OSDI'24

哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

FairyWREN: A Sustainable Cache for Emerging Write-Read-Erase Flash Interfaces

Sara McAllister Yucong "Sherry" Wang Benjamin Berg* Daniel S. Berger†
George Amvrosiadis Nathan Beckmann Gregory R. Ganger

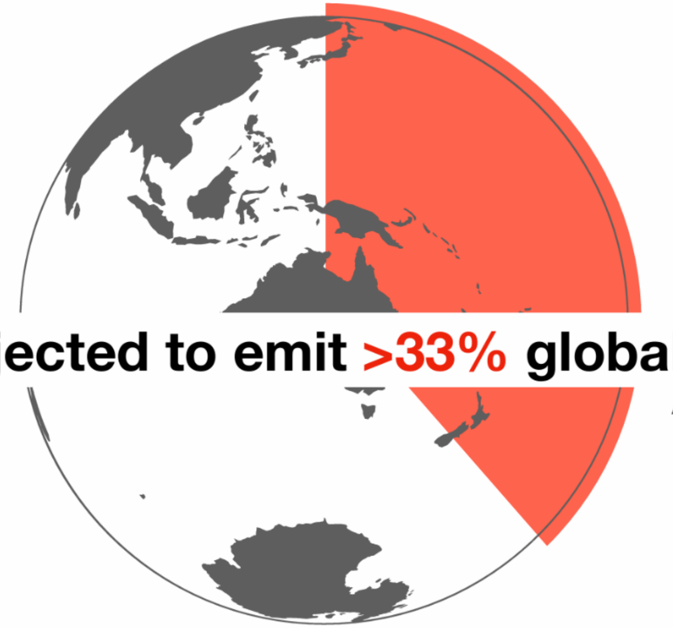
Carnegie Mellon University *UNC Chapel Hill †Microsoft Azure and University of Washington

Keywords: Flash Storage, ZNS Interface, Sustainable Cache

FairyWREN：用于新兴写-读-擦除闪存接口的可持续缓存

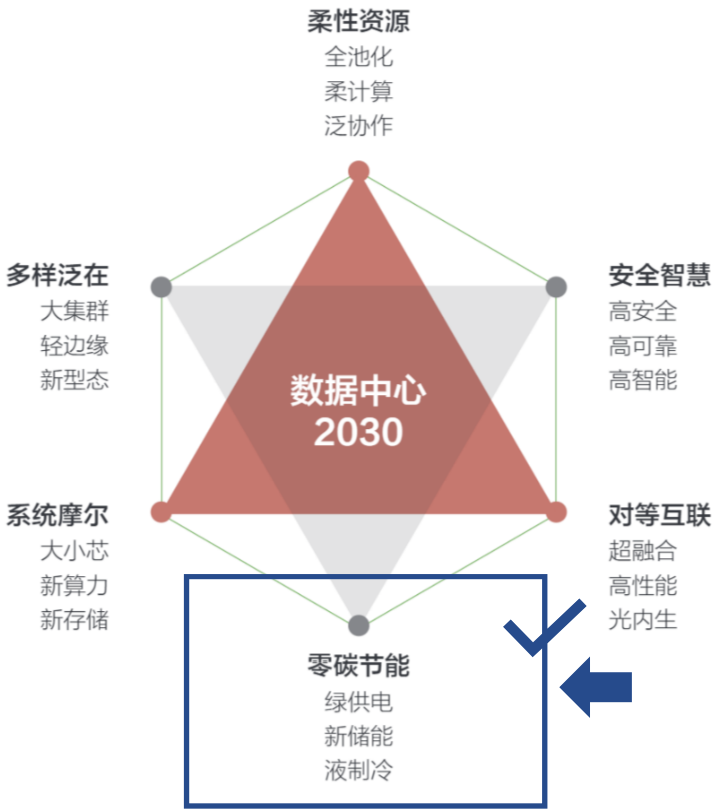
论文解读：Darong Yang

1.1 数据中心碳排放



Datacenters are projected to emit >33% global emissions by 2050

ACM TechBrief - Computing and Climate Change '21



40% of server emissions are storage

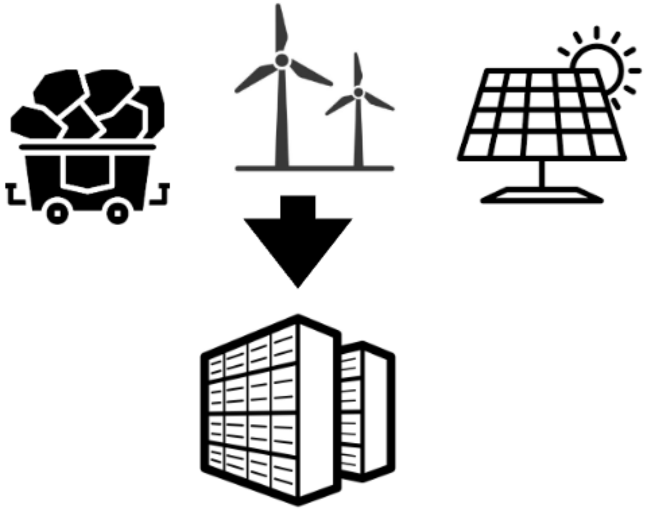
- 预计到2050年，数据中心碳排放达到33%以上
- 亚马逊、谷歌、Meta、微软都在寻求实现净零排放

1.1 数据中心碳排放

运营碳排放

Operational Emissions

Emissions from running the datacenter



隐含碳排放

Embodied Emissions

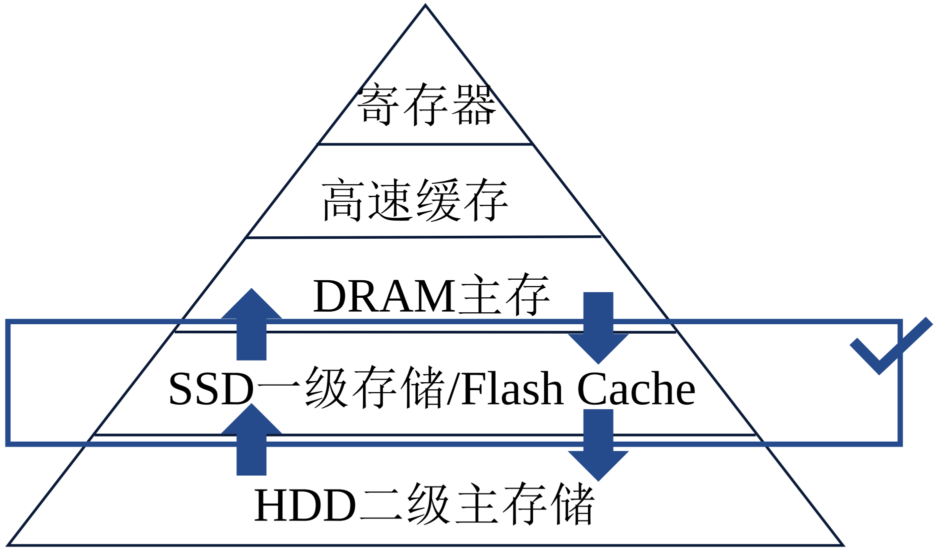
Emissions from manufacturing, transportation, raw materials, HW disposal

- 数据中心转向可再生能源，运营碳排放减少
- 隐含排放占主导地位（80% 以上）
- 隐含排放来源于一次性生命周期事件：器件的生产、功耗

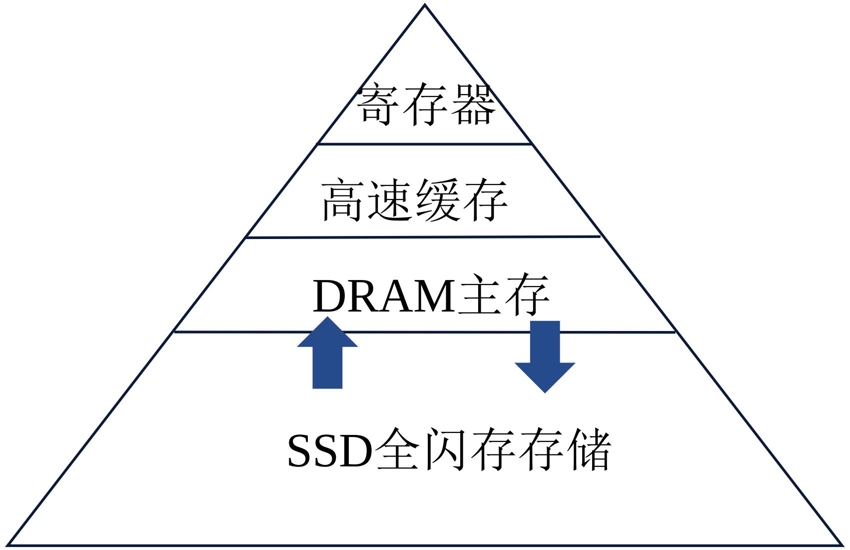
延长寿命 + 更低功耗硬件

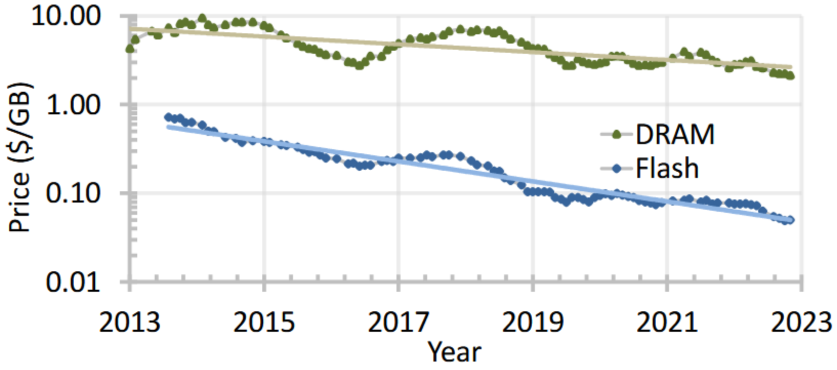
1.2 闪存缓存

闪存缓存



全闪存存储





Type	Brand	Cost-Per-GiB ($)
DRAM	Crucial DDR4-2400 (16 GiB)	3.75
SSD	Intel SSD 545s (512 GiB)	0.24
HDD	Seagate BarraCuda (2 TiB)	0.025

- 对大规模数据中心，全闪成本高
- DRAM做缓存受限于成本
- 闪存缓存是大规模数据中心常用方案

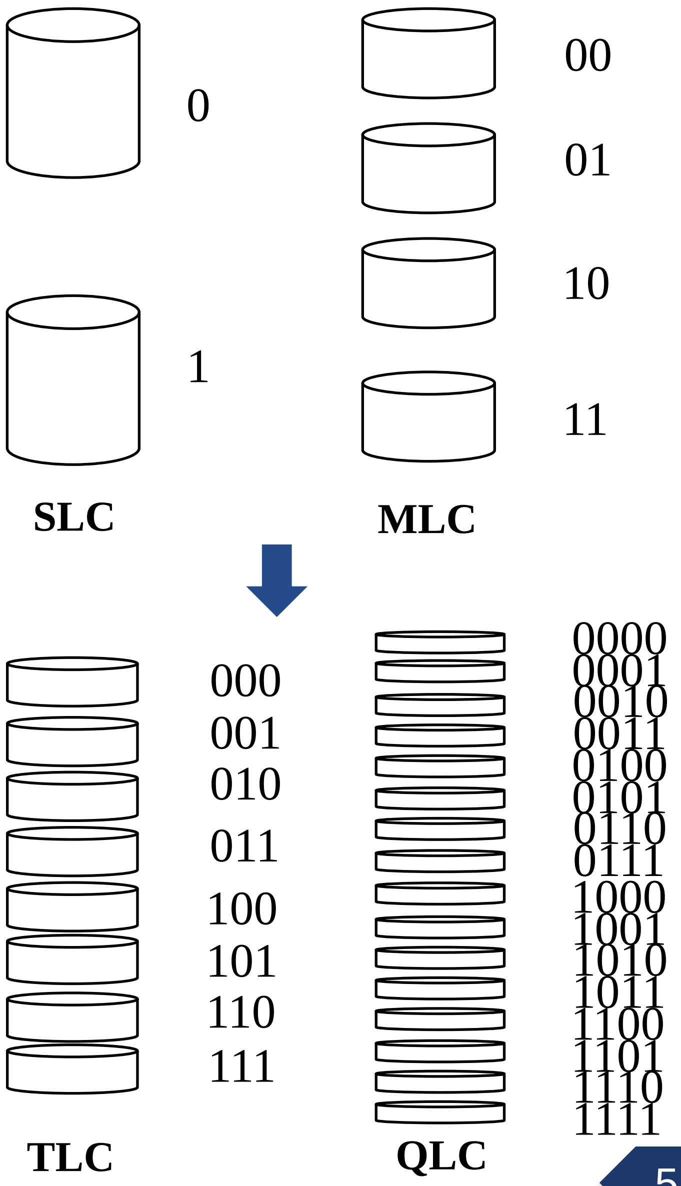
2.1 闪存助力碳减排

1.比DRAM碳排放低

- 现有工作聚集处理器设计中的隐含排放
- 然而关注较少的内存和存储占服务器排放的46%和40%
- 相比DRAM，闪存成本低、减少高达12x的碳排放、更低功耗

2.高密度趋势进一步助力碳减排

- 闪存的存储单元从SLC，发展到QLC
- 相同的硅片存放更多比特，减少隐含碳排放



2.2 闪存设备面临的挑战

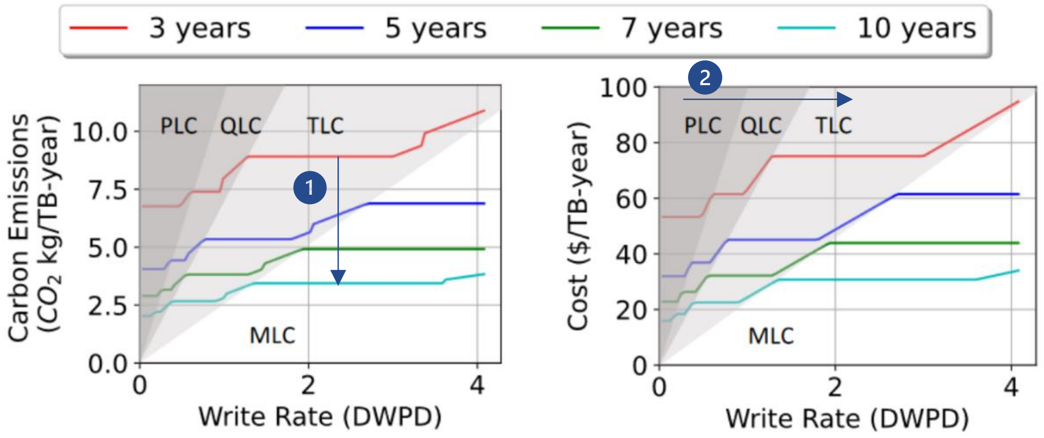
Intro-duction
Moti-vation
Design
Evalua-tion
Conclu-sion

1.数据中心的服务器寿命

- 传统数据中心的服务器寿命为3年
- **可持续发展**，服务器寿命普遍延长至**6年**（Microsoft和Meta等）

2.闪存设备的寿命挑战

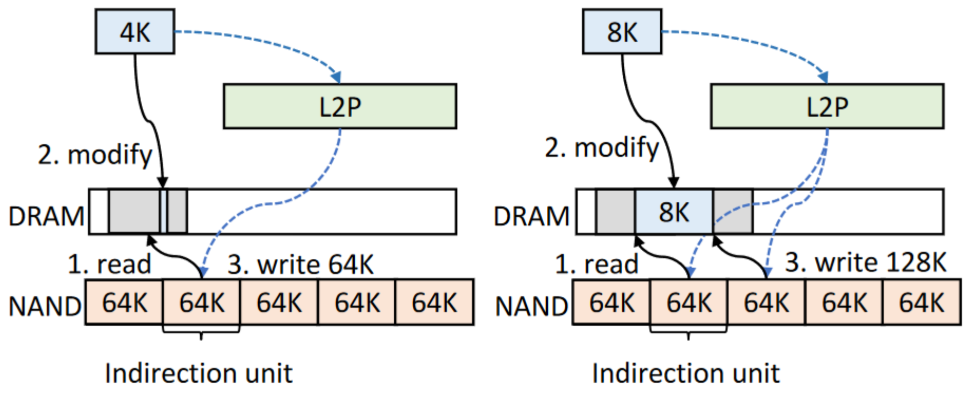
- 闪存的**写入寿命有限**，是延长服务器寿命最大挑战
- 从SLC到QLC，闪存密度增加但寿命严重下降
- 保证6年写入寿命，只能限制**每日写入量很小**



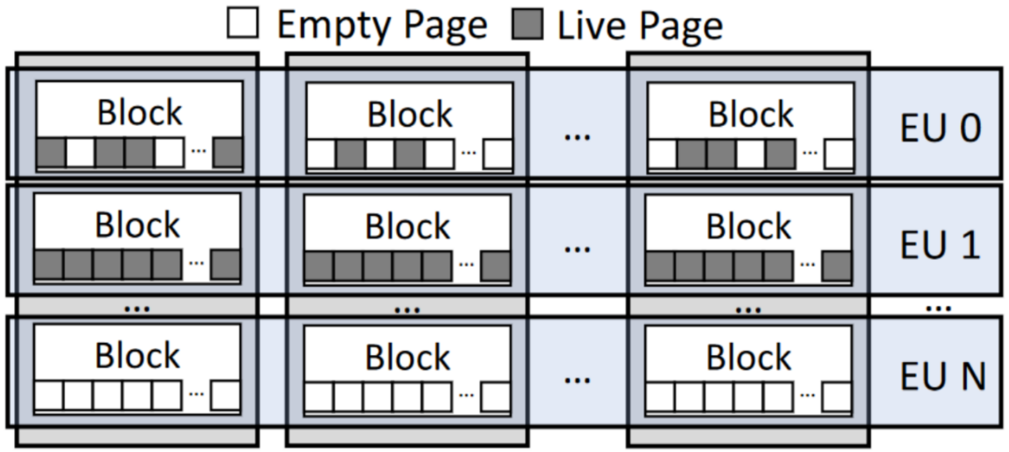
(a) Carbon emissions. *(b) Cost.*

2.2 闪存设备面临的挑战

3.闪存设备的写放大



(a) Missized write (b) Misaligned write



应用级写入放大（ALWA）

- 写入数据（100B）和写入单元（64KB）的失配
- “读”－“修改”－“写”

写入放大：数据重复写

设备级写入放大（DLWA）

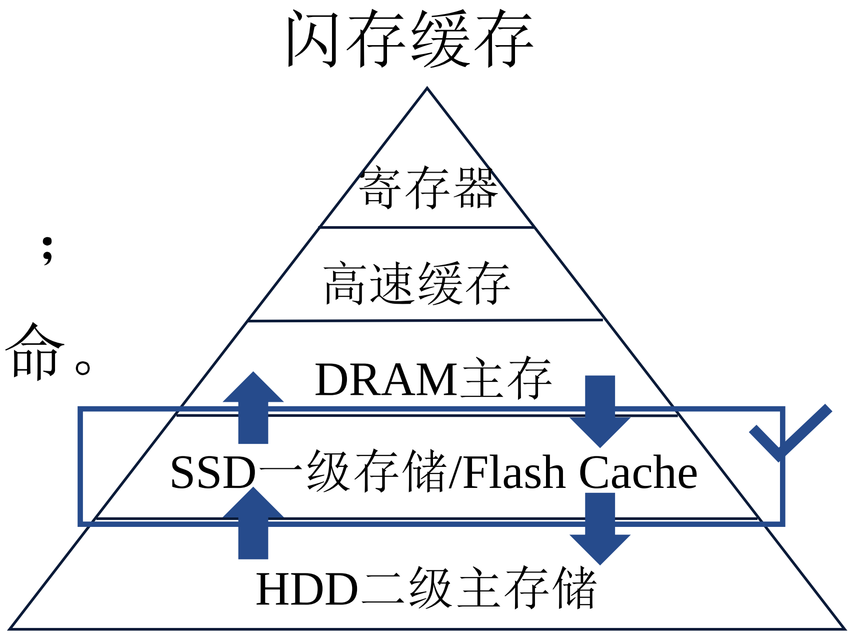
- 写入单元（4KB）和擦除单元（GB）的失配
- “读取”－“迁移”－“擦除”

2.3 现有闪存缓存

闪存缓存介绍

可持续闪存缓存要考虑：

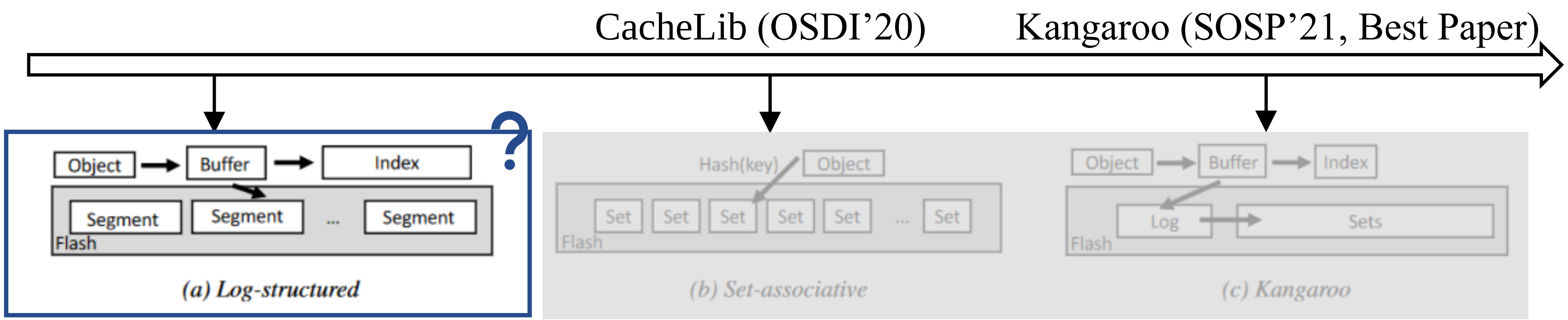
- （1）减少闲置空间，以减少无益的排放；
- （2）减少对象元数据的DRAM使用（达10GB）；
- （3）减少写入/磨损速率，以延长设备的寿命。



设计思路不同：

- 与DRAM缓存设计思路不同（寿命、写入放大）
- 与键值存储设计思路不同（快速删除、过度配置）

2.3 现有闪存缓存



1.日志结构闪存缓存

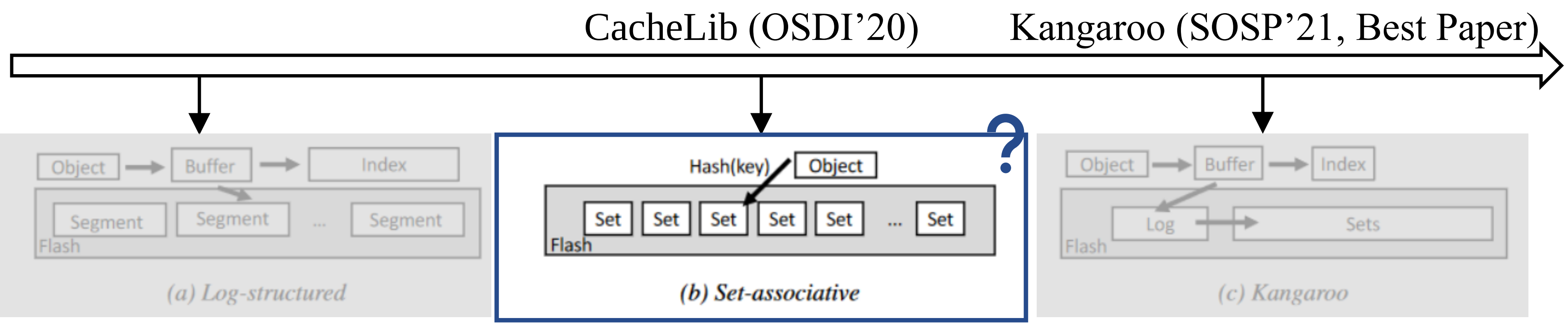
原理

- 采用日志结构，顺序追加写
- DRAM中缓存segment，写满后落盘
- 需要为每一个对象建立DRAM索引

缺点

- 闪存缓存经常要缓存小对象（100B）
- 小对象的DRAM索引会非常大
- 75GB DRAM，2TB的100B对象

2.3 现有闪存缓存



2.组相连闪存缓存

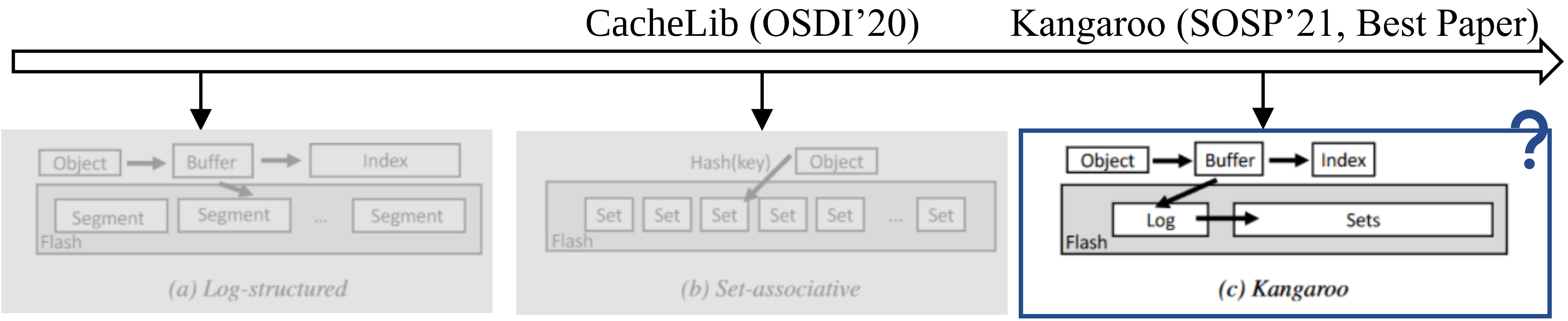
原理

- 采用哈希函数替换DRAM索引
- 将对象映射到唯一集合
- 集合写满了后发生缓存替换

缺点

- 带来小对象的随机写
- 写小对象(100 B)，至少一个页(4 KB)
- 写入放大：ALWA是40x；DLWA是2x−10x

2.3 现有闪存缓存



3.分层结构闪存缓存

原理

- 上述两者的tradeoff
- 使用日志结构的KLog（约5%）充当小对象的写缓冲，减小ALWA
- 使用哈希组关联的Kset（约95%）充当主存储，减少日志的DRAM索引

缺点

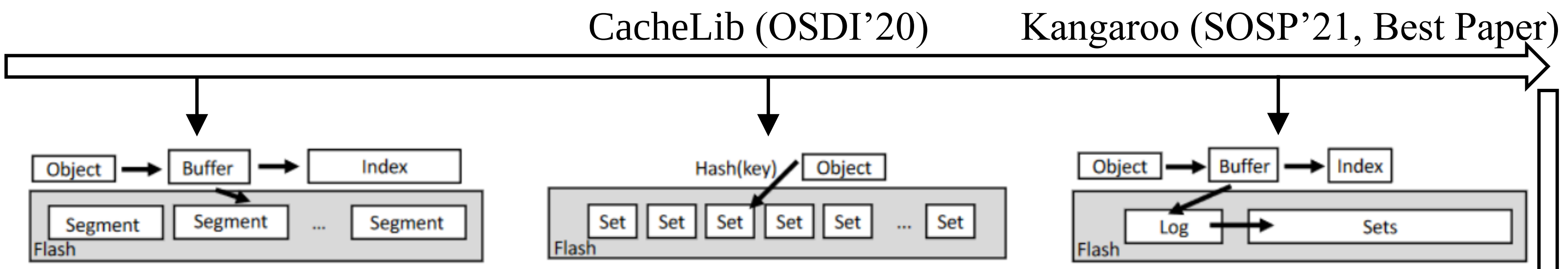
- Kset仍会带来4KB随机写入
- 没有解决DLWA的问题

?

DLWA问题的关键在？

2.3 现有闪存缓存

总结



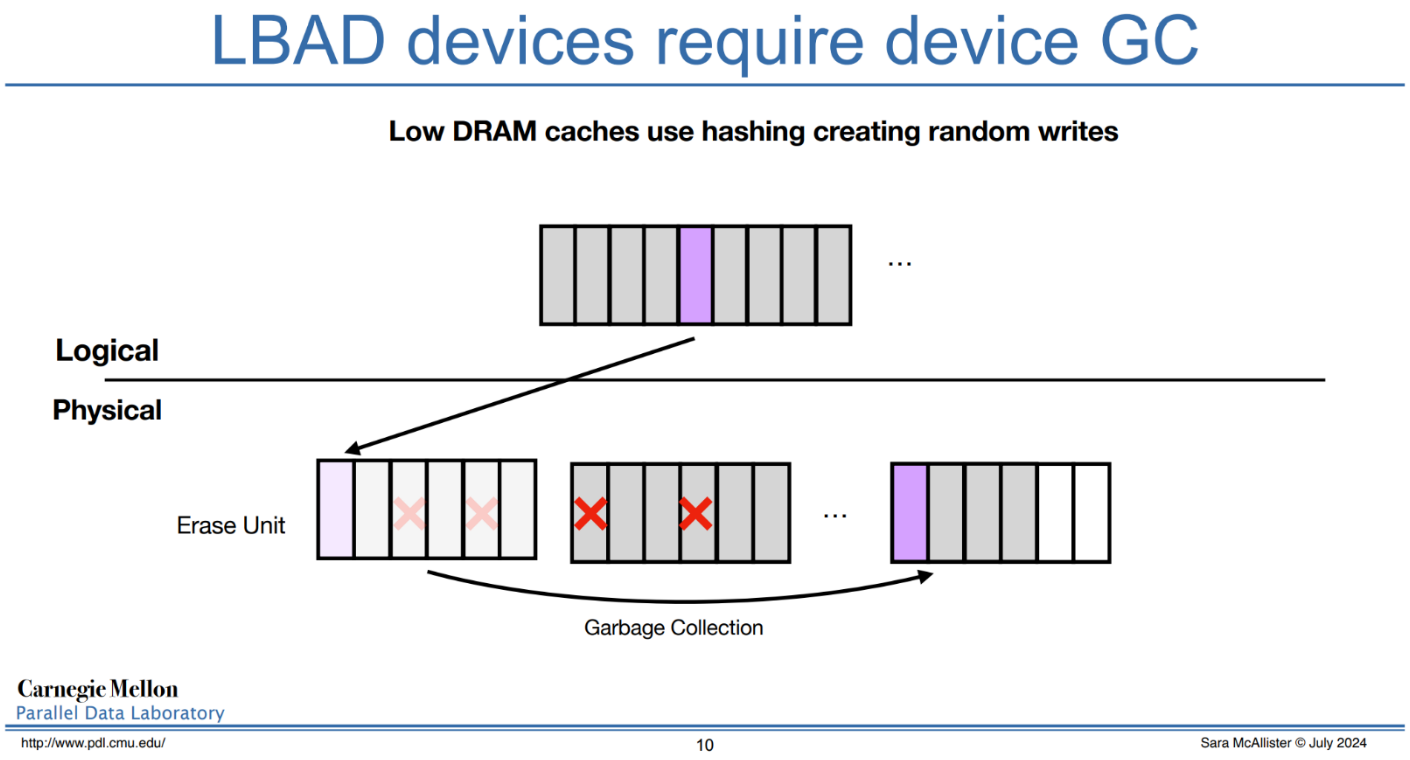
(a) Log-structured (b) Set-associative (c) Kangaroo

	Flash caches should minimize ...			
	Unused flash	DRAM	ALWA	DLWA
Key-value stores	✗	✓	✓	✓
Log-structured caches	✓	✗	✓	✓
Set-associative caches	✗	✓	✗	✗
Kangaroo [67]	✓	✓	✓	✗
FairyWREN	✓	✓	✓	✓

FairyWREN (OSDI’24)

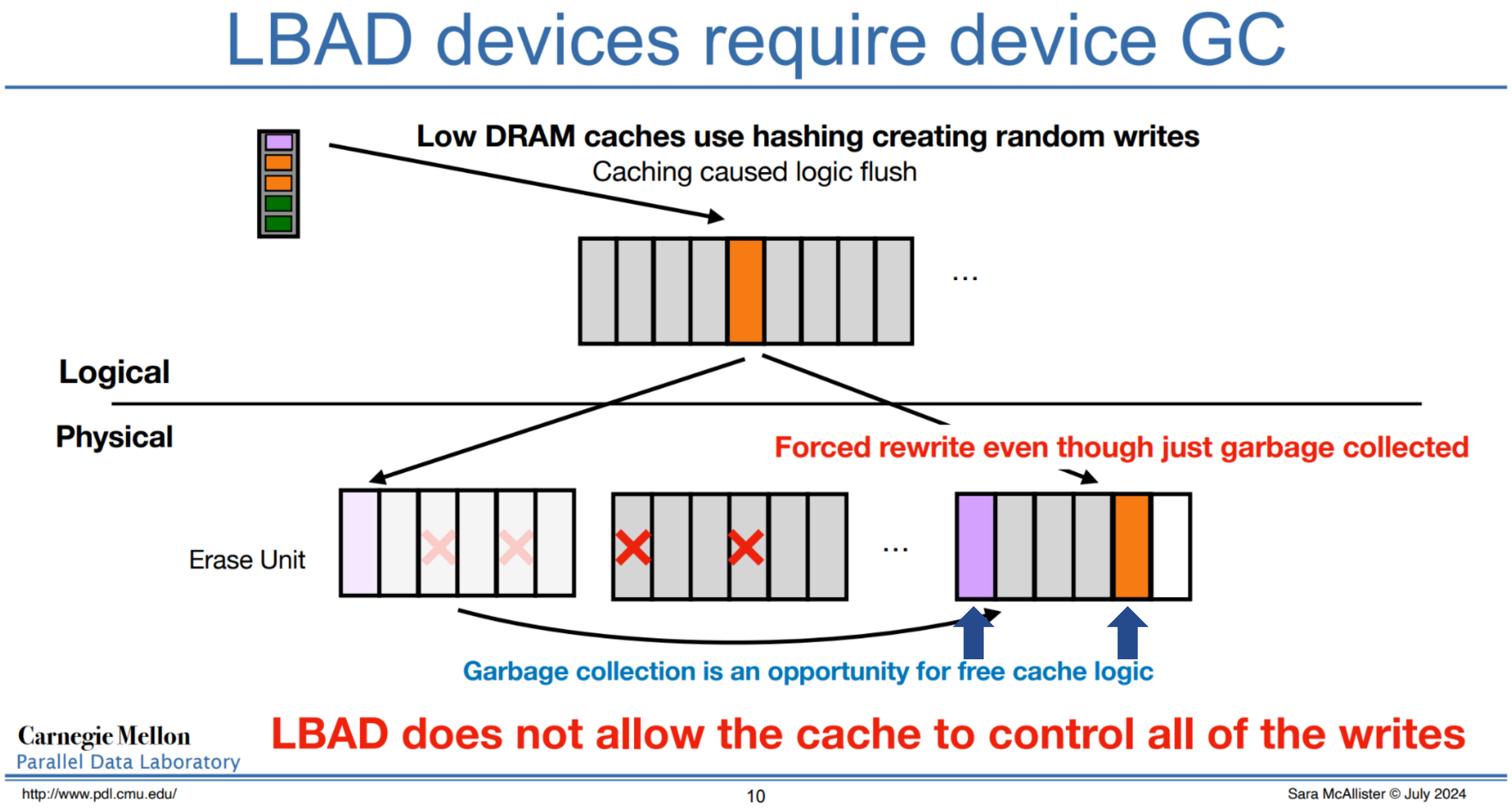
- FairyWREN实现：闲置空间消除、最小化DRAM使用、应用级写放大消除、设备级写放大消除

2.4 关键原因：LBAD接口的低效



- LBAD接口屏蔽了物理层，只暴露逻辑层
- Kangaroo基于LBAD接口，缓存驱逐/替换操作基于**逻辑层**
- LBAD的垃圾回收操作基于**物理层**，两者互不知情

2.4 关键原因：LBAD接口的低效



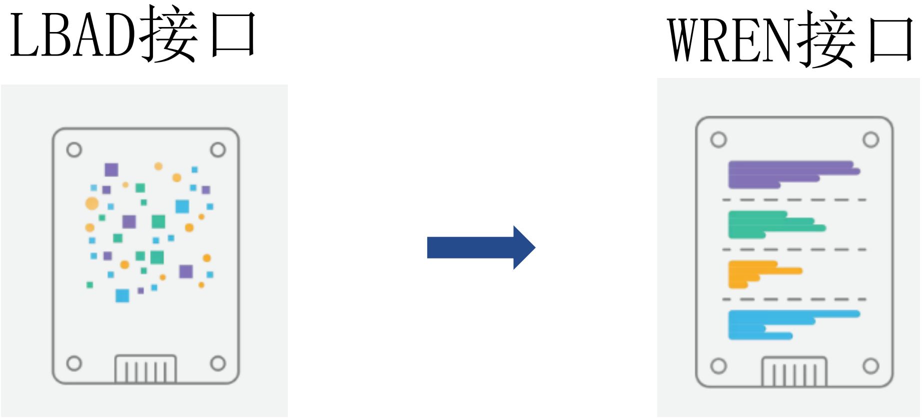
- GC刚搬迁完，数据就被替换而无效，无效搬迁
- 密集覆写的闪存缓存来说，这个问题尤为显著
- 作者认为垃圾回收是缓存替换的一个机会

2.5 新的机遇：WREN接口

- 理想的闪存缓存接口：允许缓存控制所有写入(包括 GC)
- 新兴的ZNS、FDP等接口出现，都属于这种接口

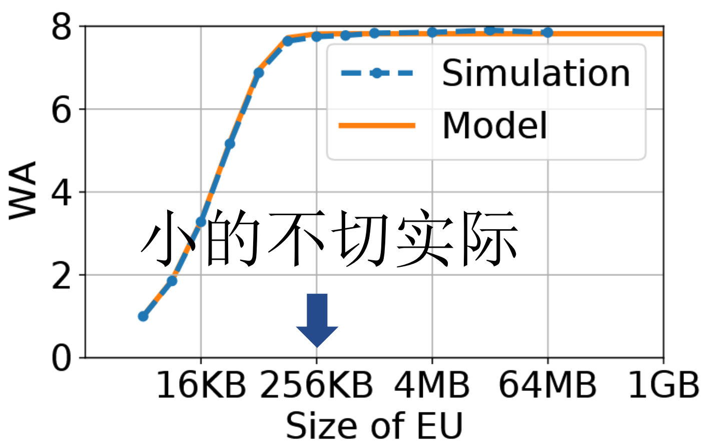
WREN接口定义

- （1）WREN操作
- （2）擦除要求（可控且整块）
- （3）多个但有限活跃擦除单元EU

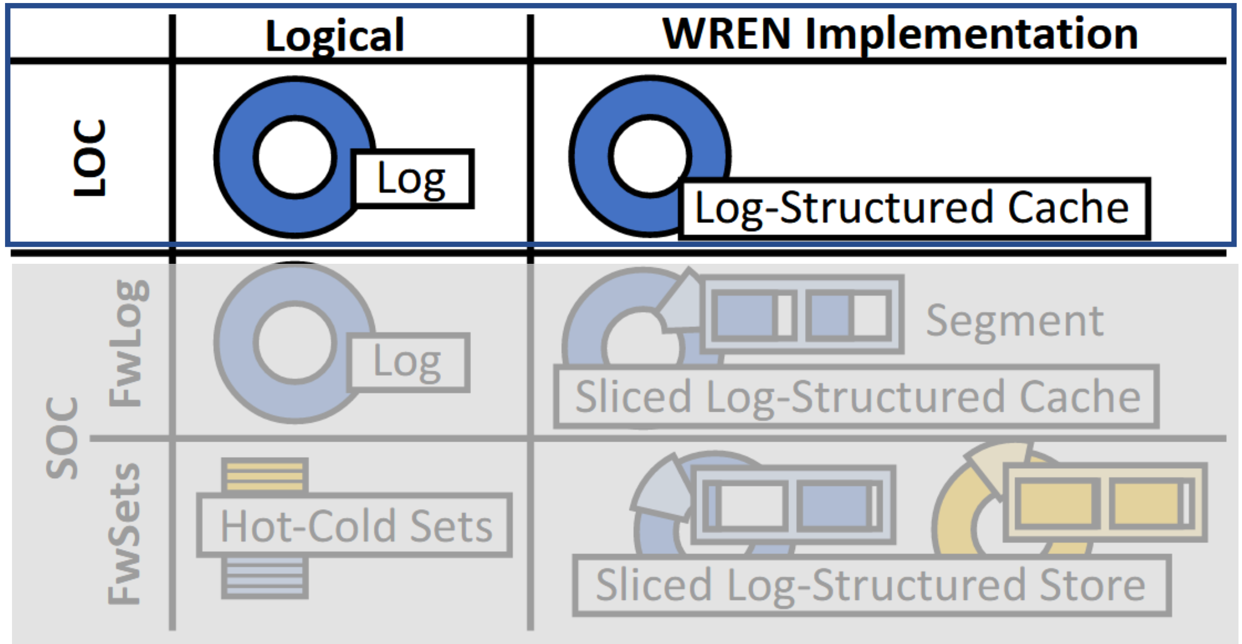


WREN接口软硬件协同

- 只靠WREN接口是不够的，需要精心设计、否则只是转移GC
- 减小擦除单元大小也是不可行的



LOC模块（大对象，>2KB）



DRAM：

- EU大小的segment缓存
- 大对象索引

LOC闪存：

- 日志化结构缓存
- segment大小定为一个EU（WREN接口）

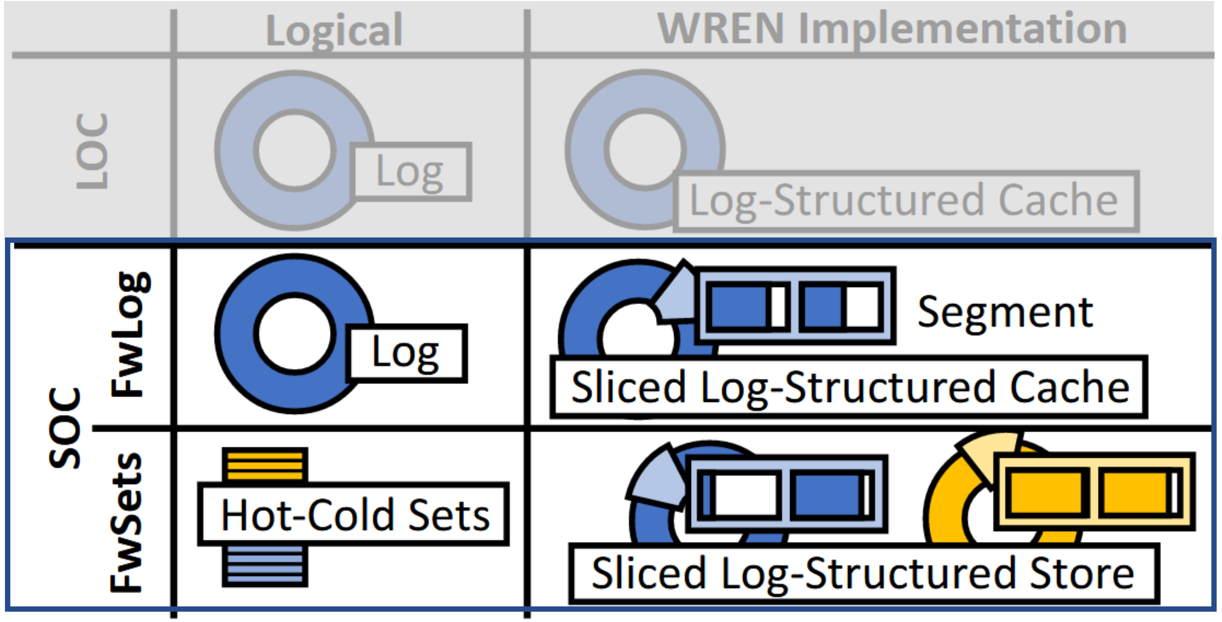
缓存操作：

- 与日志结构缓存相同
- 插入：segment缓存，建立索引，满了落盘LOC闪存
- 查询：查询索引得到地址，直接从LOC闪存中读取

LOC插入的替换逻辑 ?

3.1 FairyWREN总体结构

SOC模块（大对象，>2KB）



缓存操作：

- 插入：先插入到FwLog，FwLog满了会回收到FwSet中，进一步导致FwSet的替换
- 查询：类似于LOC方式查询FwLog，hash得到对象的Set，扫描Set找到对象

DRAM：

- FwLog的segment缓存
- FwLog的小对象索引
- FwSet的Set索引

FwLog：

- 只占5%
- 日志化结构缓存

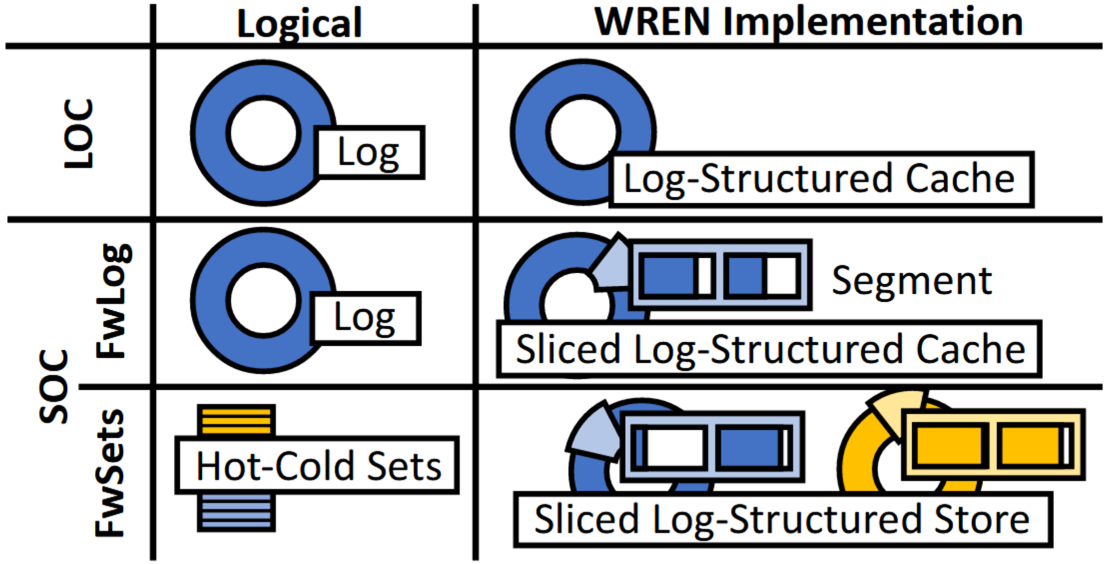
FwSet：

- 占95%
- 采用组相连缓存
- 以Set为单位追加（WREN接口）

？

SOC插入的替换逻辑

3.2 垃圾回收和缓存准入的统一



LOC插入的替换逻辑？ →

SOC插入的替换逻辑？ →

LOC模块

- 由于单层且segment与EU的对齐设计
- 通过LRU或FIFO移除整个segment，即EU即可

SOC模块 Key Insight：

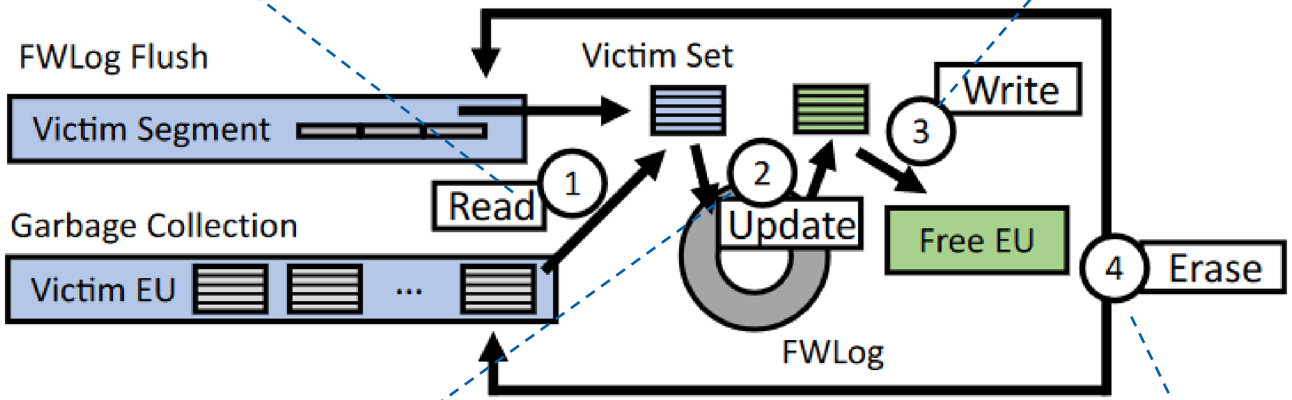
- 将**垃圾回收**和**缓存准入**进行统一
- SOC的嵌套打包算法

3.2 垃圾回收和缓存准入的统一

SOC模块：嵌套打包

①EU选择：取决于FwLog和FwSet谁写满

③Set重组：检索FwLog中Set的对象，重组为新Set



②Set散列：将选中的EU的所有对象散列为若干Set

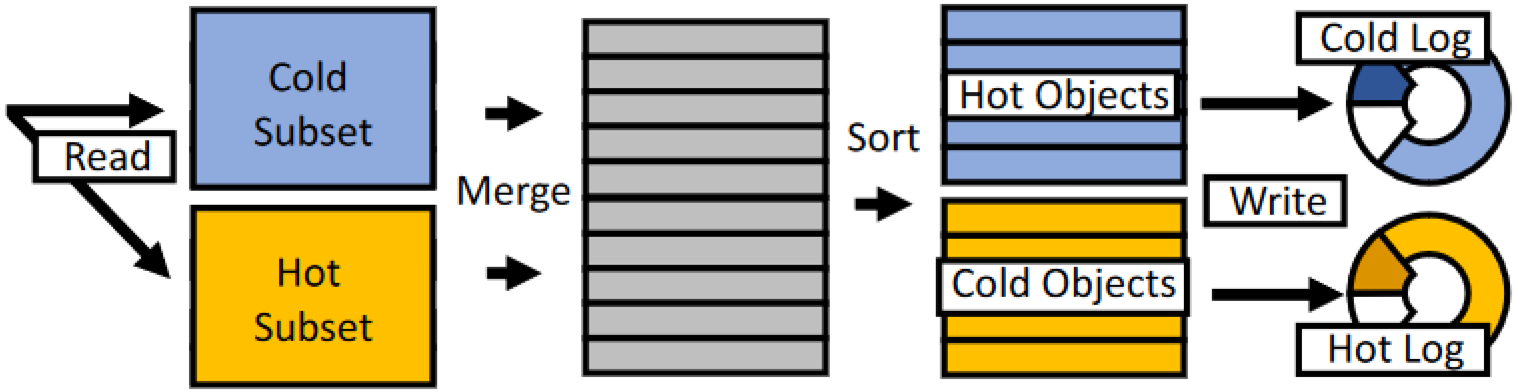
④重写与擦除：重新追加写入到FwSet，擦除上述EU

核心：让闪存缓存**控制垃圾回收**，避免重复写

3.3 KwSet的冷热对象分离

出发点： 每次从FwLog插入一个对象到FwSet，都会带来整个Set的重写

冷热对象分离

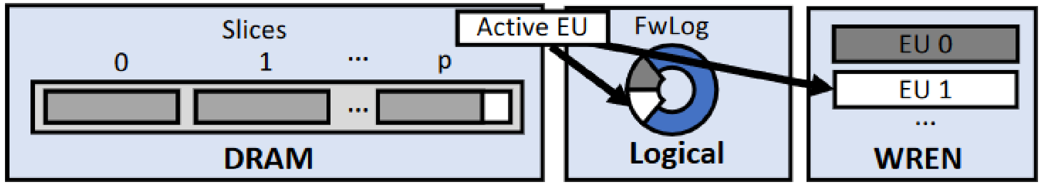


- Set划分：将KwSet划分为两个SubSet
- 对象识别与放置：热对象要放到冷SubSet中
- 冷热重分类
- 优化讨论：带来接近一半的写放大优化

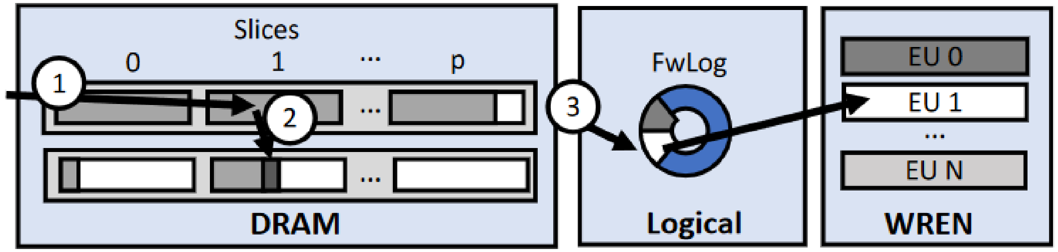
3.4 DRAM使用优化

出发点：　带来了FwLog中的Set索引的DRAM开销，冷热划分再翻倍

DRAM使用优化



(a) Naïve FwLog partitioning.



Component	Kangaroo	Naïve SOC	SOC
Log total	*48 bits/obj*	*48 bits/obj*	*48 bits/obj*
Set index	–	≈ ***3.1 b***	≈ ***1.4 b***
Sets (other)	4 b	4 b	4 b
Sets total	*4 bits/obj*	*7.1 bits/obj*	*5.4 bits/obj*
Log metadata	≈ 0.8 b	≈ 0.8 b	≈ 0.8 b
Log size	5% = 2.4 b	5% = 2.4 b	5% = 2.4 b
Set size	95% = 3.8 b	95% = 6.7 b	95% = 5.1 b
Total	**7.0 bits/obj**	**9.9 bits/obj**	**8.3 bits/obj**

- 日志结构切片
- 共享EU放置
- 双缓冲区
- 更大的Set

→ 将DRAM开销控制在19%

4.1 测试环境

1.实验环境与建模

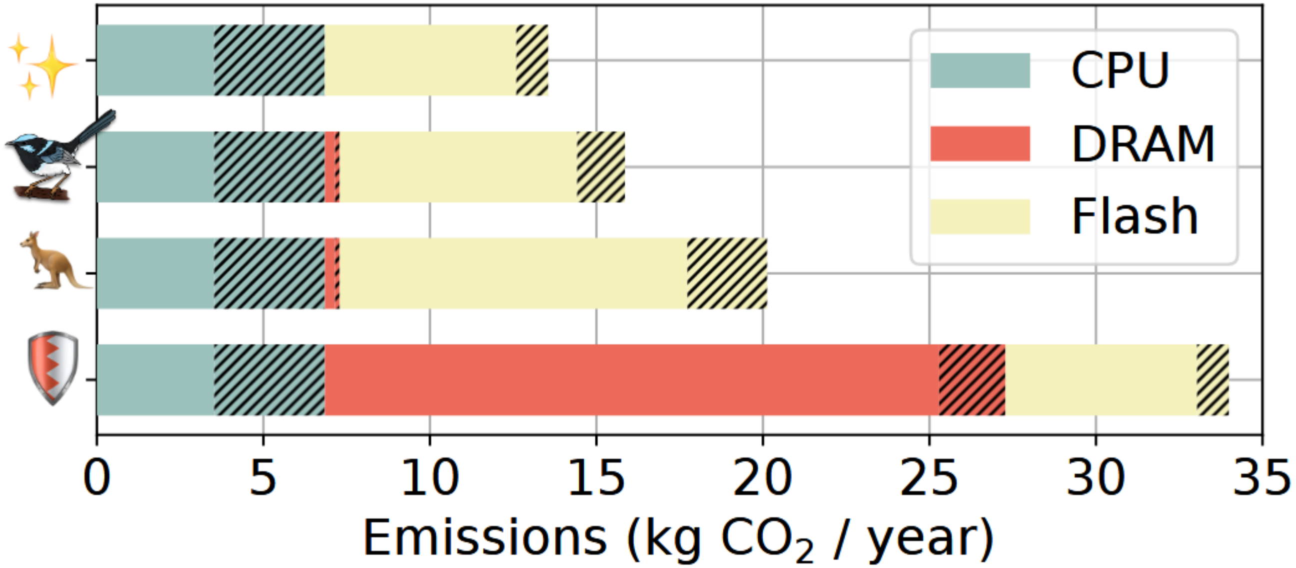
Parameter	FairyWREN	Kangaroo
Interface	WREN (ZNS)	LBAD
Flash capacity	400 GB	400 GB
Usable flash capacity	383 GB	376 GB
LOC size	10% of flash	10% of flash
SOC log size	5% of SOC	5% of SOC
SOC set size	4 KB hot, 4 KB cold	4 KB
Hot-set write frequency	every 5 cold set writes	
Set over-provisioning	5%	

- 基于CacheLib
- 使用模拟和真实的ZNS SSD
- Meta和Twitter的I/O Trace
- 写入寿命和成本依据 Micron SSD建模
- 使用ACT模型（ISCA’22）对CPU、DDR4 DRAM和闪存进行碳排放建模

2.对比对象

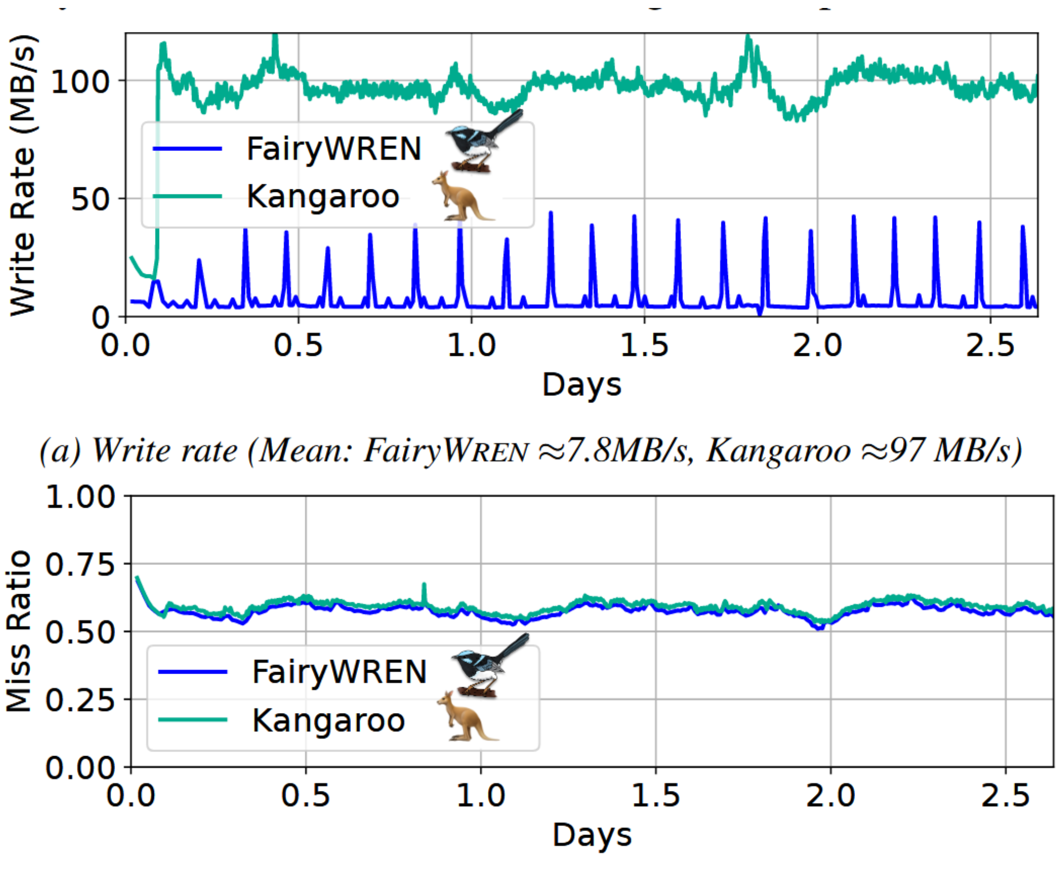
- 理想写入：WA为1且无DRAM开销
- Flashield：日志结构缓存
- Kangaroo（SOTA）
- FairyWREN
- Physical Separation：简单的移植Kangaroo

4.2 总体碳排放



- Flashield 的总体碳排放量比 Kangaroo 高1.7x，由于其高额的DRAM
- FairyWREN 保持了低内存开销，同时大大降低了闪存写入量
- FairyWREN 与 Kangaroo 相比，总体碳排放量减少了21.2%

4.3 闪存与性能测试



(a) Write rate (Mean: FairyWREN ≈7.8MB/s, Kangaroo ≈97 MB/s)

(b) Miss ratio (Mean: FairyWREN ≈ 0.575, Kangaroo ≈ 0.594)

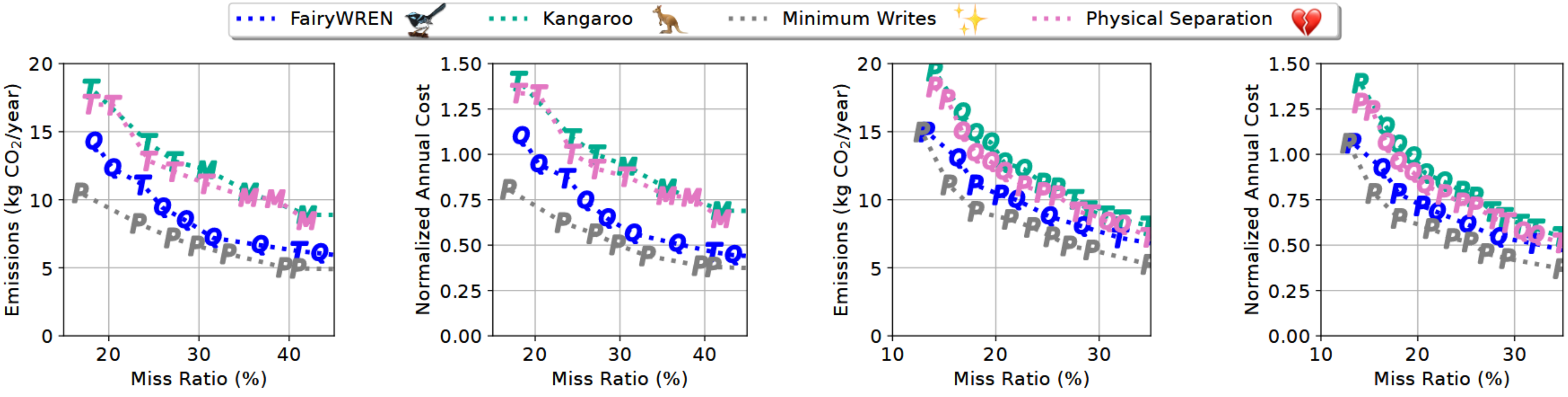
闪存测试

- FairyWREN 的写入速度比 Kangaroo 减少了12.5倍，从 97 MB/s 减少到 7.8 MB/s
- FairyWREN 和 Kangaroo 的Cache率几乎一致

性能测试

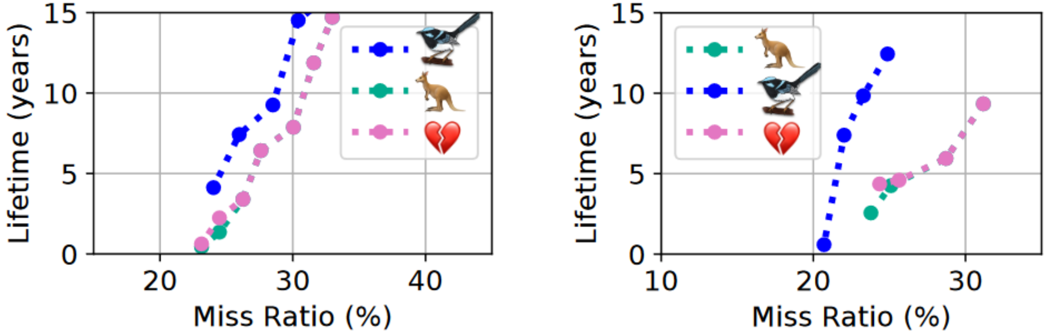
- FairyWREN和Kangaroo的吞吐量分别为104 KOps/s和40.5 KOps/s
- 99% 延迟分别为170μs和1370μs

4.4 缓存命中率测试



(a) Twitter Carbon Emissions (b) Twitter Cost (c) Meta Carbon Emissions (d) Meta Cost

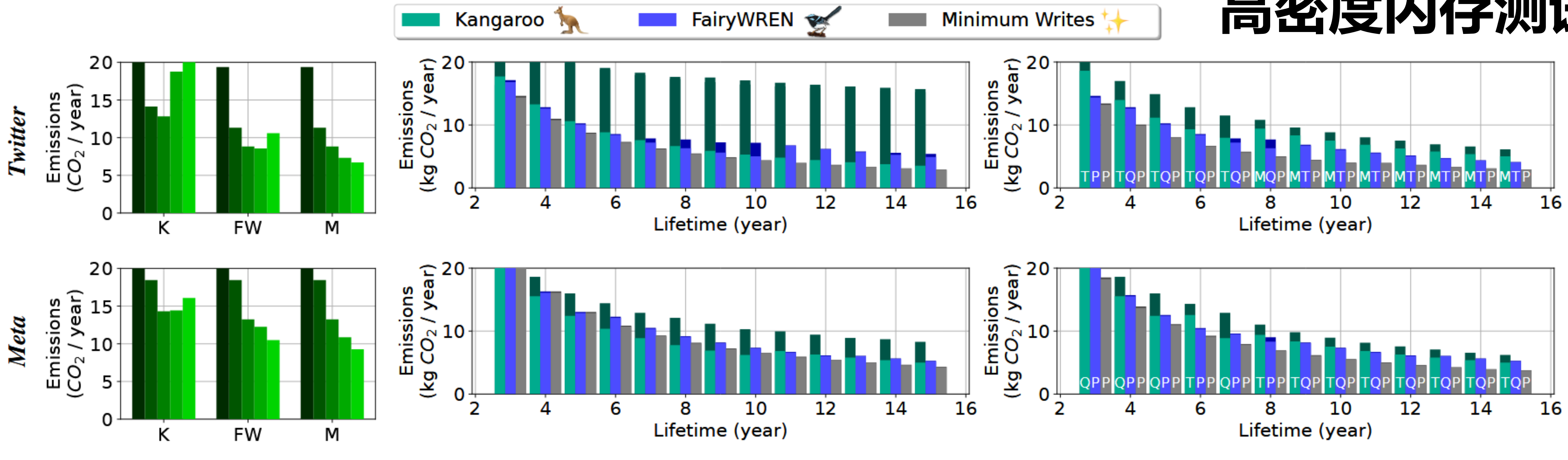
- FairyWREN 和 Kangaroo 相比，相同的Cache Ratio下，**碳排放和成本更低**
- FairyWREN 接近于理想情况
- 直接将 Kangaroo 迁移到WREN接口没有太大优化



(a) Twitter (b) Meta

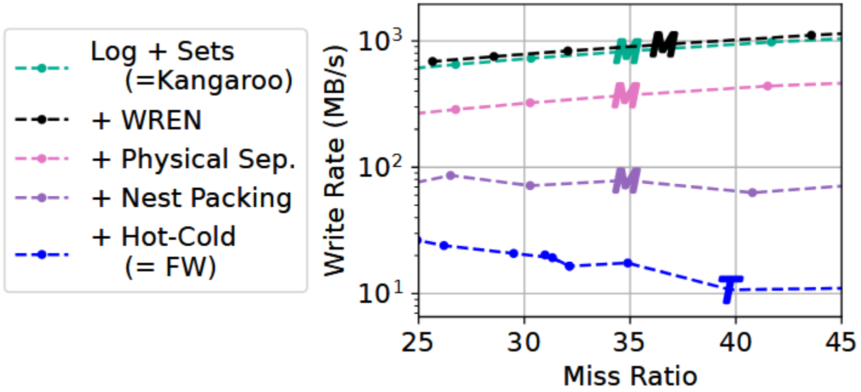
4.5 高密度闪存与分解测试

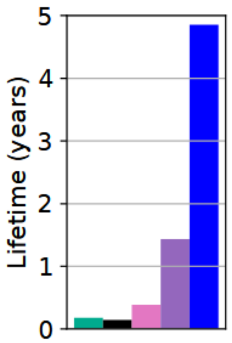
高密度闪存测试



(a) Impact of density (6 years) *(b) Impact of lifetime (on QLC)* *(c) Impact of lifetime (on best flash density)*

- FairyWREN 能够充分利用最新高密度闪存QLC，碳排放更低





分解测试

- 嵌套打包和冷热分离技术均贡献了多倍的优化

5 论文总结

1.背景： 数据中心碳减排和可持续发展的的趋势，而存储器件占碳排放的大部分

	Flash caches should minimize ...			
	Unused flash	DRAM	ALWA	DLWA
Key-value stores	✗	✓	✓	✓
Log-structured caches	✓	✗	✓	✓
Set-associative caches	✗	✓	✗	✗
Kangaroo [67]	✓	✓	✓	✗
FairyWREN	✓	✓	✓	✓

2.问题

- 高密度闪存带来碳减排的契机，但面临**严峻的寿命挑战**
- 传统的LBAD接口会导致**两级写放大**的问题：ALWA和DLWA
- 现有的闪存缓存方案**无法满足可持续使用**的需求

3.本文方案

- 提出**适用于新兴WREN接口**的闪存缓存设计
- 执行**“嵌套打包”算法**，将缓存接纳和垃圾收集统一
- 实现**冷热对象分离**，对WREN接口带来的**DRAM开销优化**

5 论文总结

4.本文优点&启发

Opportunity 1: *Flash is less carbon-intensive than DRAM, so caches are more sustainable with less DRAM.*

DRAM often makes up 40% to 50% of server cost [58, 79, 82] and is no longer scaling (Fig. 2). DRAM also has a large embodied carbon footprint and has large operational emissions due to requiring up to half of system power [38].

Flash is cheaper per-bit, embodies 12× less carbon, and re-

（1）写作

- 总结性的写法，行文很舒畅
- 将高密度闪存的寿命，放到数据中心的碳排放的故事下让人眼前一亮
- 宏大的故事背后是实打实的论述、建模和实验，让人佩服

（2）工作

A Call for Research on Storage Emissions (HotCarbon’24)

- 闪存缓存会带来高频繁的小对象（100B）更新，对工作有一定启发

5.本文不足&疑惑

- 本文的见解很深刻，但设计上还是偏简单的（对OSDI同类工作而言）
- 本文的设计实现写作基于前作Kangaroo耦合展开，自己的设计混杂在其中。虽是出于系统完整性考虑，但哪些是新工作独创的没有明确。

OSDI'24

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

FairyWREN: A Sustainable Cache for Emerging Write-Read-Erase Flash Interfaces

Sara McAllister Yucong "Sherry" Wang Benjamin Berg* Daniel S. Berger†
George Amvrosiadis Nathan Beckmann Gregory R. Ganger

Carnegie Mellon University *UNC Chapel Hill †Microsoft Azure and University of Washington

谢谢阅读与观看