A framework for implementing IO-bound maintenance applications

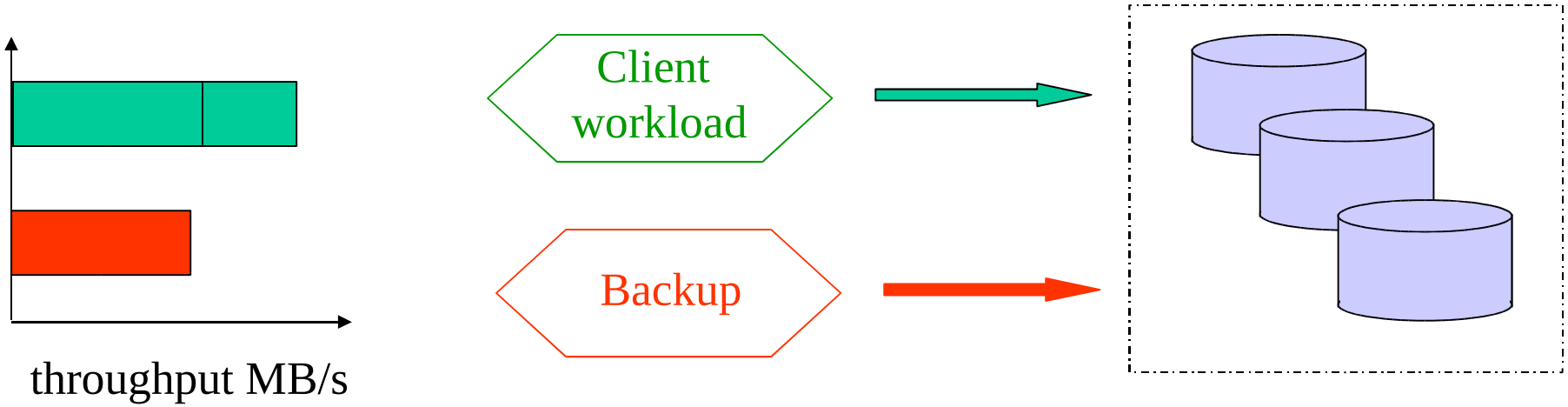
Eno Thereska

PARALLEL DATA LABORATORY
Carnegie Mellon University

Disk maintenance applications

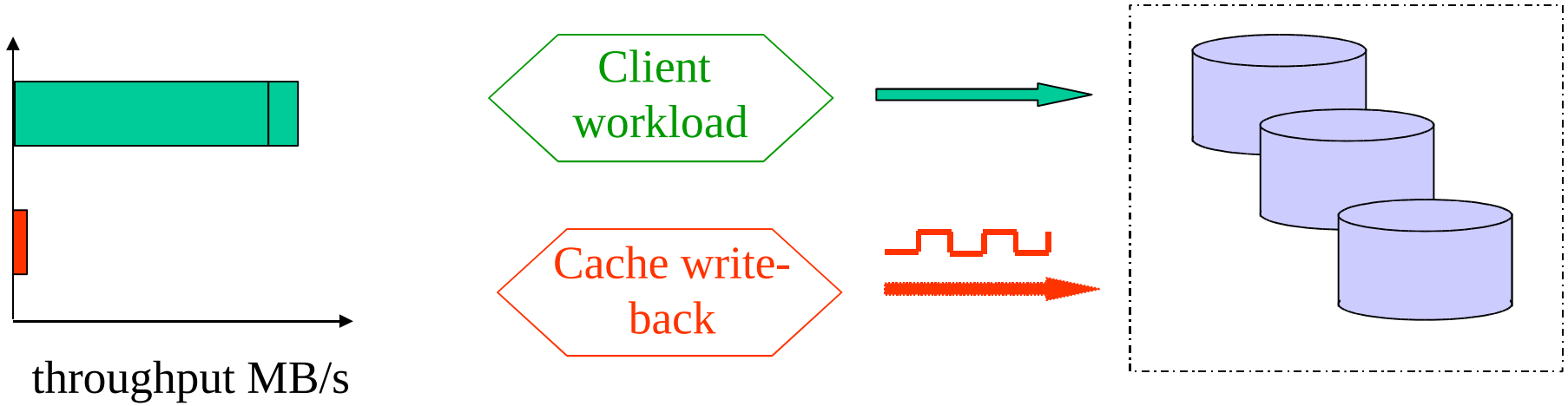
- Lots of disk maintenance apps
  - data protection (backup)
  - storage optimization (defrag, load bal.)
  - caching (write-backs)
- Important for system robustness
- Background activities
  - should *eventually* complete
  - ideally *without interfering* with primary apps

Current approaches



- Implement maintenance application as foreground application
  - competes for bandwidth or off-hours only

Current approaches



- Trickle maintenance activity periodically
  - lost opportunities due to inadequate scheduling decisions

Real support for background applications

- Push maintenance activities in the background
  - priorities and explicit support for them

- APIs allow application expressiveness
- Storage subsystem does the scheduling
  - using idle time, if there is any
  - using otherwise-wasted rotational latency in a busy system

Outline

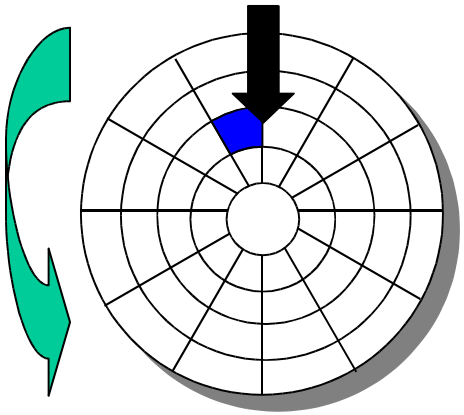
- Motivation and overview
- The freeblock subsystem
- Background application interfaces
- Example applications
- Conclusions

The freeblock subsystem

- Disk scheduling subsystem supporting new APIs with explicit background requests

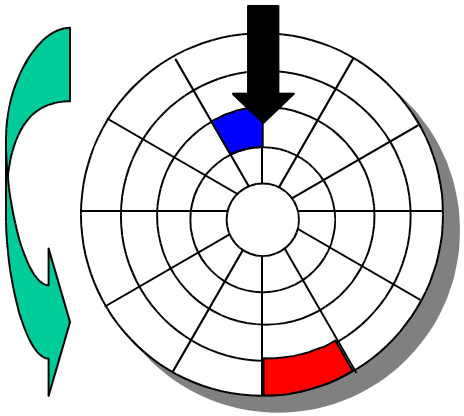
- Finds time for background activities
  - by detecting idle time (short and long bursts)
  - by utilizing otherwise-wasted rotational latency in a busy system

After reading blue sector



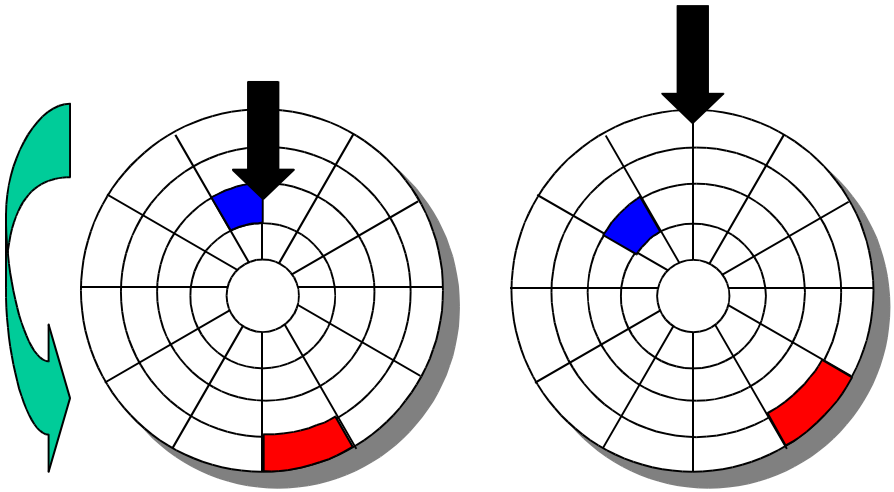
After BLUE read

Red request scheduled next



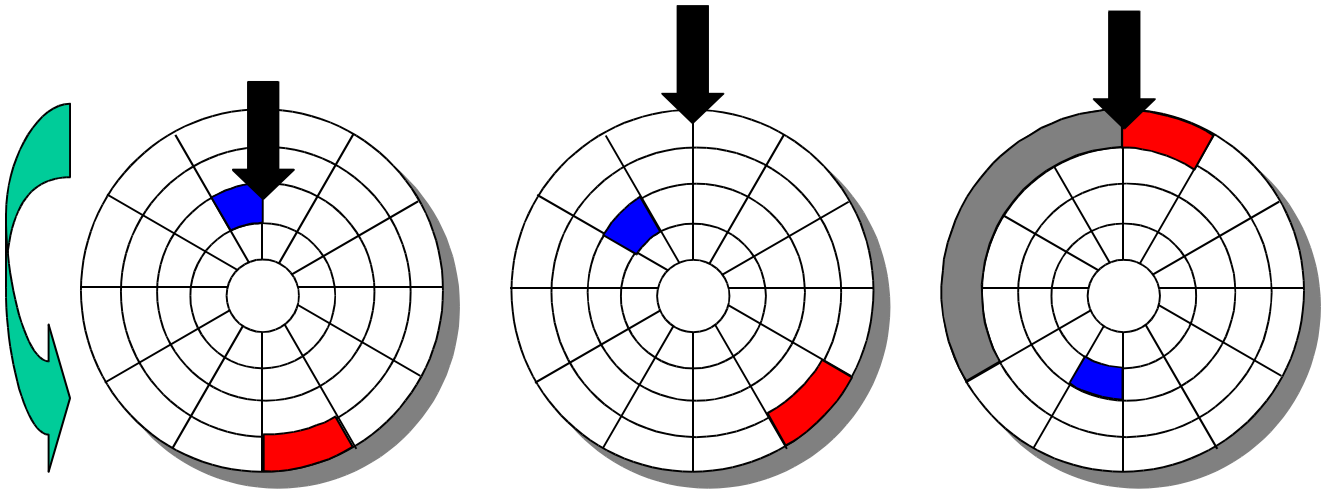
After BLUE read

Seek to Red's track



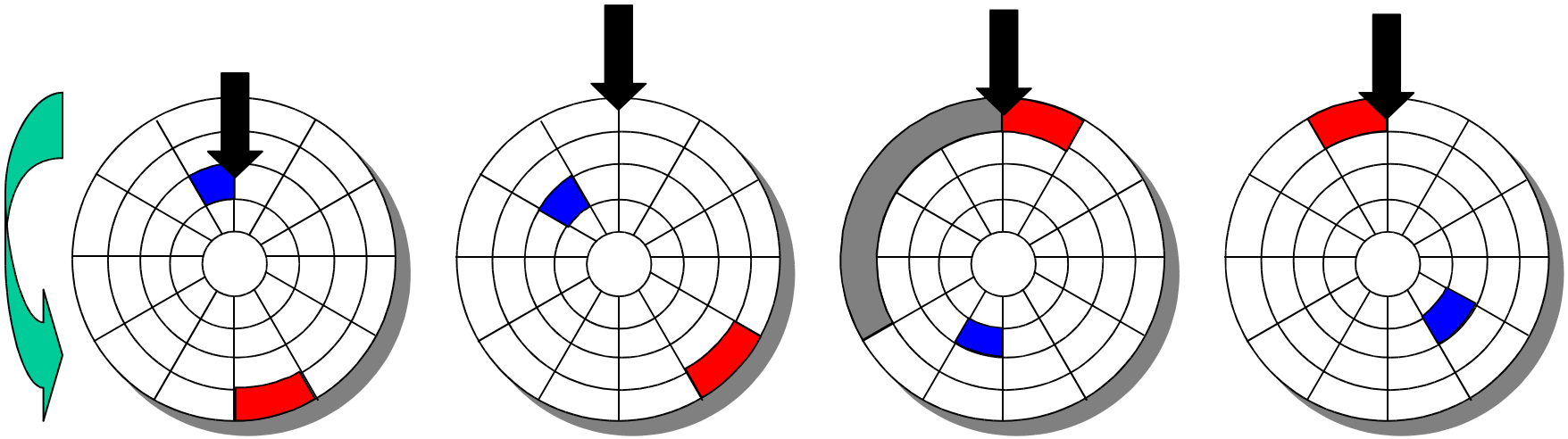
After BLUE read Seek for RED

Wait for Red sector to reach head



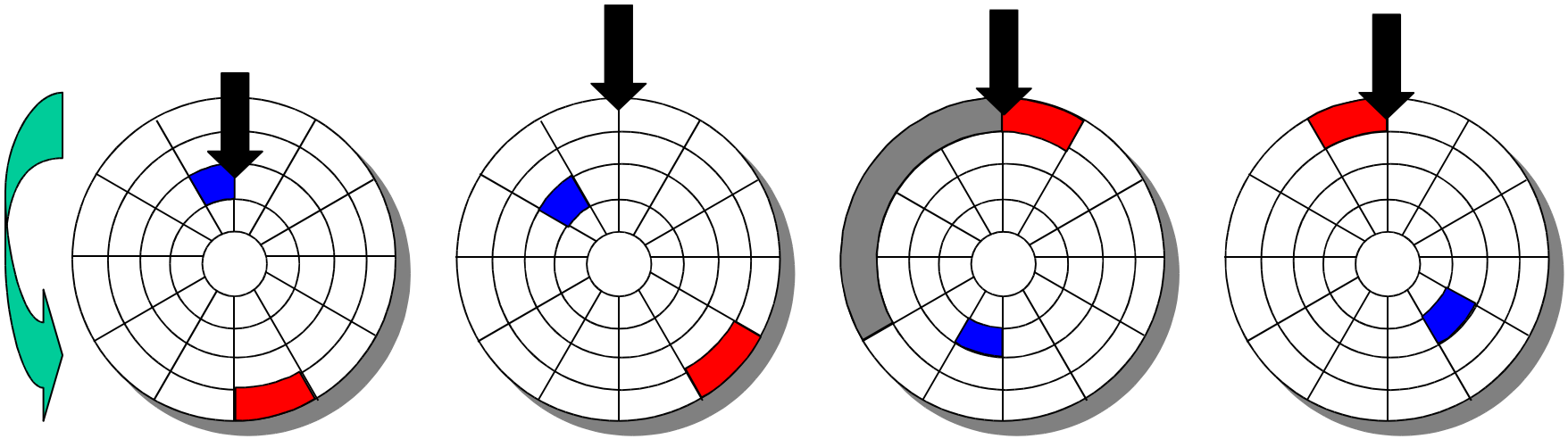
After BLUE read Seek for RED Rotational latency

Read Red sector



After BLUE read | Seek for RED | Rotational latency | After RED read

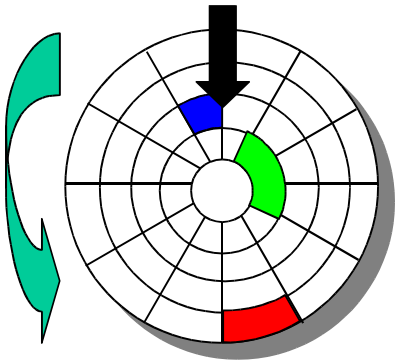
Traditional service time components



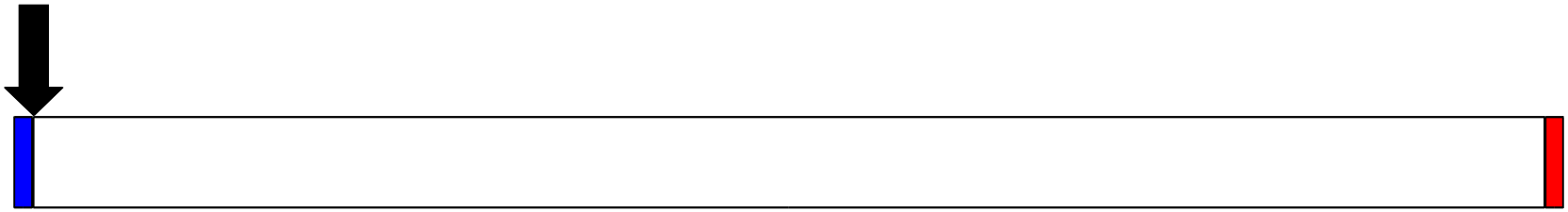
After BLUE read | Seek for RED | Rotational latency | After RED read

- Rotational latency is wasted time

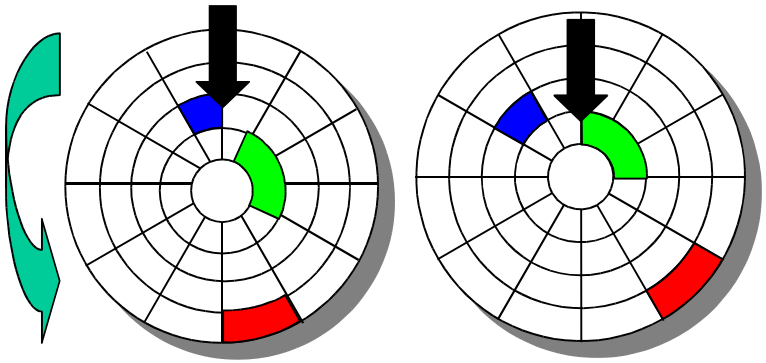
Rotational latency gap utilization



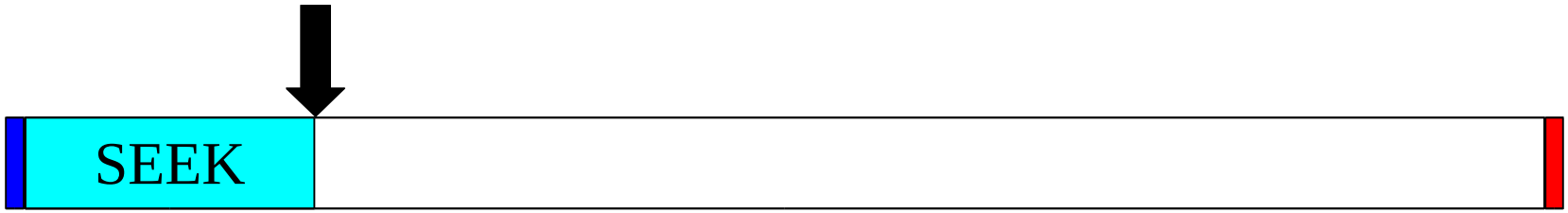
After BLUE read



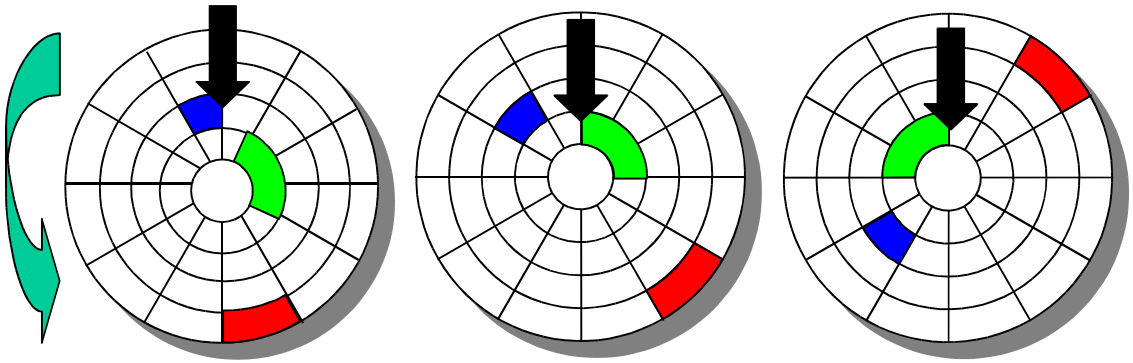
Seek to Third track



After BLUE read Seek to Third

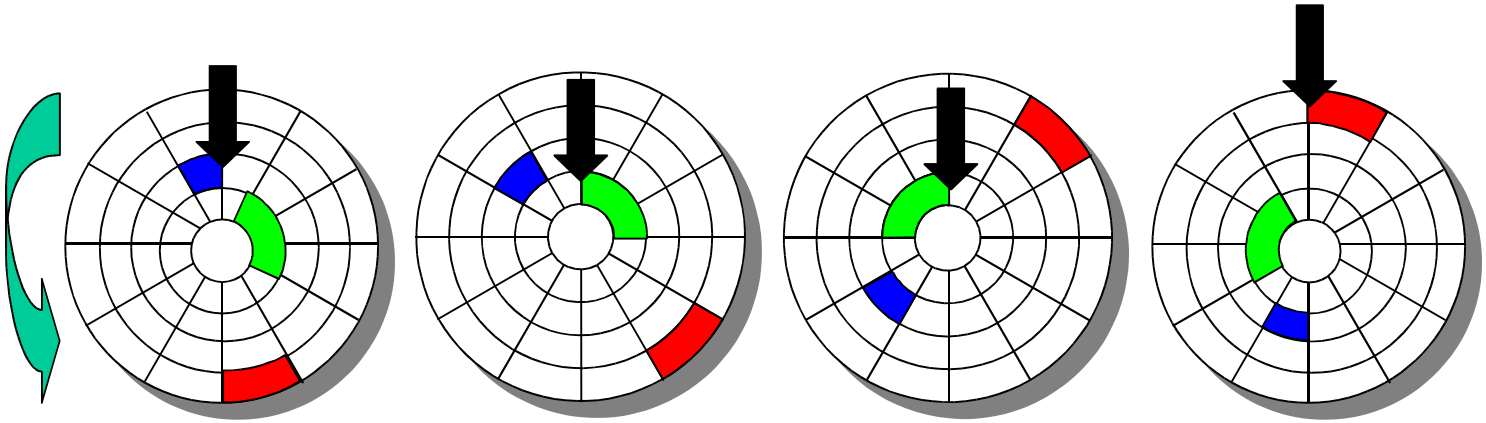


Free transfer



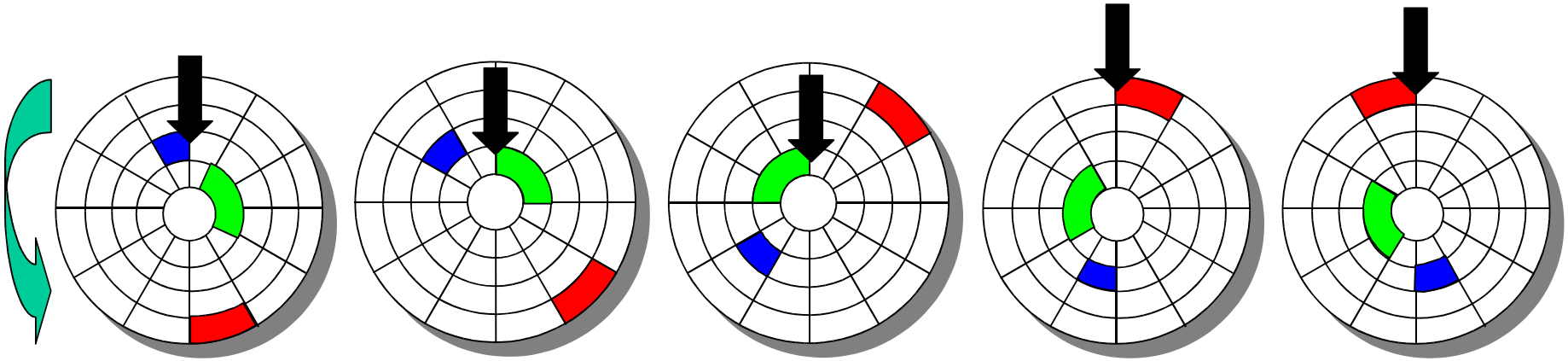
After BLUE read Seek to Third Free transfer

Seek to Red's track



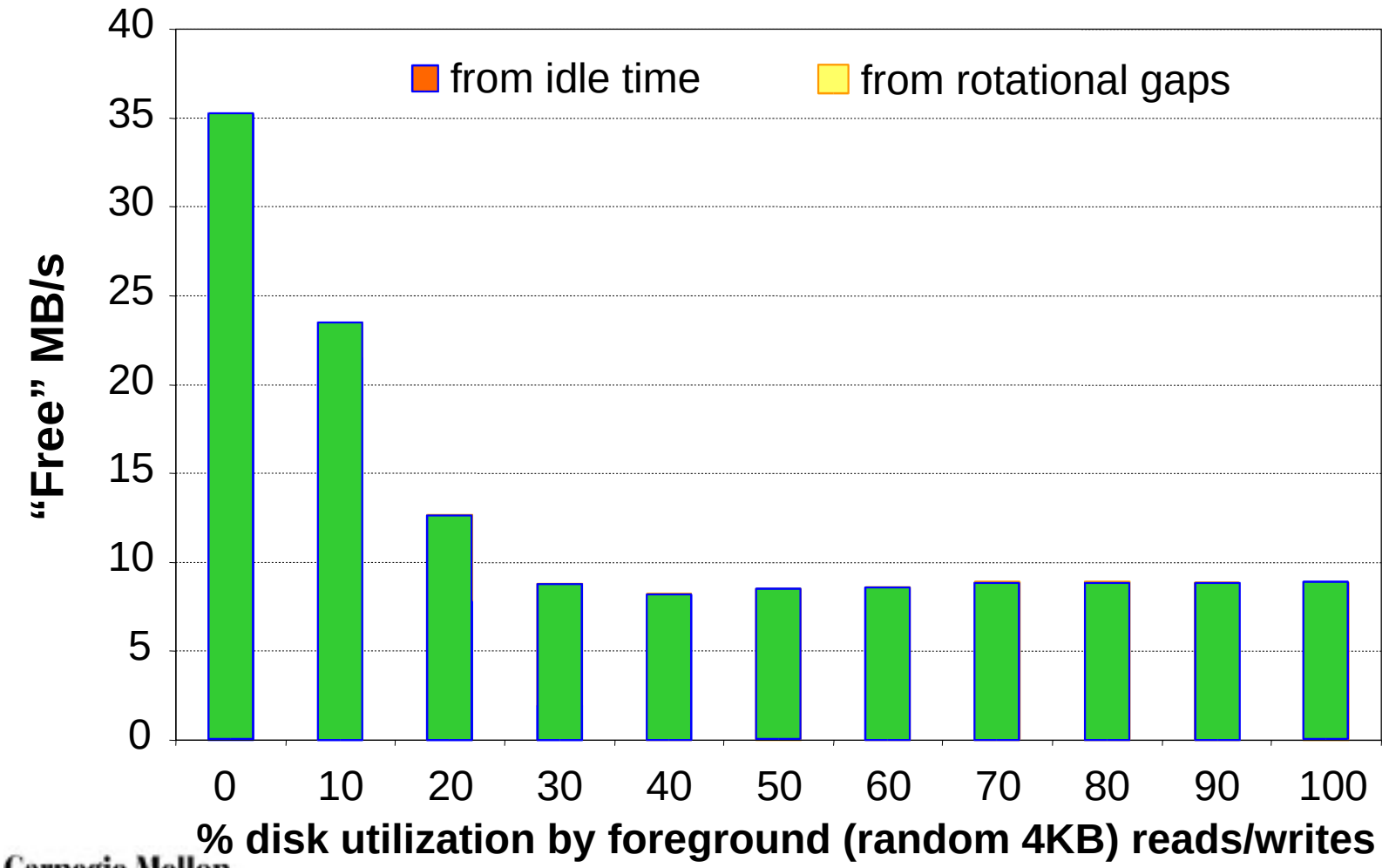
After BLUE read | Seek to Third | Free transfer | Seek to RED

Read Red sector



After BLUE read | Seek to Third | Free transfer | Seek to RED | After RED read

Steady background I/O progress



Carnegie Mellon
Parallel Data Laboratory

The freeblock subsystem *(cont…)*

- Implemented in FreeBSD
- Efficient scheduling
  - low CPU and memory utilizations
- Minimal impact on foreground workloads
  - < 2%

- *See refs for more details*

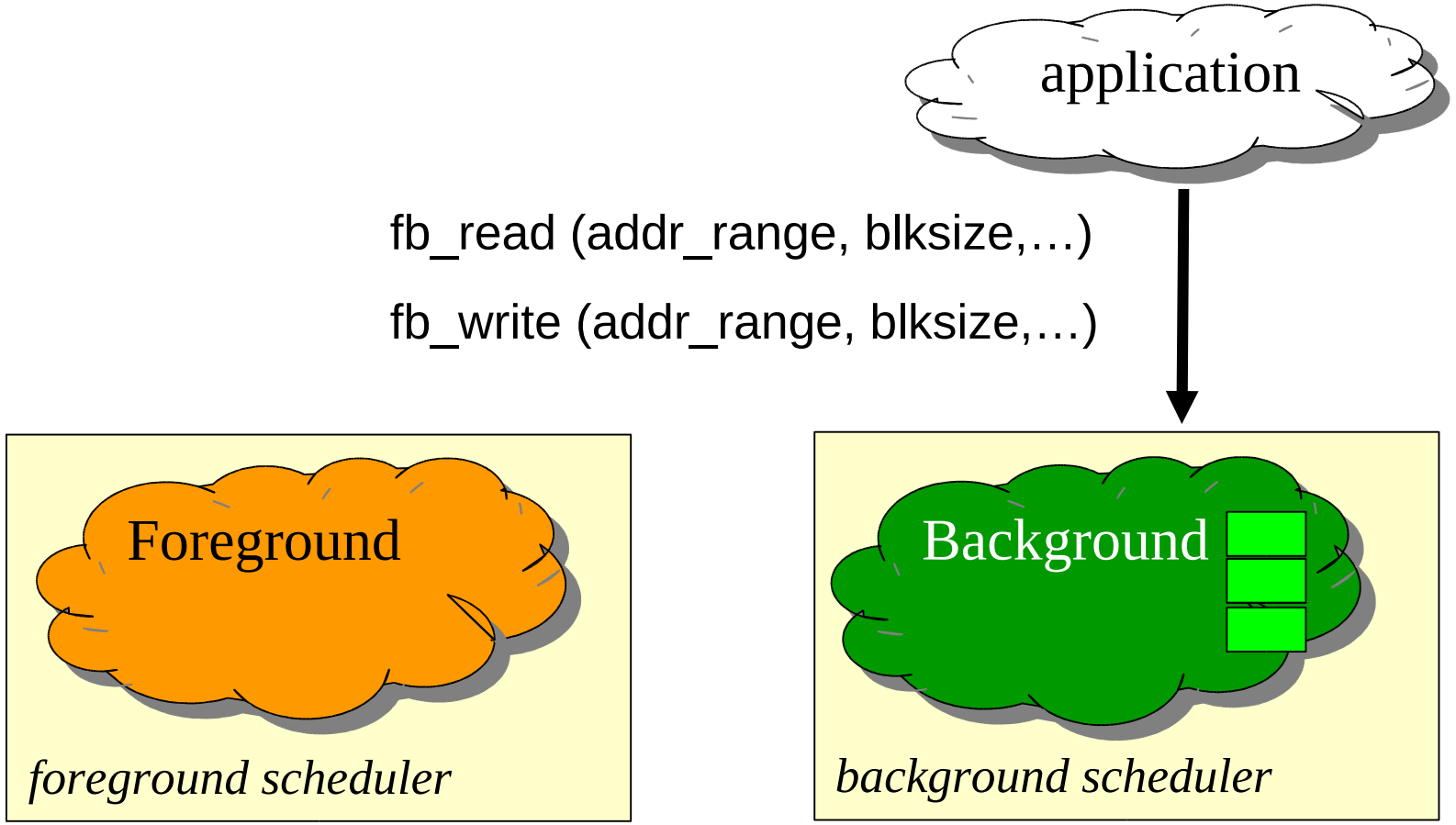
Outline

- Motivation and overview
- The freeblock subsystem
- Background application interfaces
- Example applications
- Conclusions

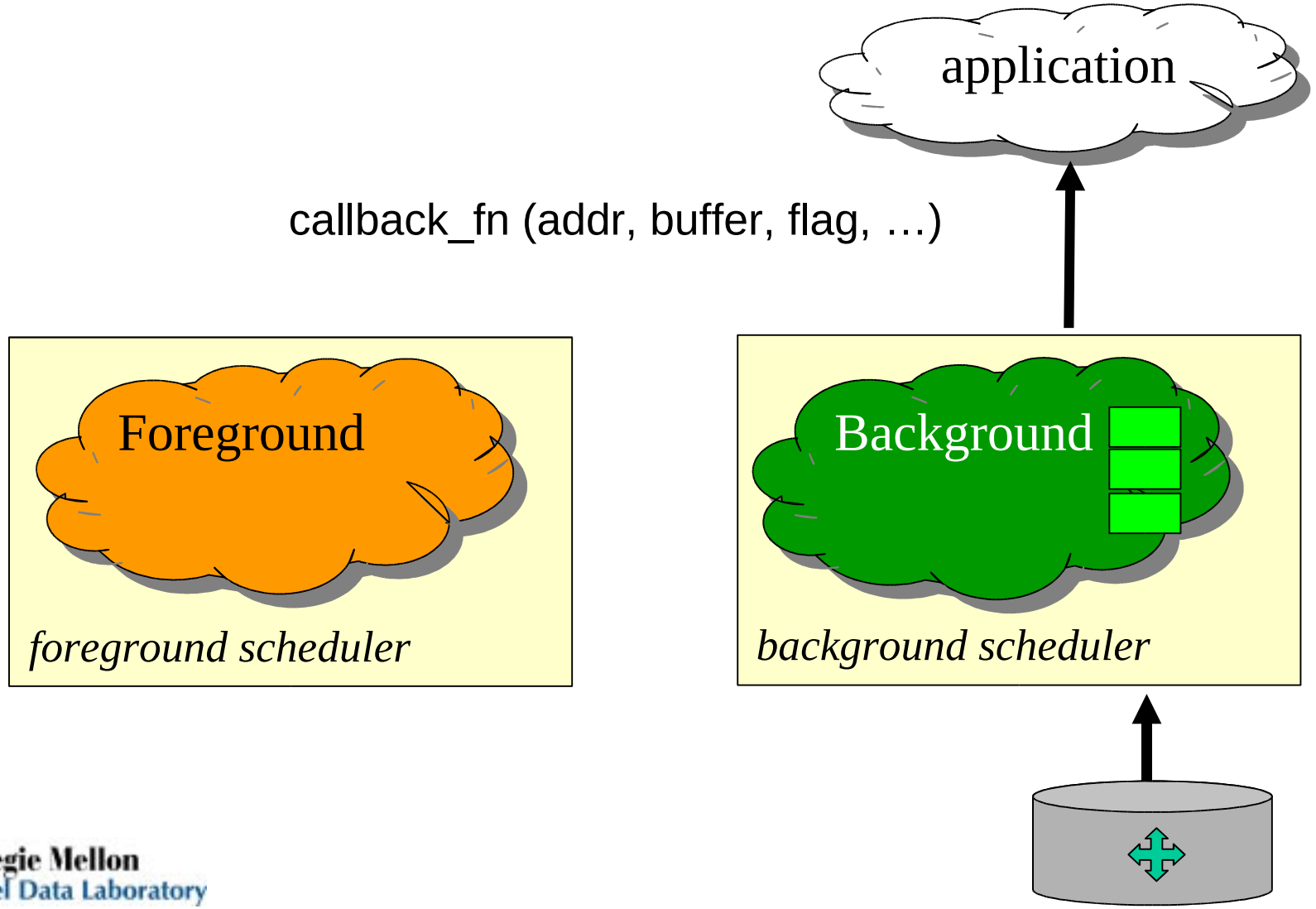
Application programming interface (API) goals

- Work exposed but done opportunistically
  - all disk accesses are asynchronous
- Minimized memory-induced constraints
  - late binding of memory buffers
  - late locking of memory buffers
- “Block size” can be application-specific
- Support for speculative tasks
- Support for rate control

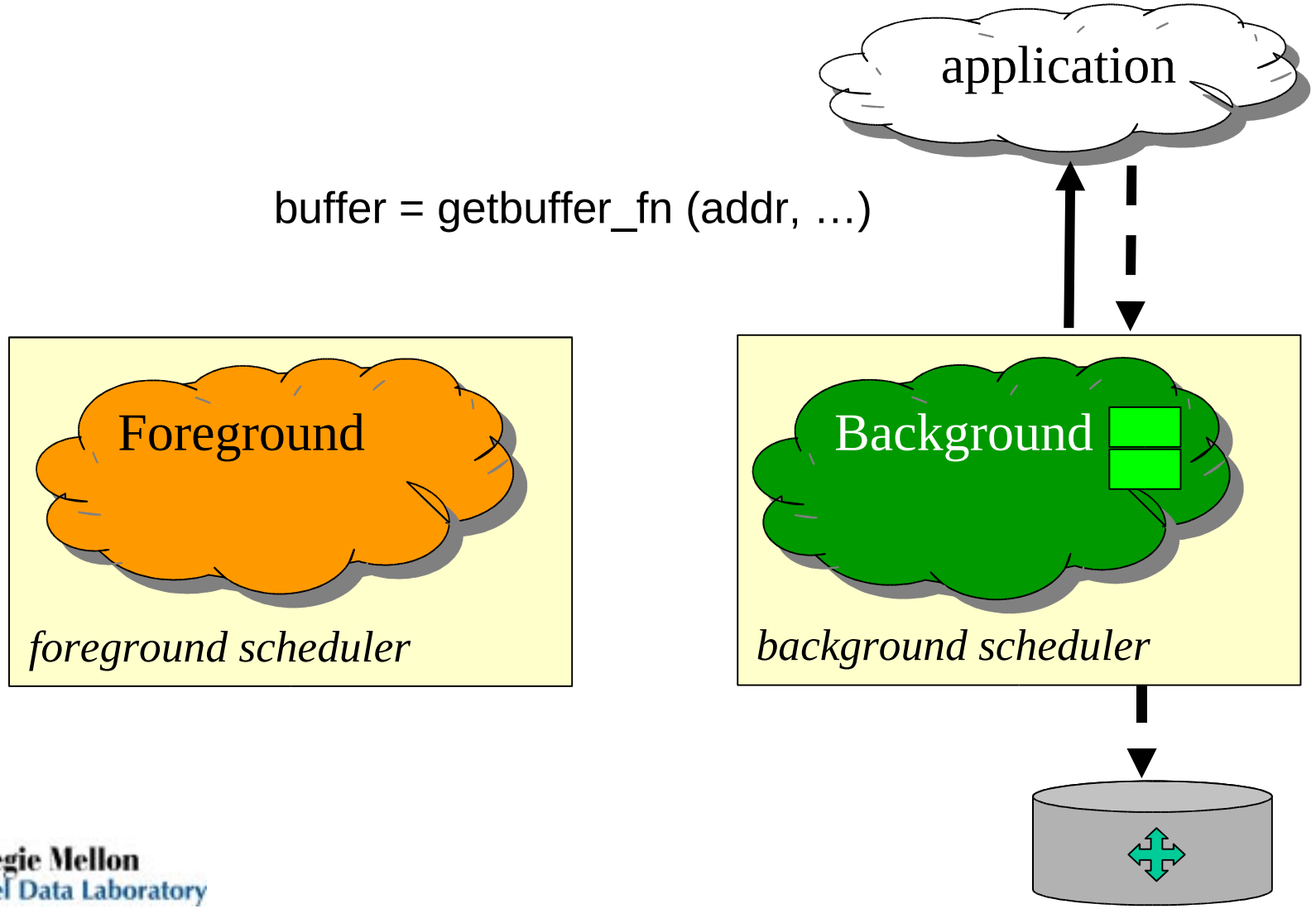
API description: task registration



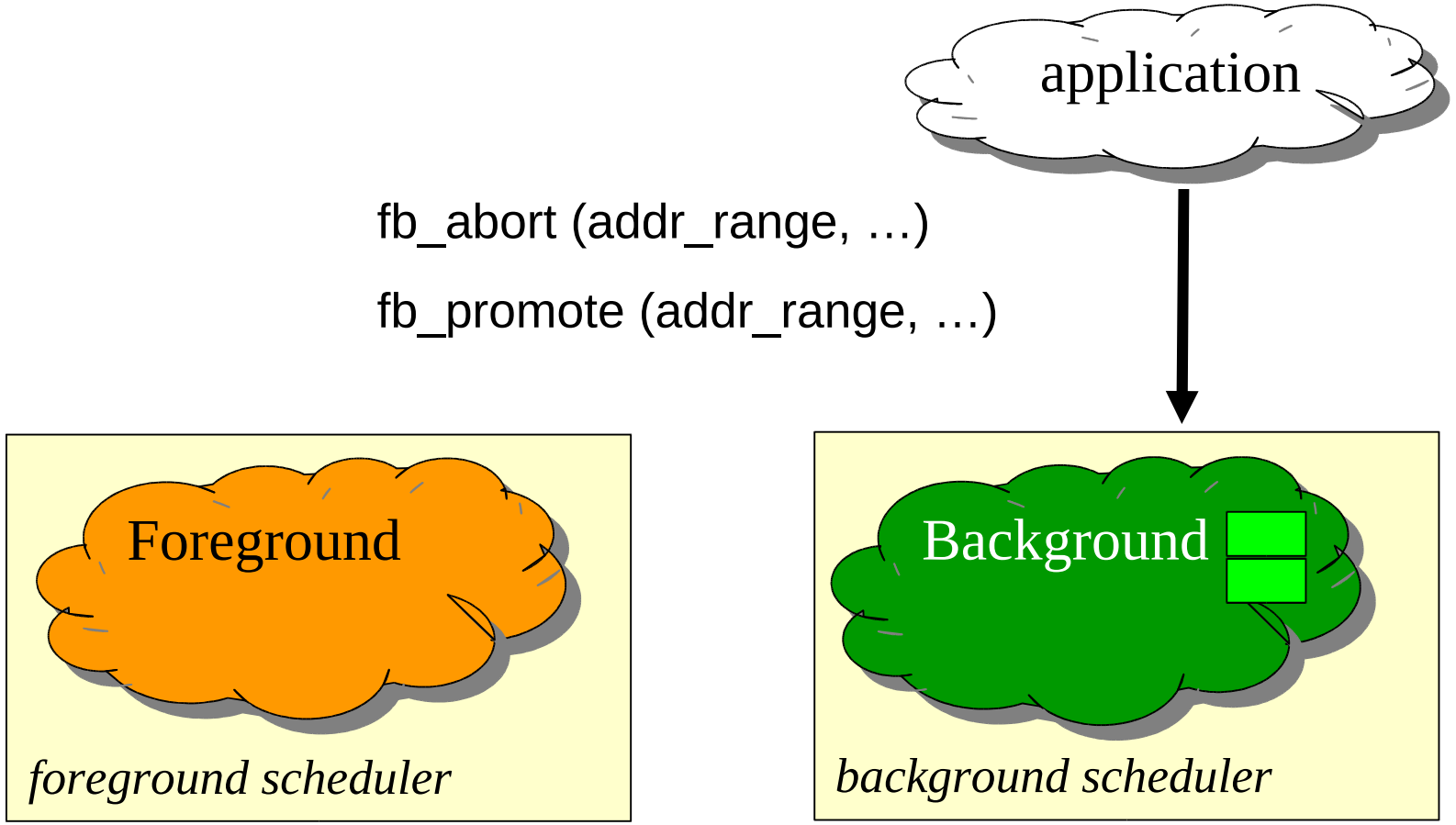
API description: task completion



API description: late locking of buffers



API description: aborting/promoting tasks



Complete API

Function Name	Arguments	Description
fb_open	*priority, callback_fn, getbuffer_fn*	Open a freeblock session (ret: *session_id*)
fb_close	*session_id*	Close a freeblock session
fb_read	*session_id, addr_range, blksize, callback_param*	Register a freeblock read task
fb_write	*session_id, addr_range, blksize, callback_param*	Register a freeblock write task (ret: *task_id*)
fb_abort	*session_id, addr_range*	Abort parts of registered task
fb_promote	*session_id, addr_range*	Promote parts of registered task
fb_suspend	*session_id*	Suspend scheduling of a session's tasks
fb_resume	*session_id*	Resume scheduling of a session's tasks
**(callback_fn)*	*session_id, addr, buffer, flags, callback_param*	Report that part of task completed
**(getbuffer_fn)*	*session_id, addr, callback_param*	Get memory address for selected write

Designing disk maintenance applications

- APIs talk in terms of logical blocks (LBNs)
- Some applications need structured version
  - as presented by file system or database

- Example consistency issues
  - application wants to read file “foo”
  - registers task for inode’s blocks
  - by time blocks read, file may not exist anymore!

Designing disk maintenance applications

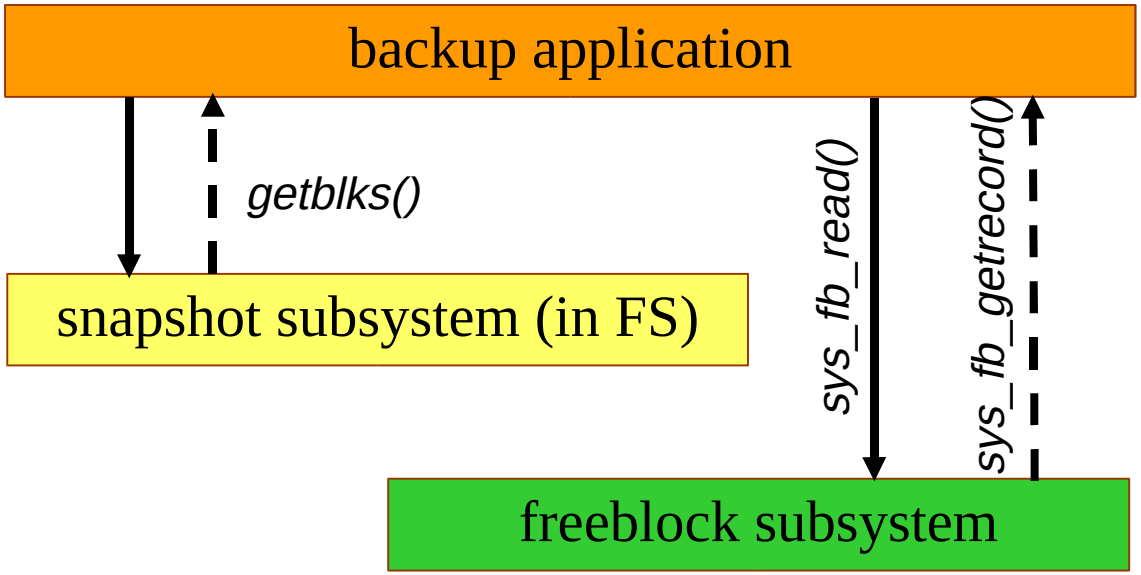
- Application does not care about structure
  - scrubbing, data migration, array reconstruction
- Coordinate with file system/database
  - cache write-backs, LFS cleaner, index generation
- Utilize snapshots
  - backup, background *fsck*

Outline

- Motivation and overview
- The freeblock subsystem
- Background application interfaces
- Example applications
- Conclusions

Example #1: Physical backup

- Backup done using snapshots

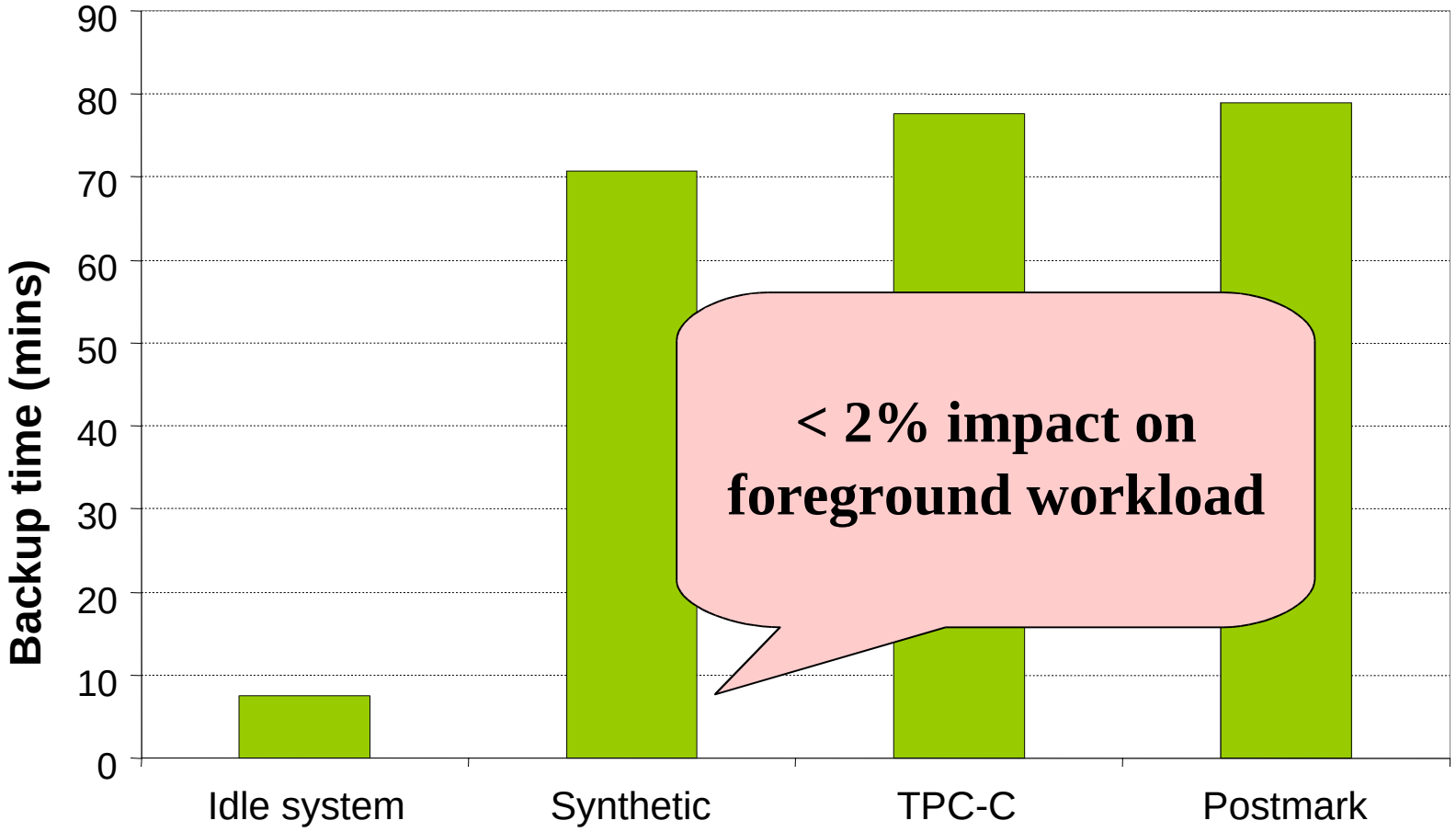


Example #1: Physical backup

- Experimental setup
  - 18 GB Seagate Cheetah 36ES
  - FreeBSD in-kernel implementation
  - PIII with 384MB of RAM
  - 3 benchmarks used: *Synthetic, TPC-C, Postmark*
  - snapshot includes 12GB of disk

- *GOAL*: read whole snapshot for free

Backup completed for free



Example #2: Cache write-backs

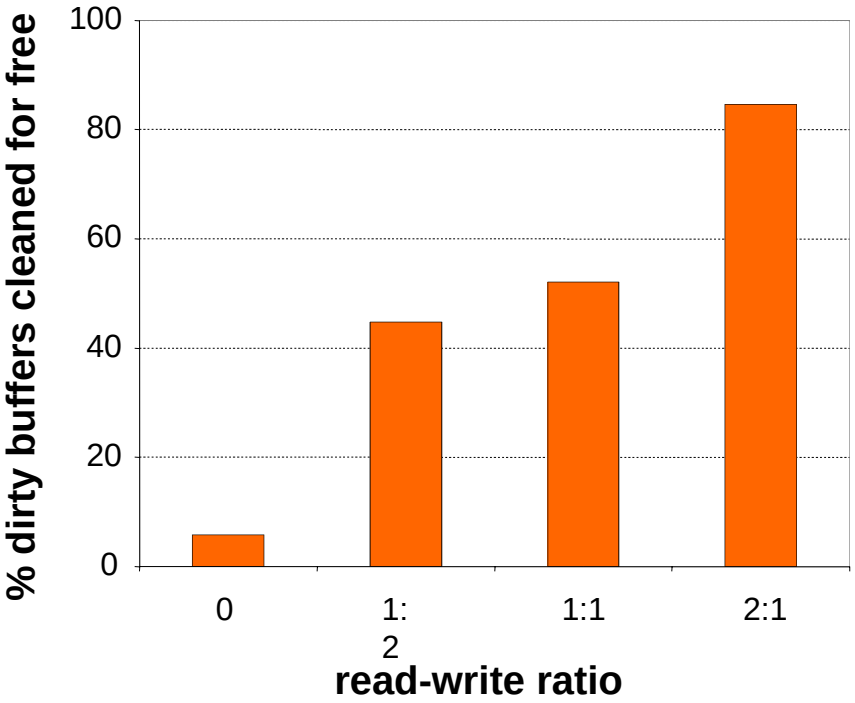
- Must flush dirty buffers
  - for space reclamation
  - for persistence (if memory is not NVRAM)

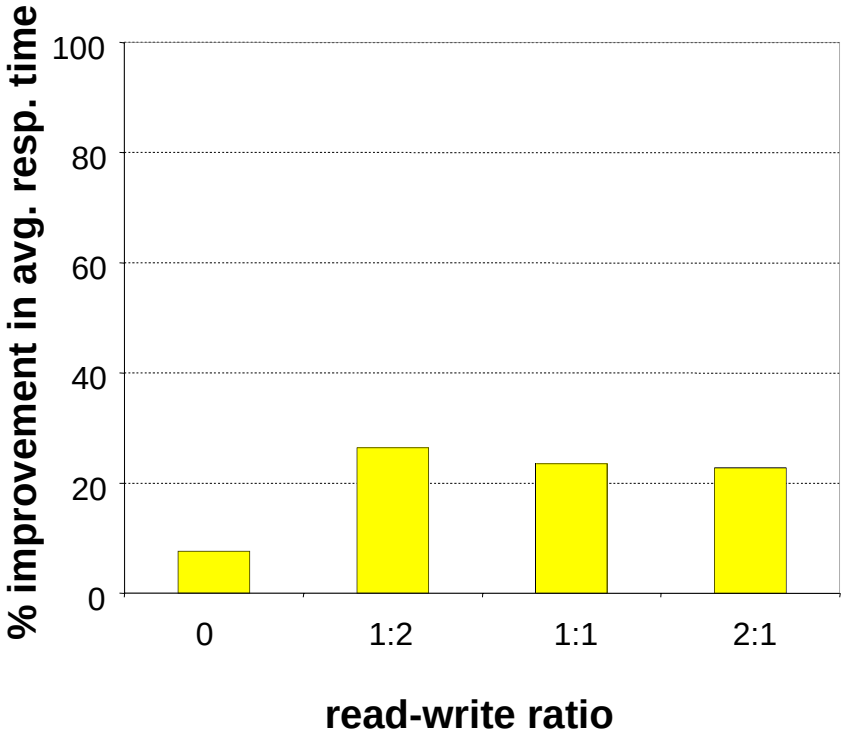
- Simple cache manager extensions
  - *fb_write(dirty_buffer,…)* → when blocks become dirty
  - *getbuffer_fn(dirty_buffer,…)* → calling back to lock
  - *fb_promote(dirty_buffer,…)* → when flush is forced
  - *fb_abort(dirty_buffer,…)* → when block dies in cache

Example #2: Cache write-backs

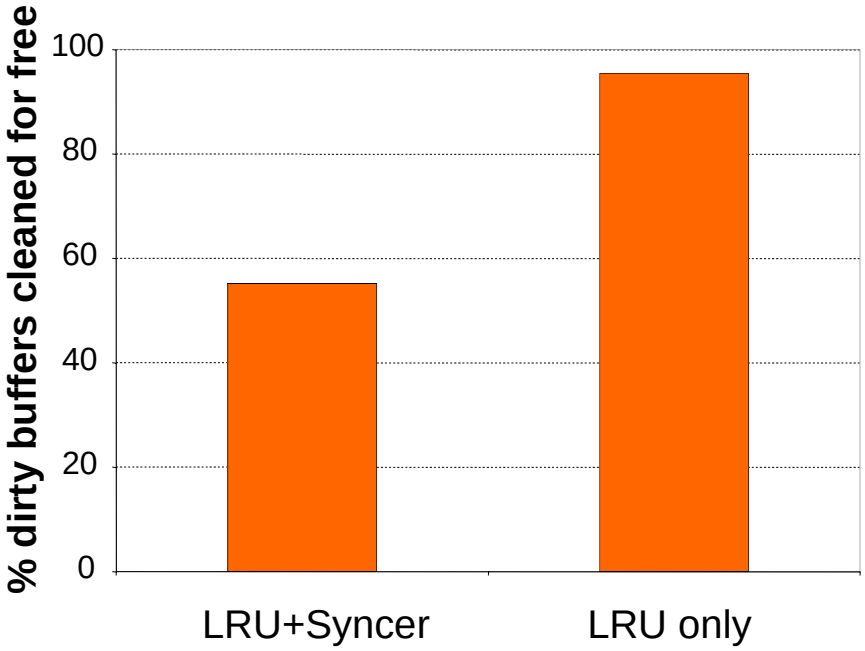
- Experimental setup
  - 18 GB Seagate Cheetah 36ES
  - PIII with 384MB of RAM
  - controlled experiments with synthetic workload
  - benchmarks (same as used before) in FreeBSD
  - syncer daemon wakes up every 1 sec and flushes entries that have been dirty > 30secs
- *GOAL*: write back dirty buffers for free

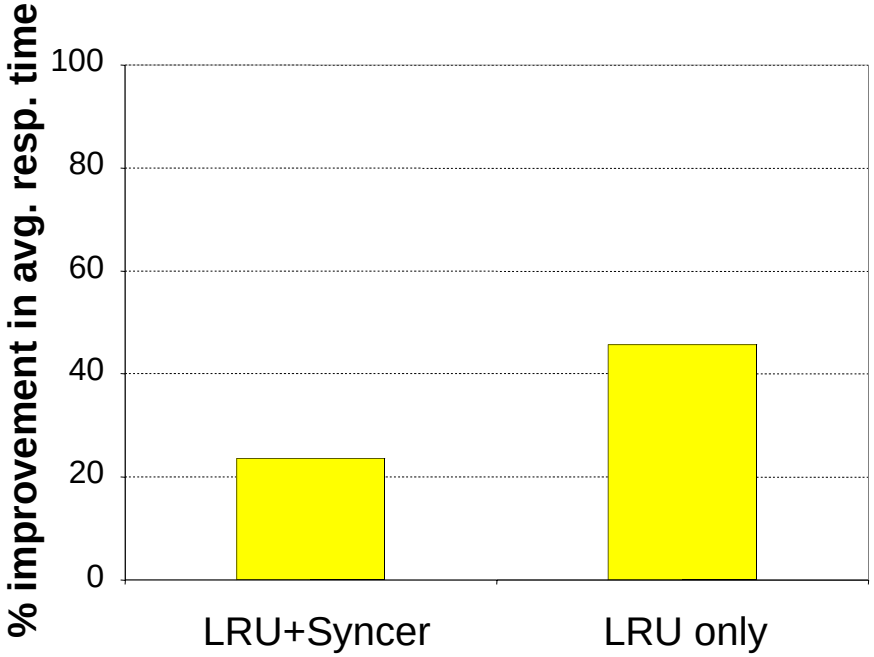
Foreground read:write has impact



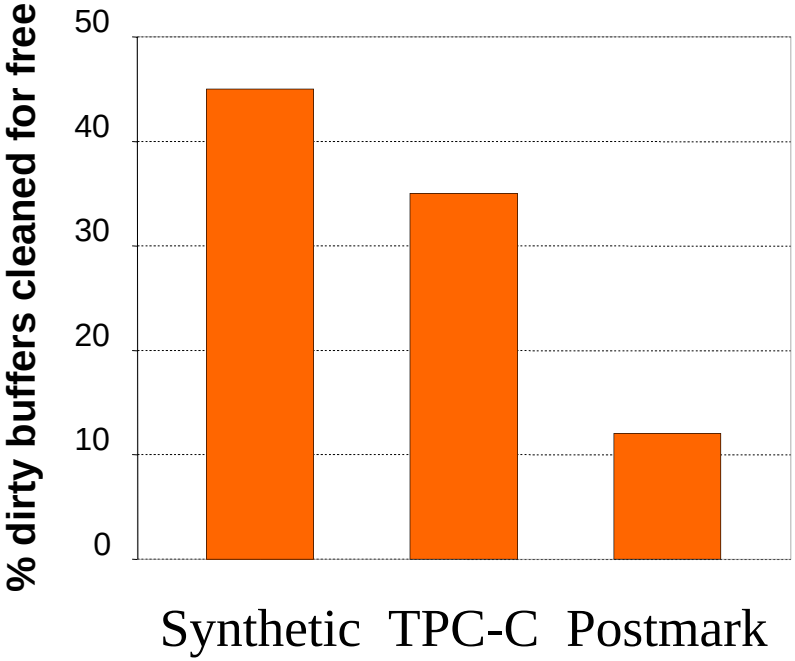


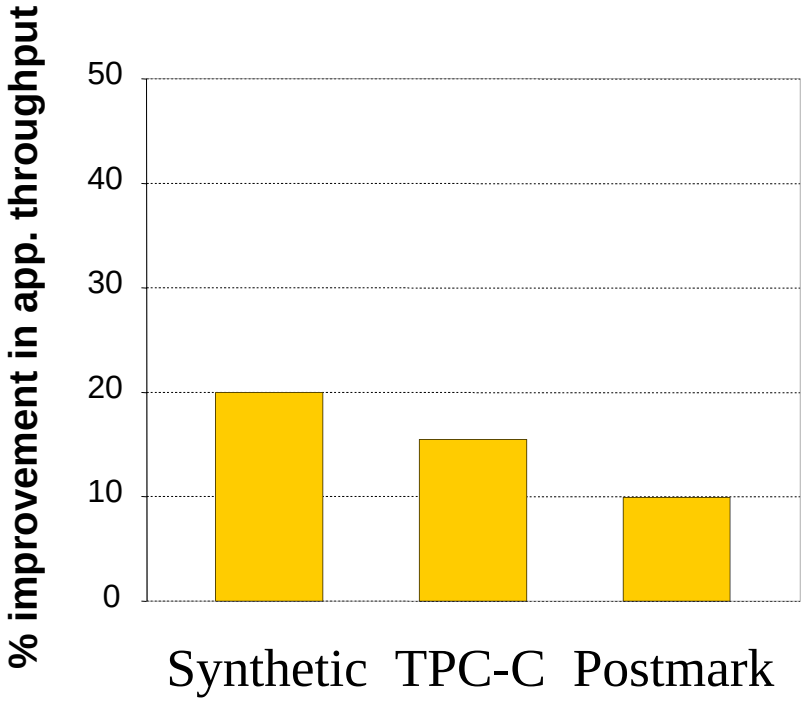
95% of NVRAM cleaned for free





10-20% improvement in overall perf.



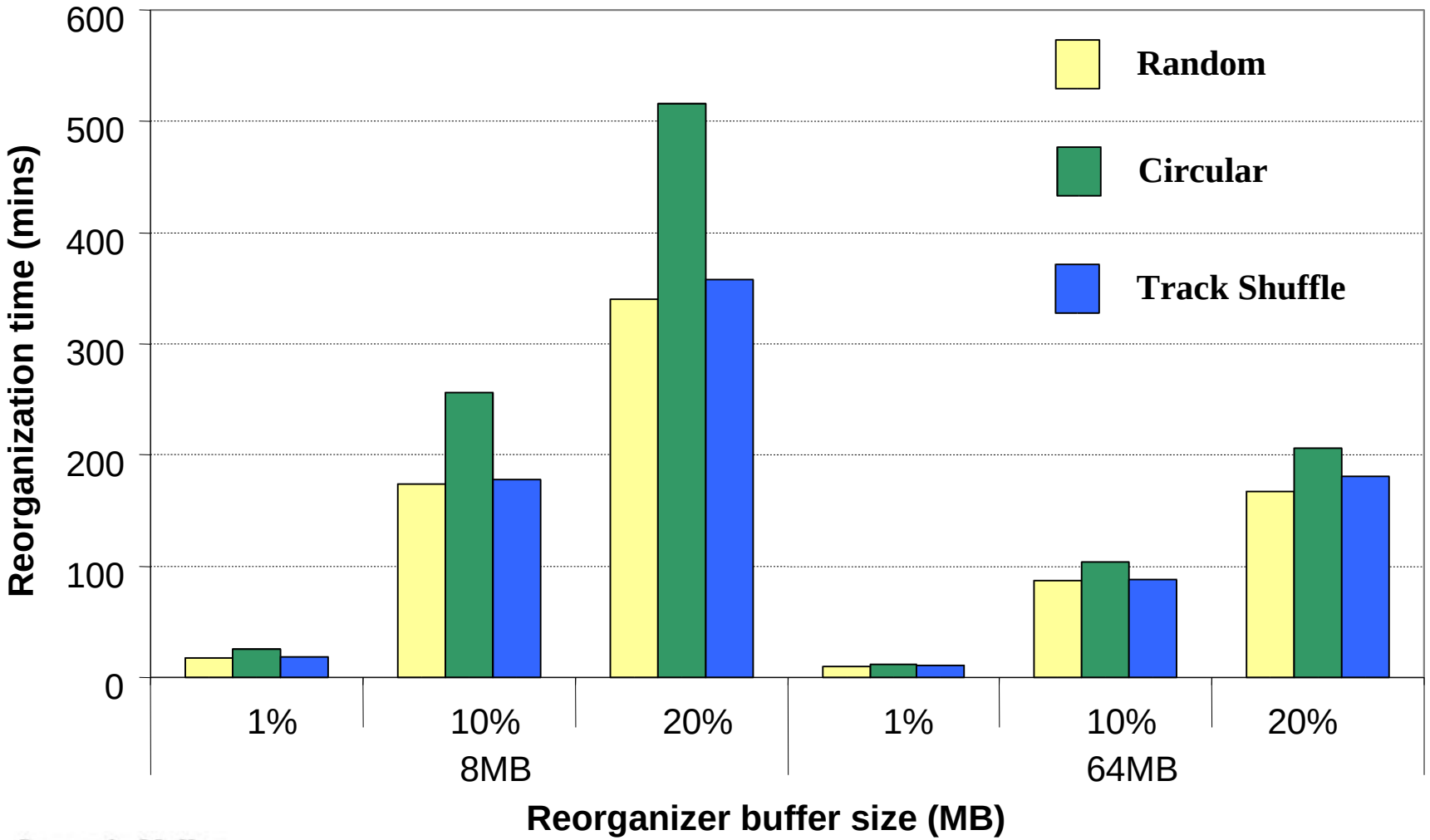


Example #3: Layout reorganizer

- Layout reorganization improves access latencies
  - defragmentation is a type of reorganization
  - typical example of background activity

- Our experiment:
  - disk used is 18GB
  - we want to defrag up to 20% of it
  - goal: defrag for free

Disk Layout Reorganized for Free!



Carnegie Mellon
Parallel Data Laboratory

Other maintenance applications

- Virus scanner
- LFS cleaner
- Disk scrubber
- Data mining
- Data migration

Summary

- Framework for building background storage applications
- Asynchronous interfaces
  - applications describe what they need
  - storage subsystem satisfies their needs
- Works well for real applications

http://www.pdl.cmu.edu/Freeblock/