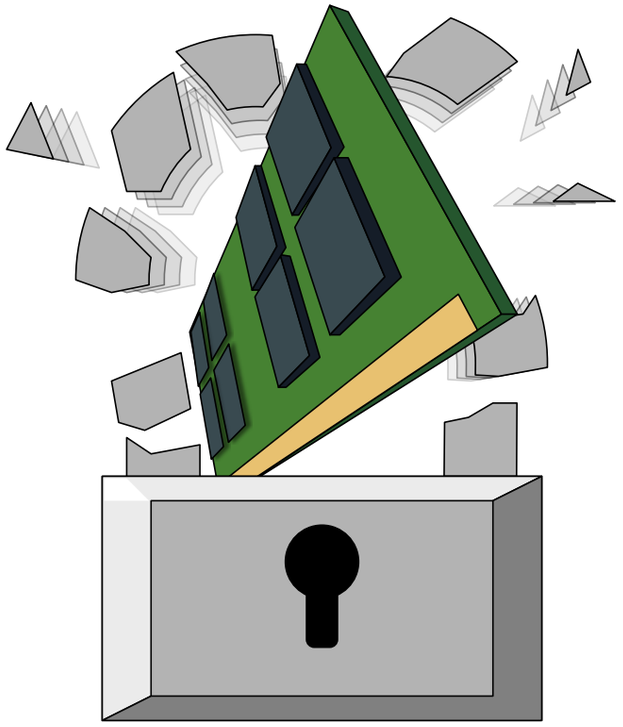
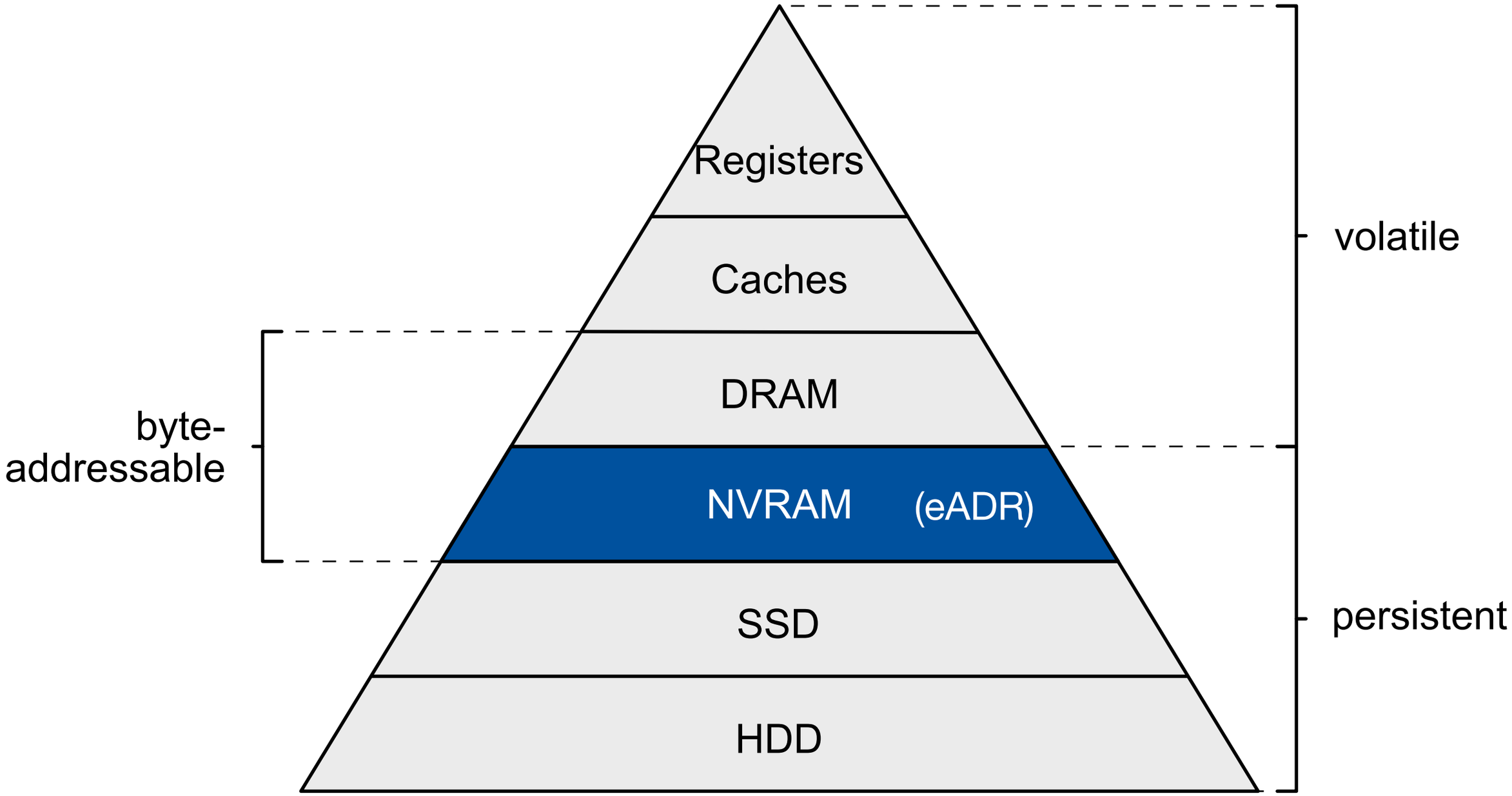
MA Thesis

Lo(ck|g)-Free Page Allocator for NVM



Lars Wrenger, lars@wrenger.net

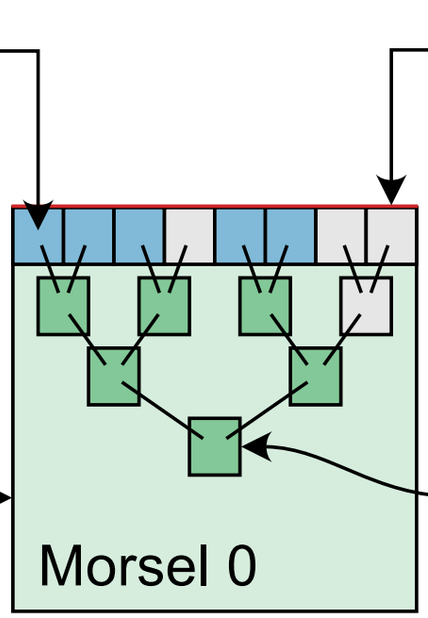
Non-Volatile Memory



Context - ParPerOs & Morsels

User-Data
- Exposed to user
- Page granularity

Morsel
- Self-contained
- Optionally persistent
- Shareable between processes



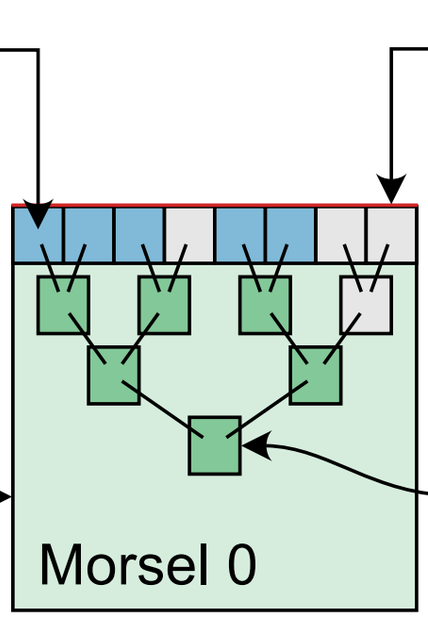
Surface
- 4K·512N size and alignment
- Thinly and lazily populated

Meta-Data
- User-Data ↔ Surface mapping
- MMU-specific page tables
- Allocated lazily

Context - ParPerOs & Morsels

User-Data
- Exposed to user
- Page granularity

Morsel
- Self-contained
- Optionally persistent
- Shareable between processes



Surface
- 4K·512N size and alignment
- Thinly and lazily populated

Meta-Data
- User-Data ↔ Surface mapping
- MMU-specific page tables
- Allocated lazily

Lo(ck|g)-free Allocator for NVM

- Alloc / free pages for morsels
  - Hardware specific pages: 4KiB, 2MiB, 1GiB

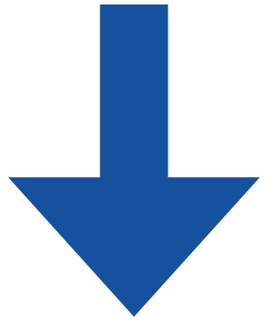
- Goals
  - **Concurrency**
    - Parallelism & scaling
  - **Persistency**
    - Recovery from power loss
    - Only limited number of lost pages

Concurrency

- Locks (Semaphore, Mutex) → Crash tolerance!
- Logs (ACID) → Many write cycles!

Concurrency

- Locks (Semaphore, Mutex) → Crash tolerance!
- Logs (ACID) → Many write cycles!



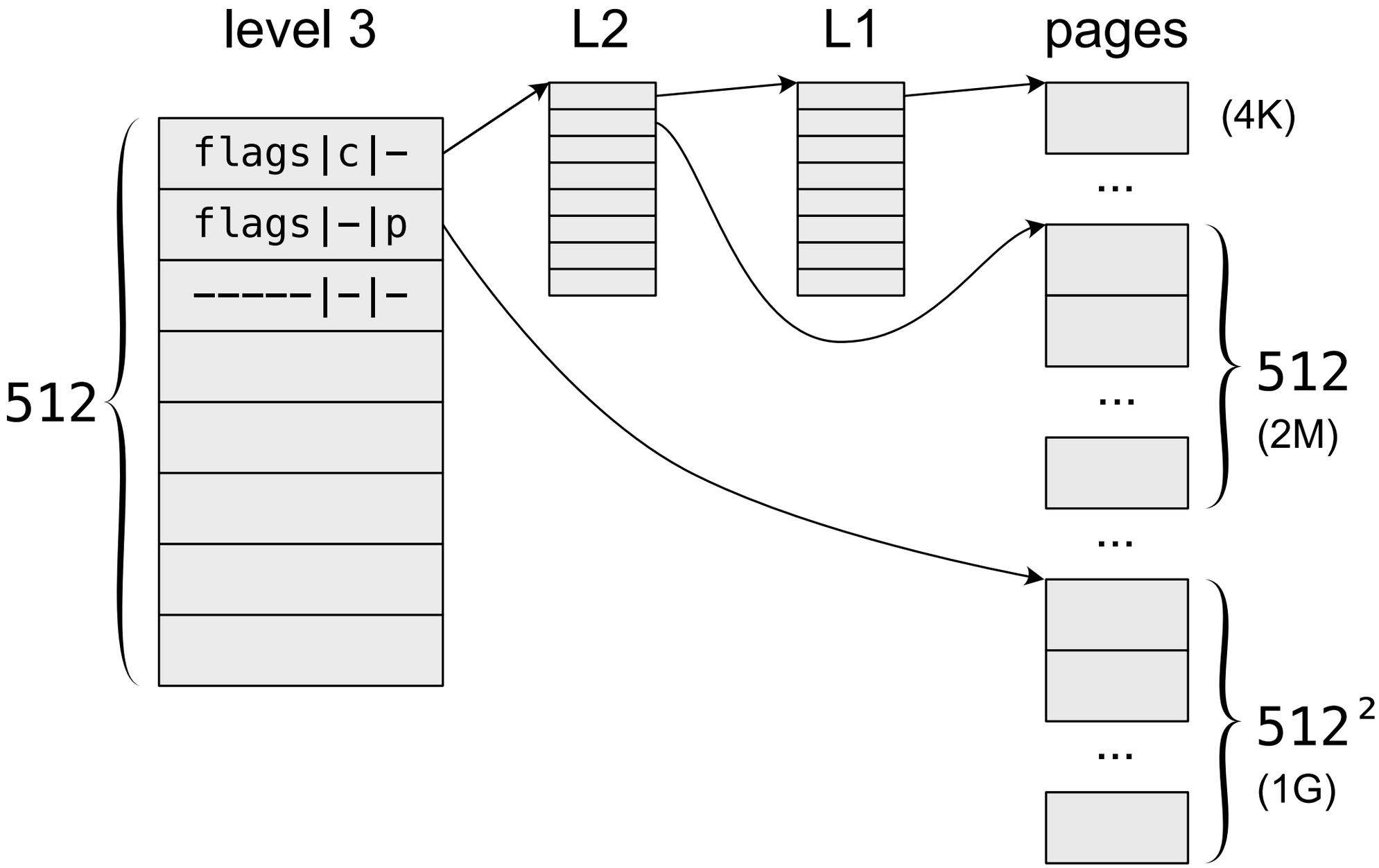
Lo(ck|g)-Free

Lock-Free Programming

- Lock-free algorithms guarantee system-wide progress
  - Individual threads are allowed to starve

- Implemented with Atomics
  - Compare-And-Swap (cmpxchg)
  - Jumping between valid states

Allocator - Page Table Architecture

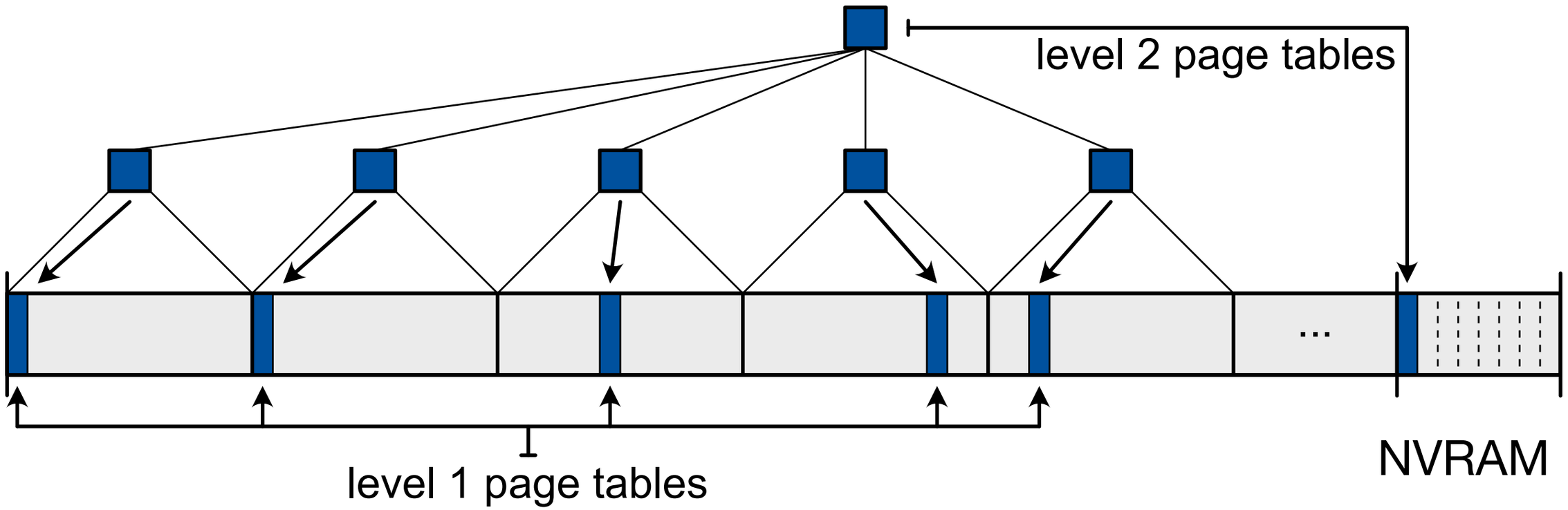


Implementation

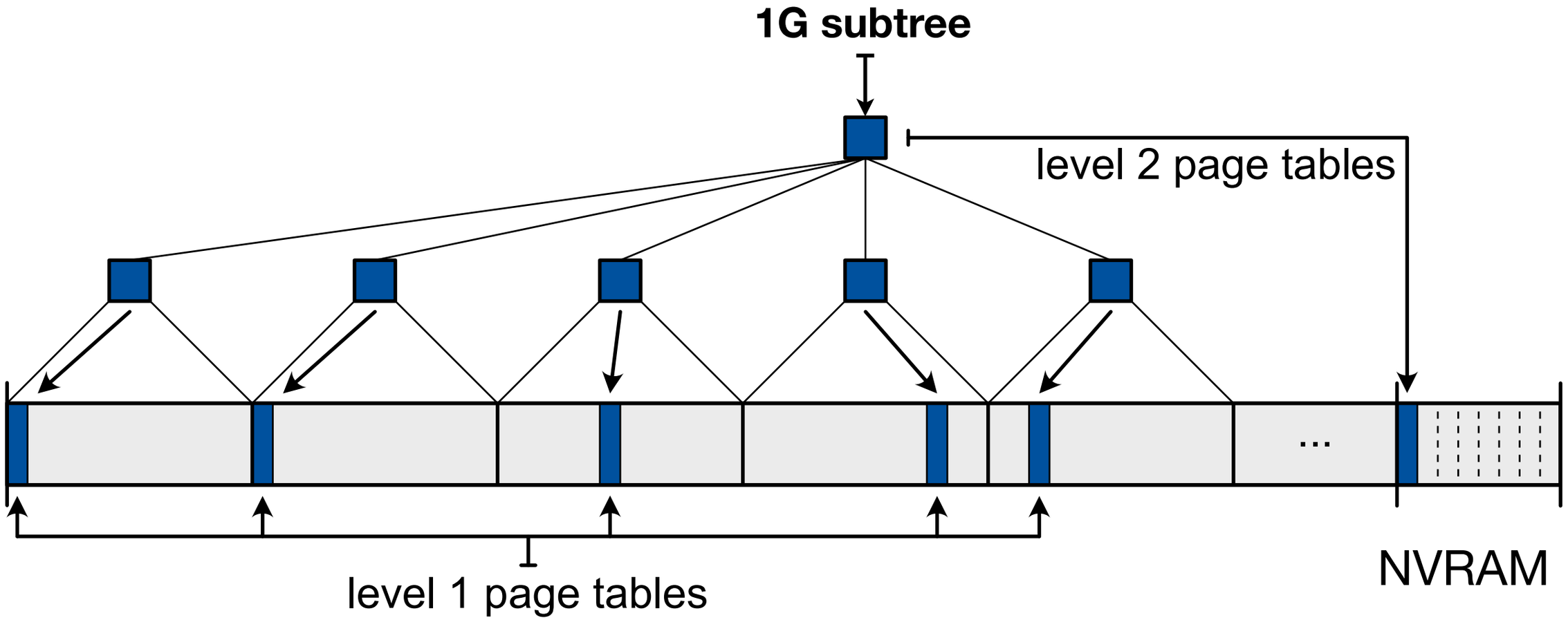
- **Lower:** Page allocations (persistent)
  - Level 1 and 2-page tables are stored in NVRAM
- **Upper:** Subtree reservations (volatile)
  - Rebuilt on boot

- Search algorithm
  - Iterate sequentially through page tables
  - Return first free page found

Lower: Page Tables



Lower: Page Tables



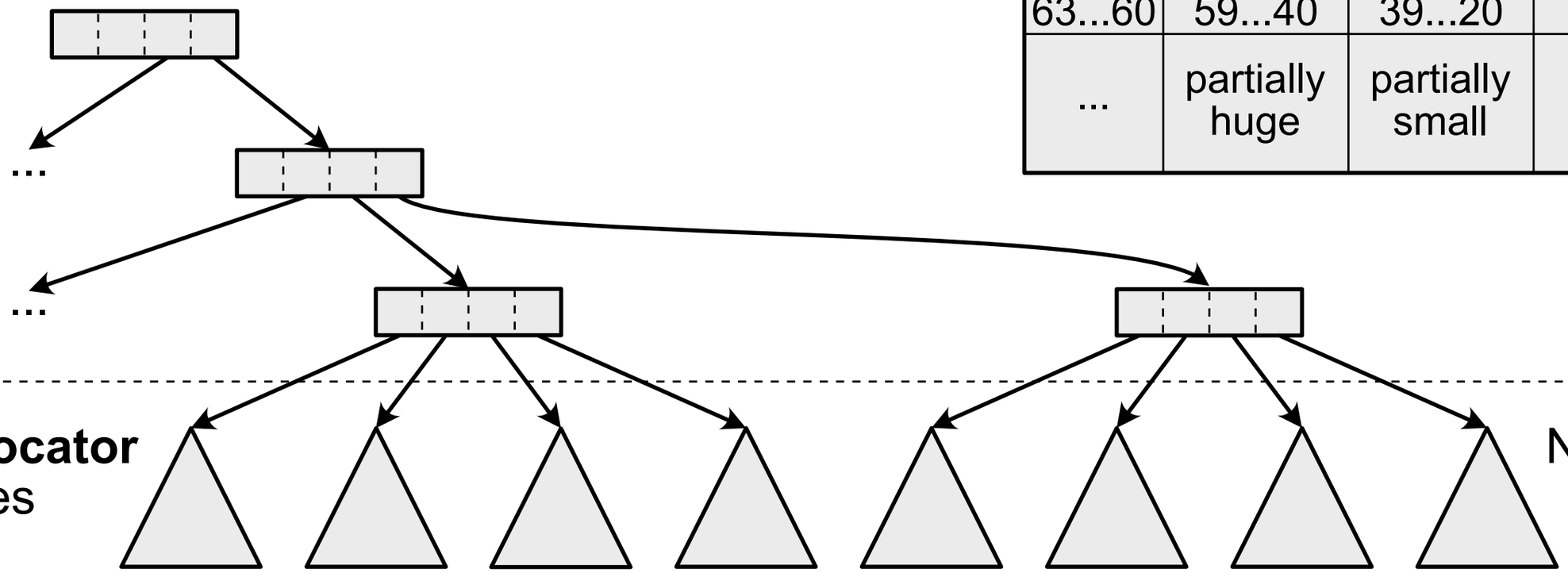
Upper: Table Approach

level 5

level 4 ...

level 3 ...

Lower Allocator
1G subtrees



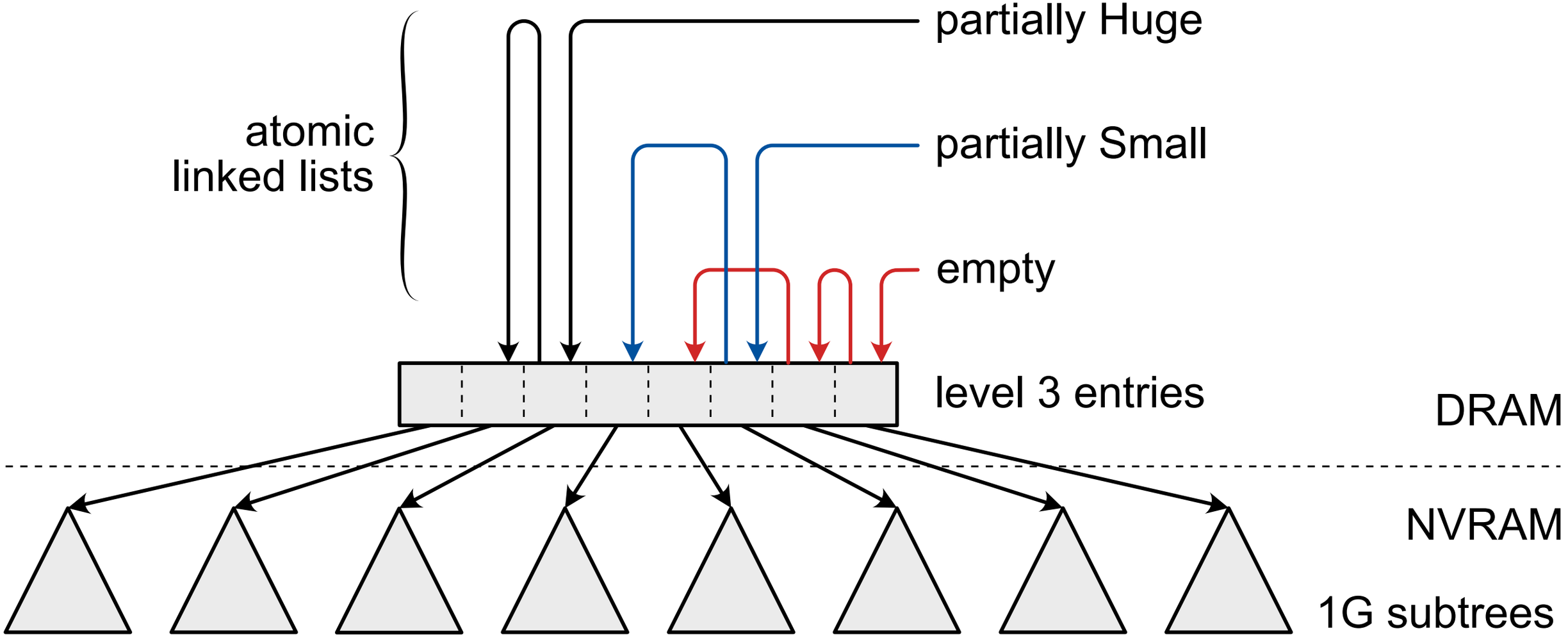
DRAM

NVRAM

level 4/5 entry

63...60	59...40	39...20	19...0
...	partially huge	partially small	empty

Upper: Array Approach



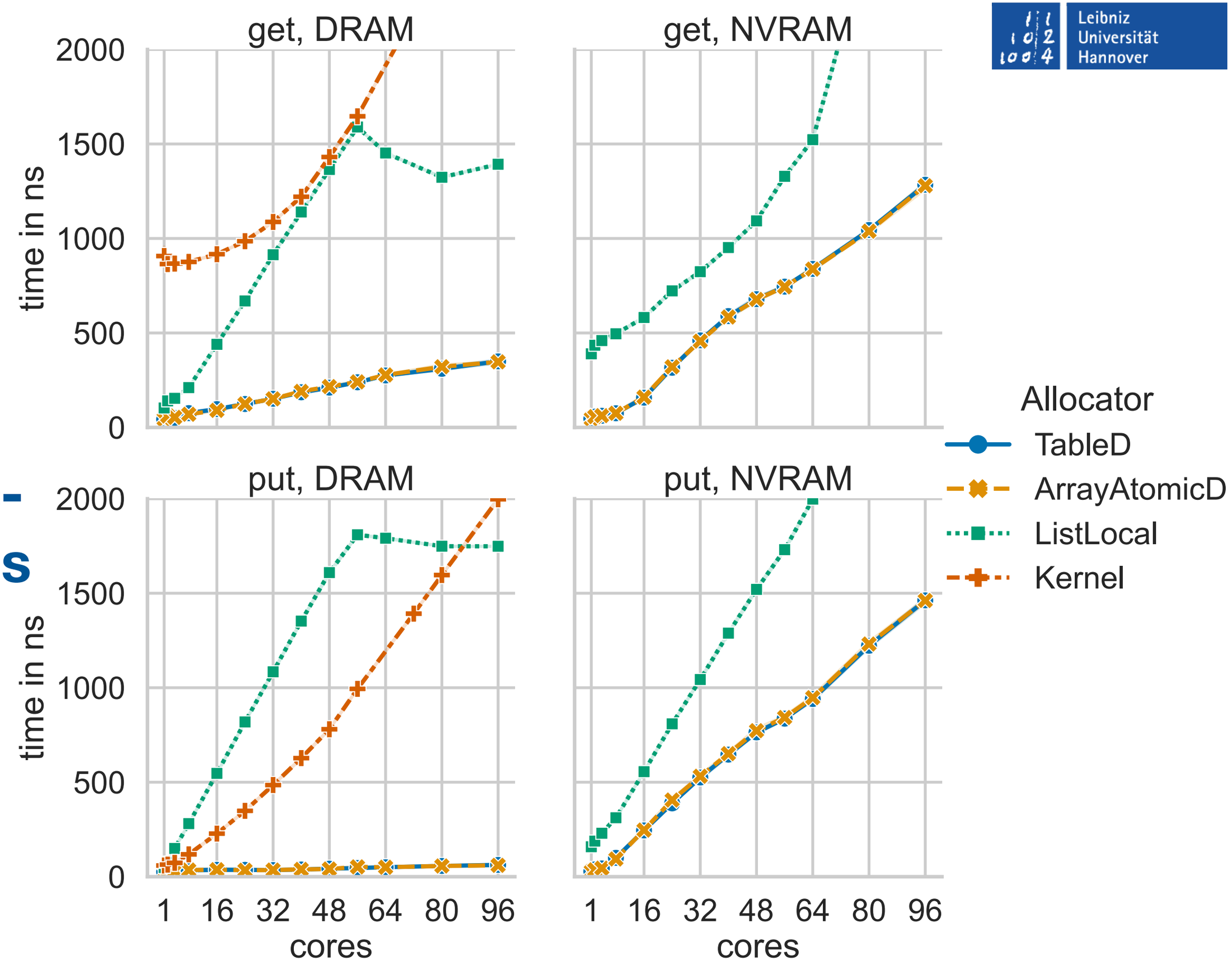
Evaluation

- Multi-threaded testing
  - Race conditions & (false-)sharing

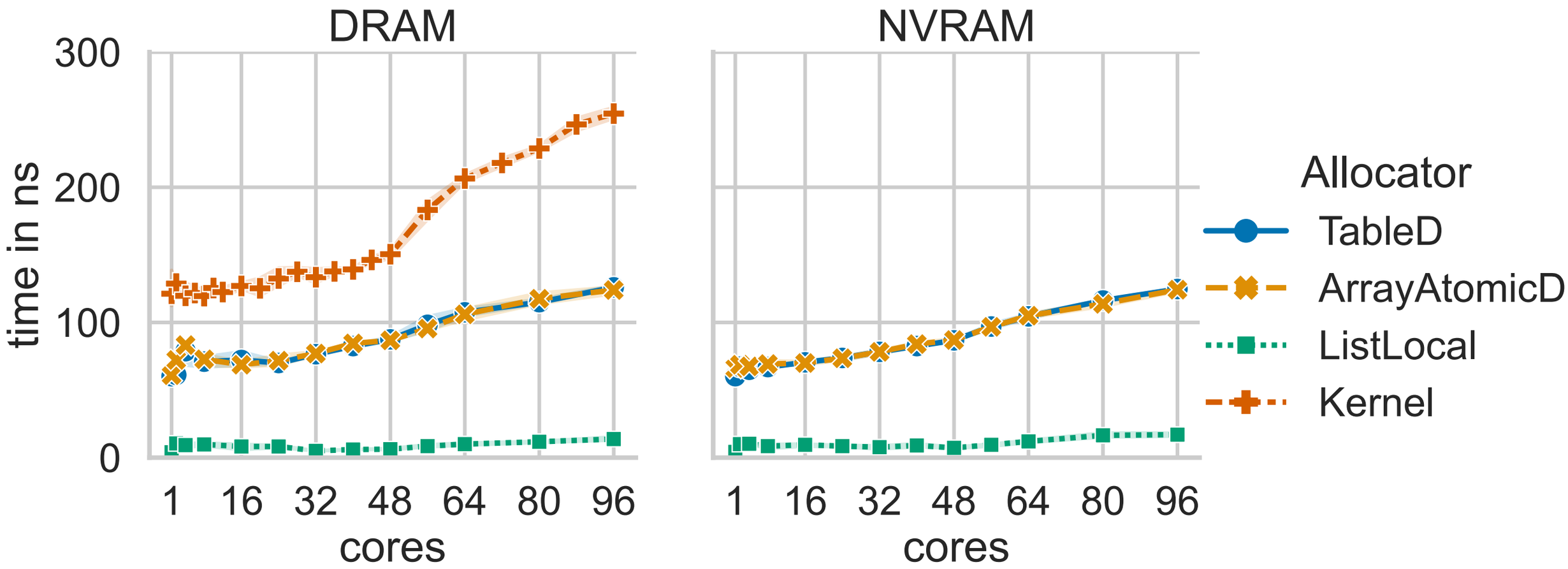
- Persistence testing
  - Using shared memory mappings & fork

- Microbenchmarks
  - Compared to the linux page allocator

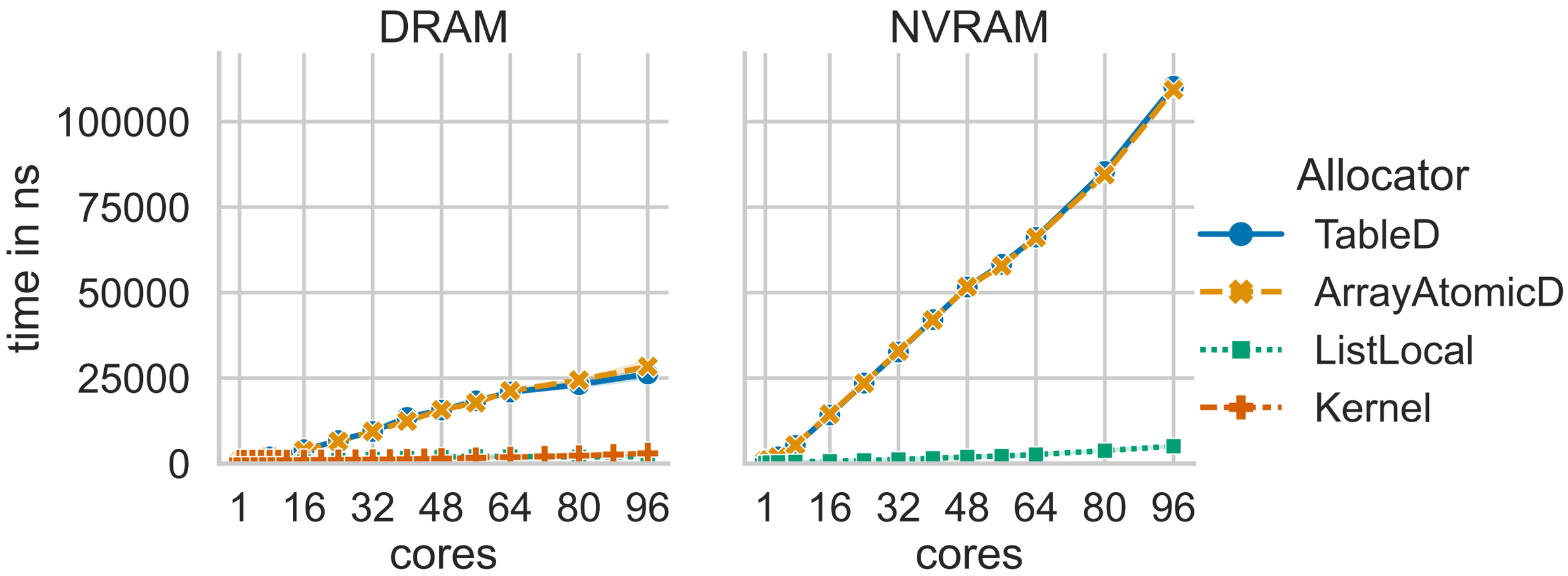
Bulk Alloc-ations



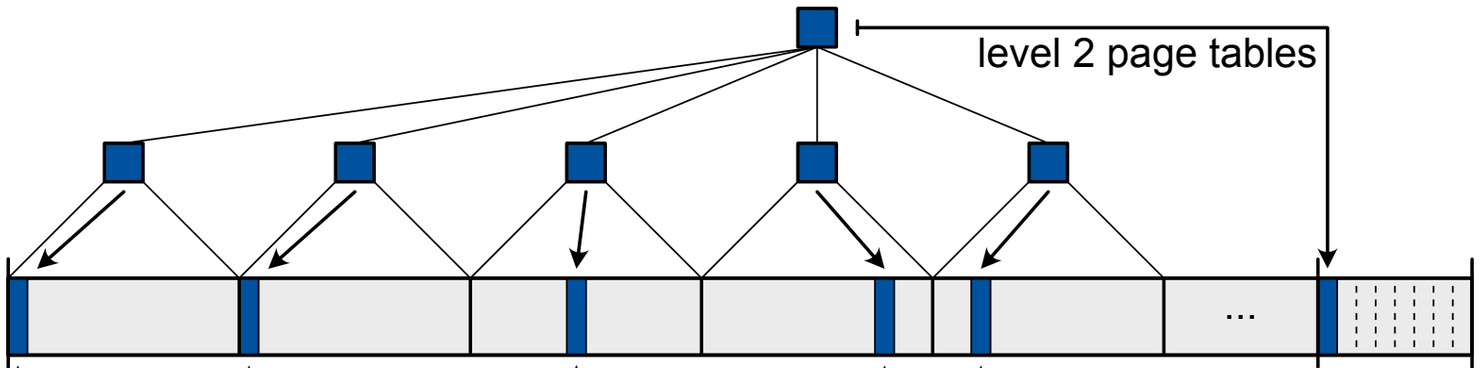
Repeat Free + Allocation

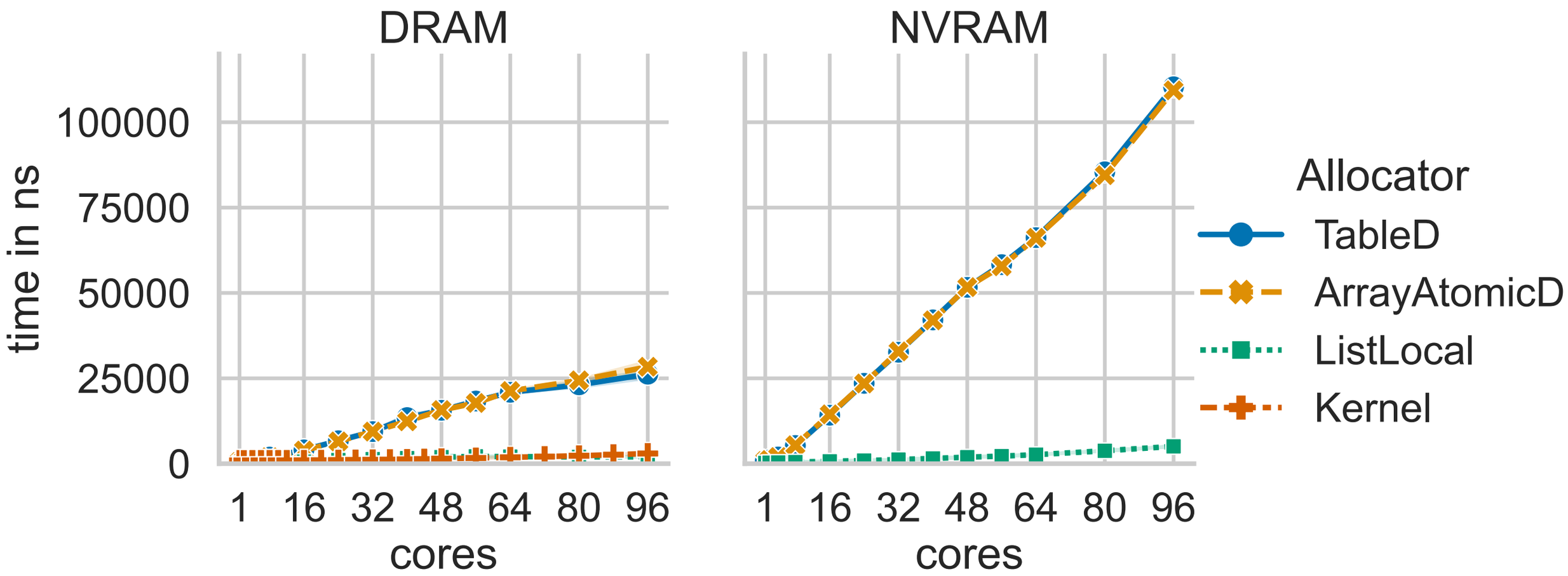


Random Free + Allocation

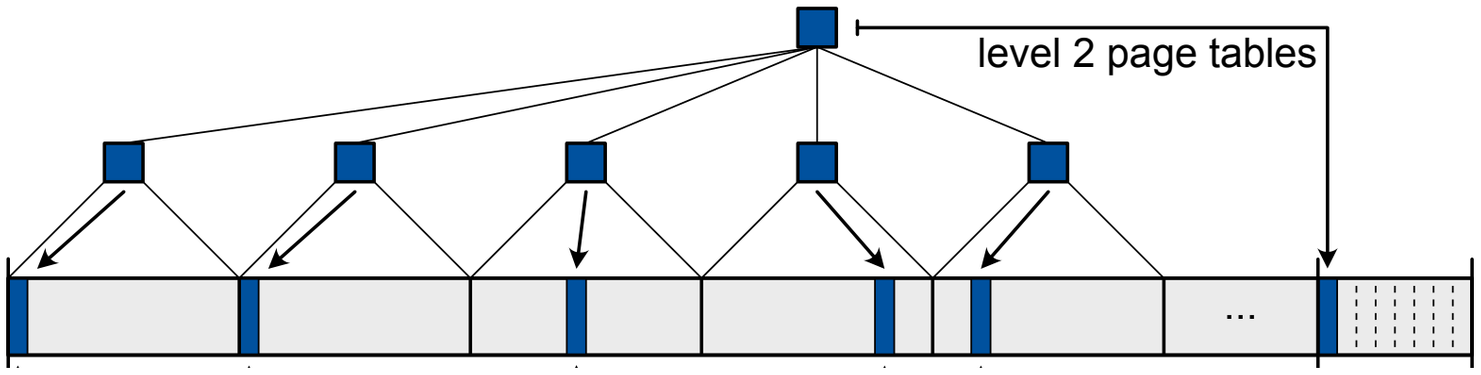


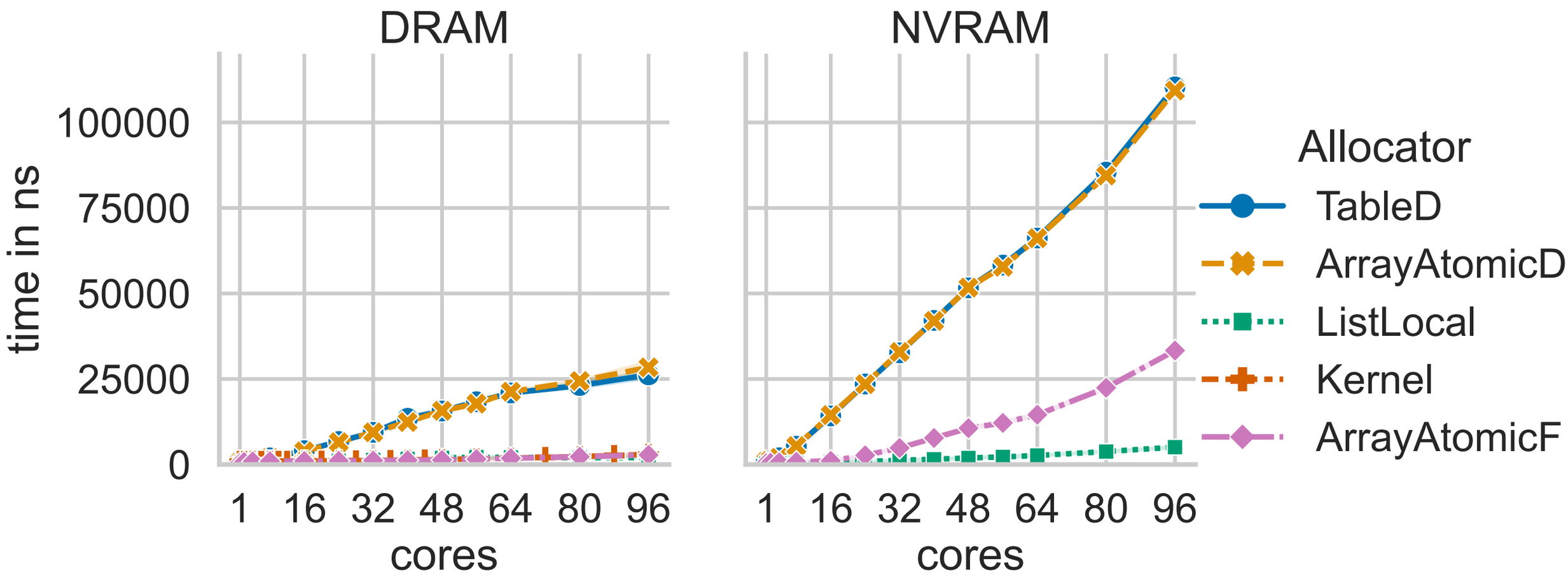
Random Free + Allocation





Random Free + Allocation





Summary

- ParPerOS Project
  - Morsel as a minimal address-space abstraction

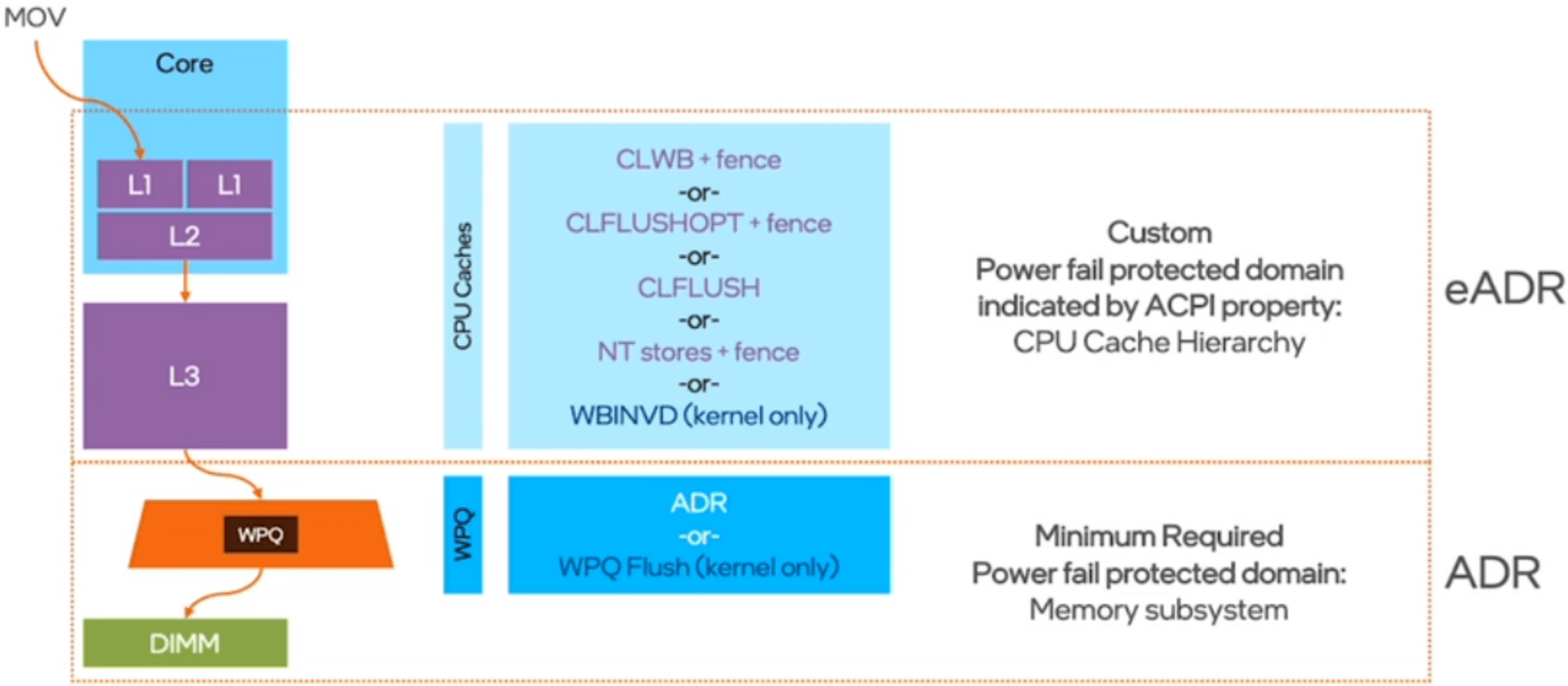
- Lo(ck|g)-free Allocator
  - Based on page tables and atomic updates
  - ✓ Lock and log free
  - ✓ Persistent & recoverable
  - ✓ Fast (scales well on multicore)

- Outperforms Kernel allocator for bulk & realloc benchmarks
  - Weakness: Random Frees

Bonus Slides

Discussion & Questions

Power Fail Protected Domains



https://software.intel.com/content/dam/develop/external/us/en/images/power-fail-protected-domains.png

Page Table Entries

level 3

63	62	61	60...20	19...0
page	huge	rese-rved	idx	pages

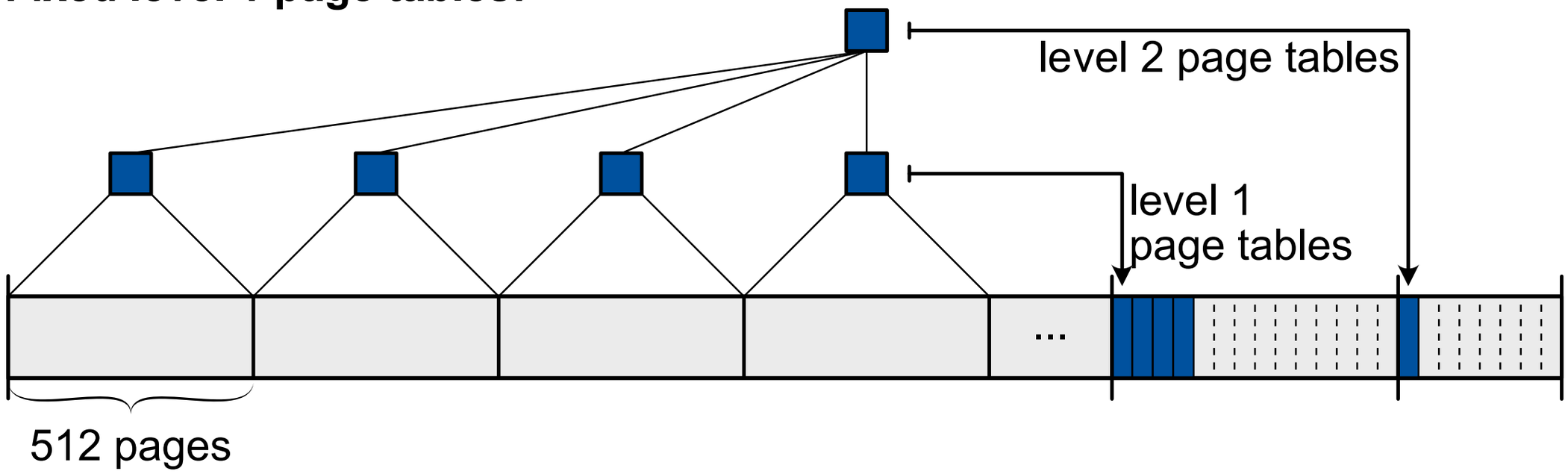
level 2

63	62	61...19	18...10	9...0
page	giant	...	i1	pages

level 1

63	62...0
page	...

Fixed level 1 page tables:



Dynamic level 1 page tables: