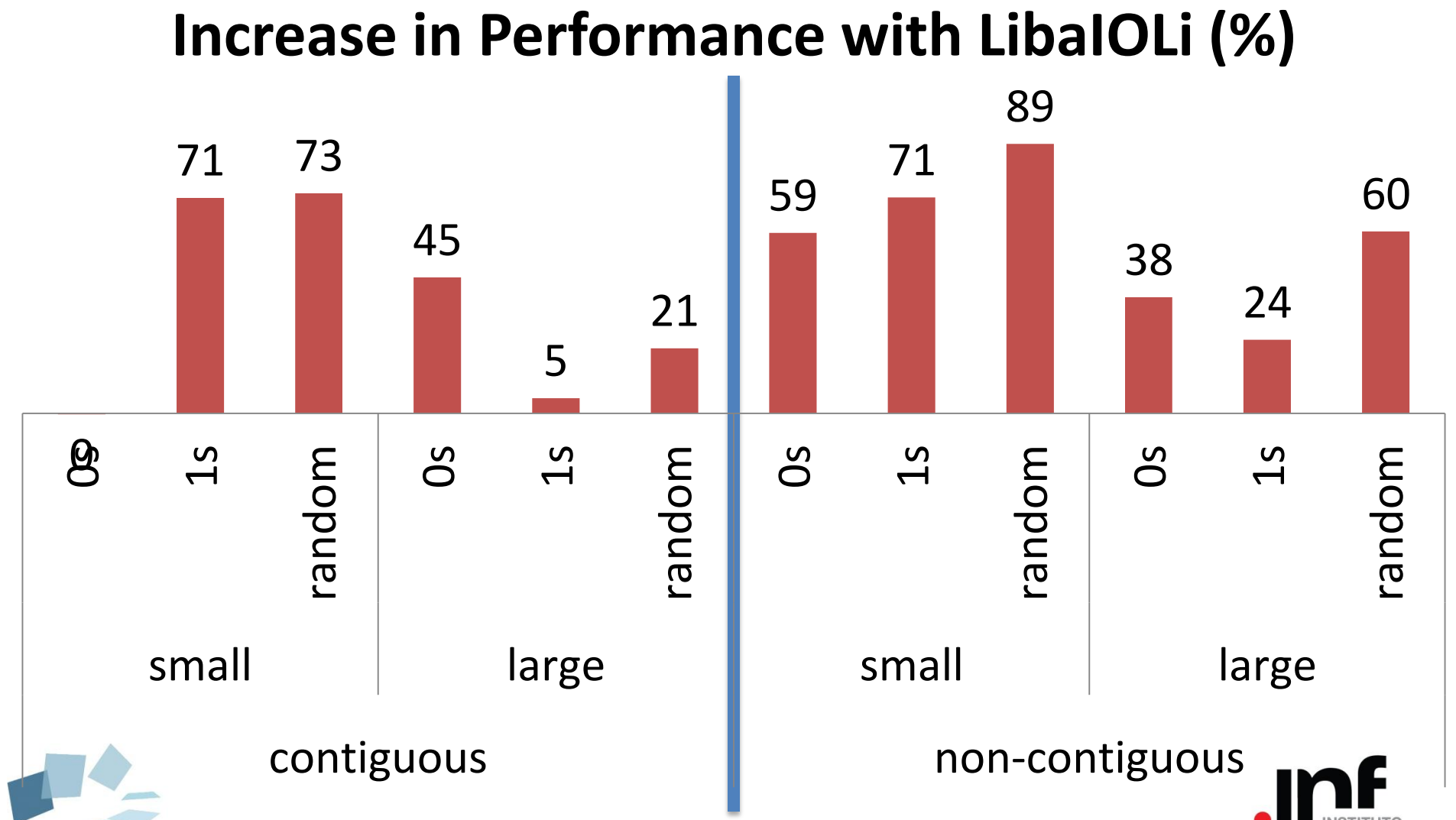
Preliminary Results – Write Operations



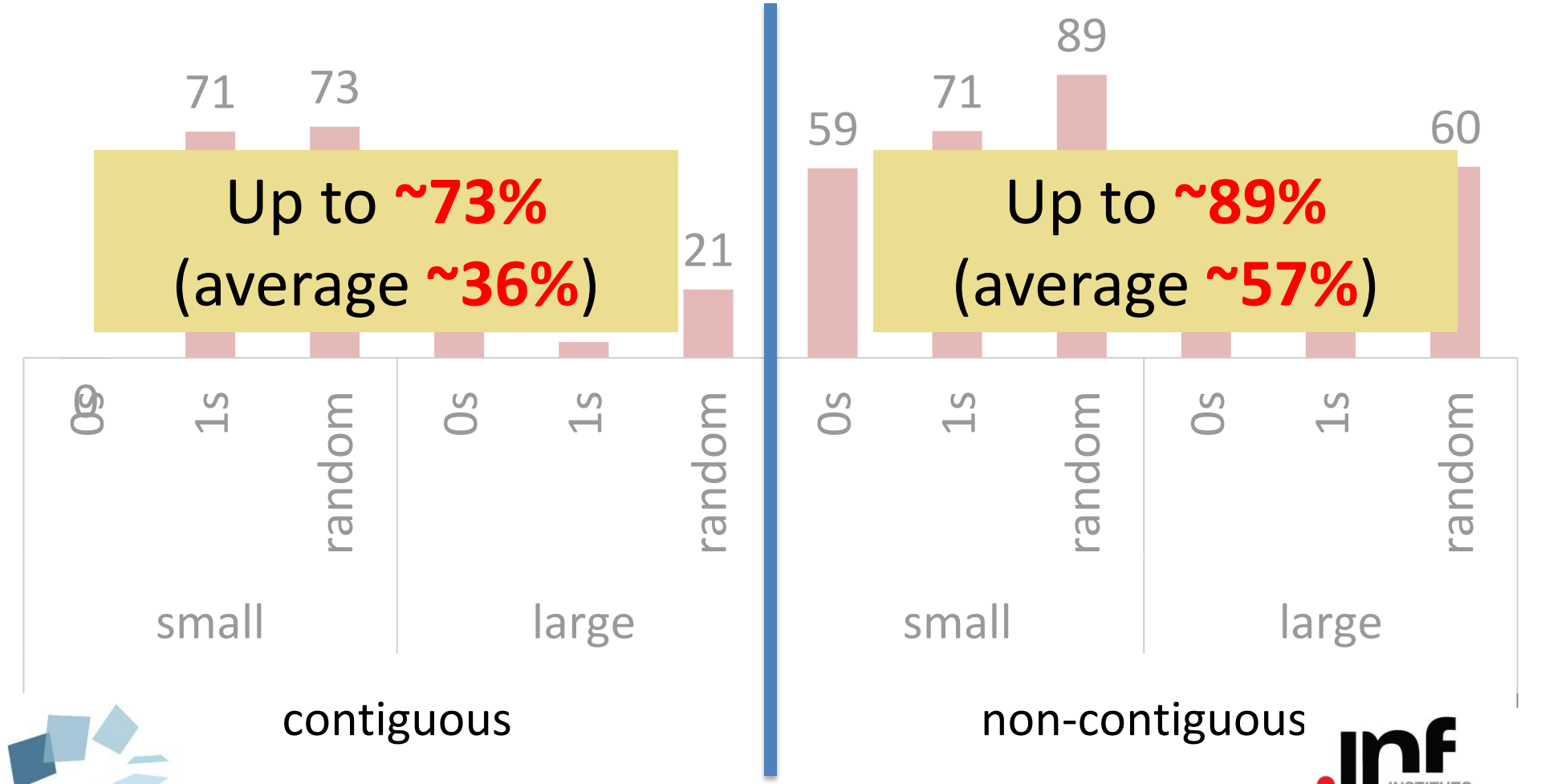
Preliminary Results – Write Operations



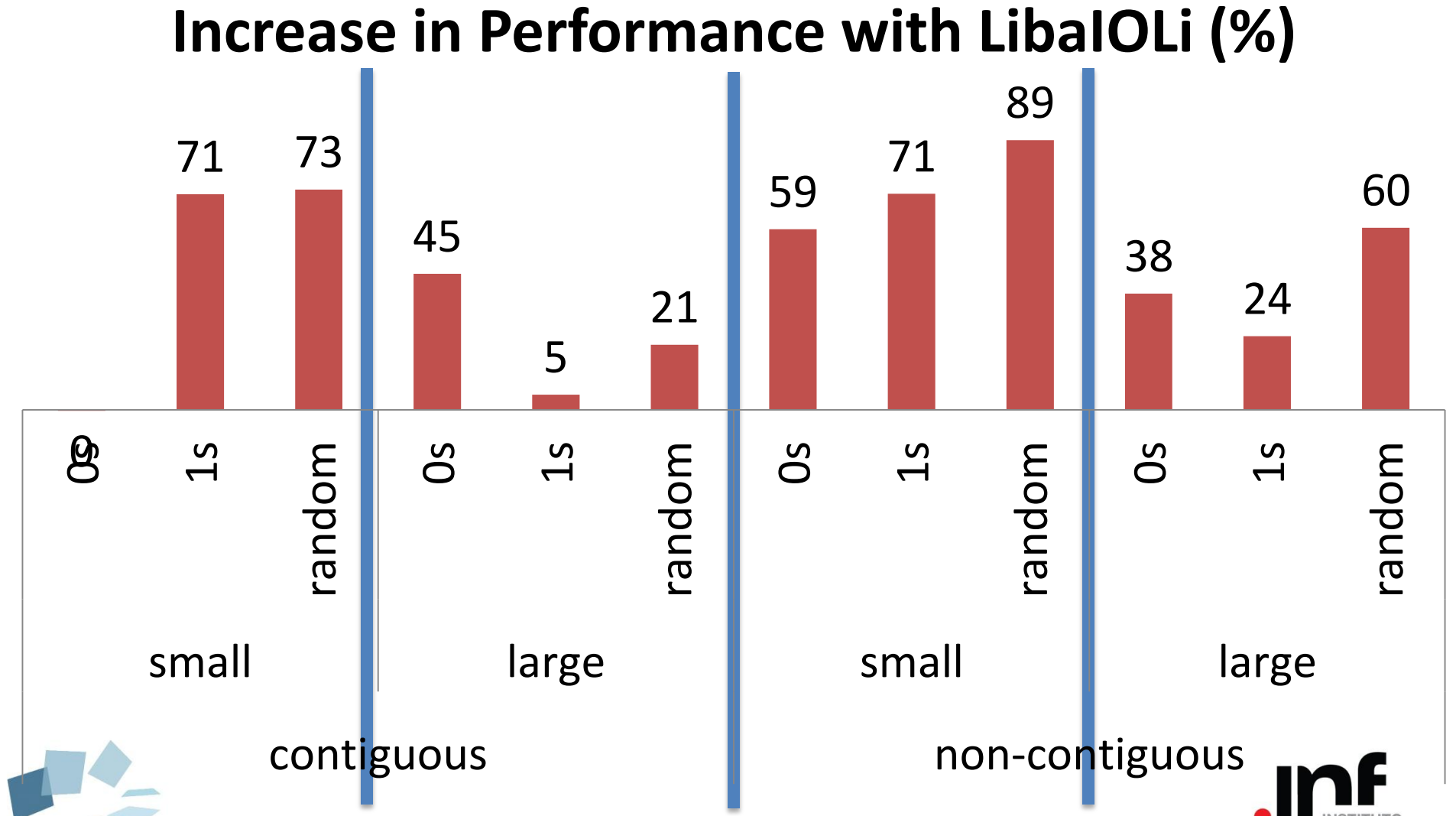
Preliminary Results – Read Operations



Preliminary Results – Read Operations



Preliminary Results – Read Operations



Preliminary Results – Read Operations