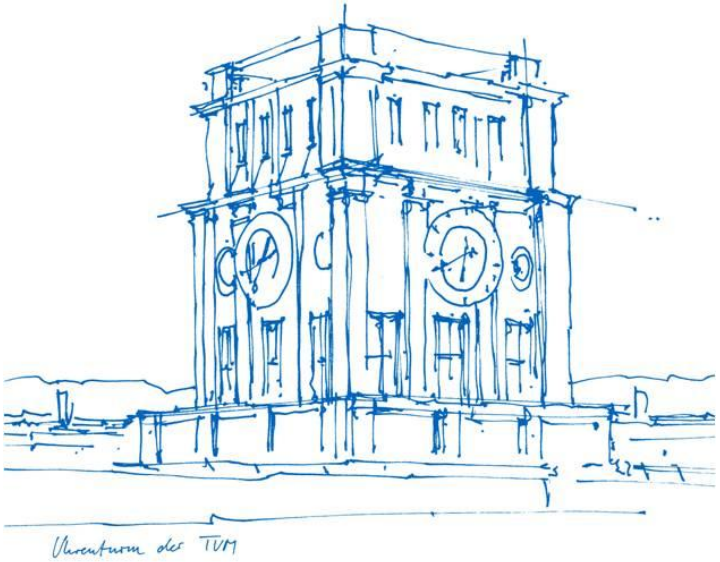
Chair of Computer Architecture and Parallel Systems
TUM School of Computation, Information and Technology
Technical University of Munich

Every Mapping Counts in Large Amounts: Folio Accounting

David Hildenbrand (TUM, Red Hat), Martin Schulz (TUM), Nadav Amit (Technion)



USENIX ATC '24

Santa Clara, July 12, 2024

Introduction

Traditional: **Huge Pages** to reduce TLB misses

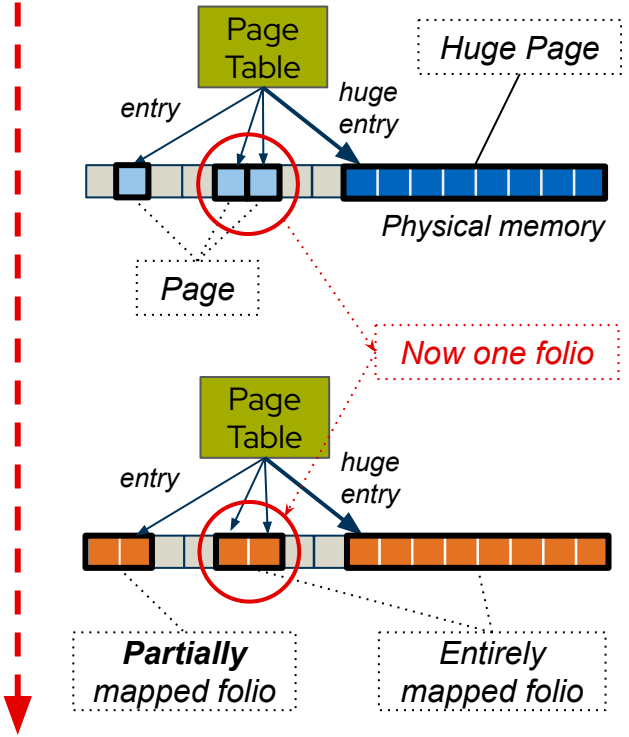
- Utilize **“huge”** page table entries

Trend: Manage larger memory units to improve **OS efficiency**

- Shorter LRU lists, reduced allocation overhead …

Linux’ ***Folio*** *abstraction*: Unit of contiguous pages

- Aggregate state at folio: Reference Counter, Flags, …
- *Might span* ***multiple*** page table entries
- Can be ***partially mapped*** in address spaces



➜ **Challenge: Aggregating per-page mapping state**

Motivation

Per-page state (“map_count”) is **expensive**

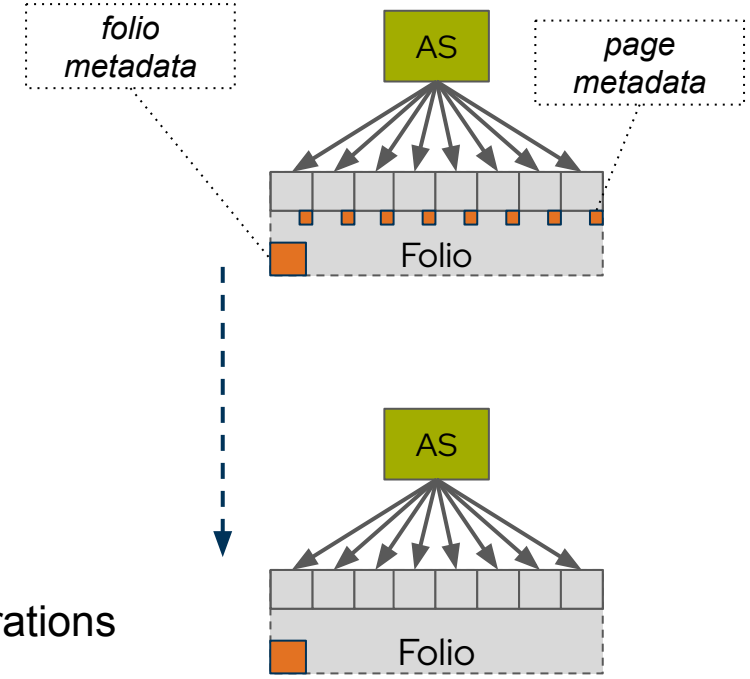
- Memory: maintain page metadata
- Performance: updating page metadata

Per-page state used to determine **page exclusivity**

- **Single vs. multiple address spaces**

Determining exclusivity is crucial

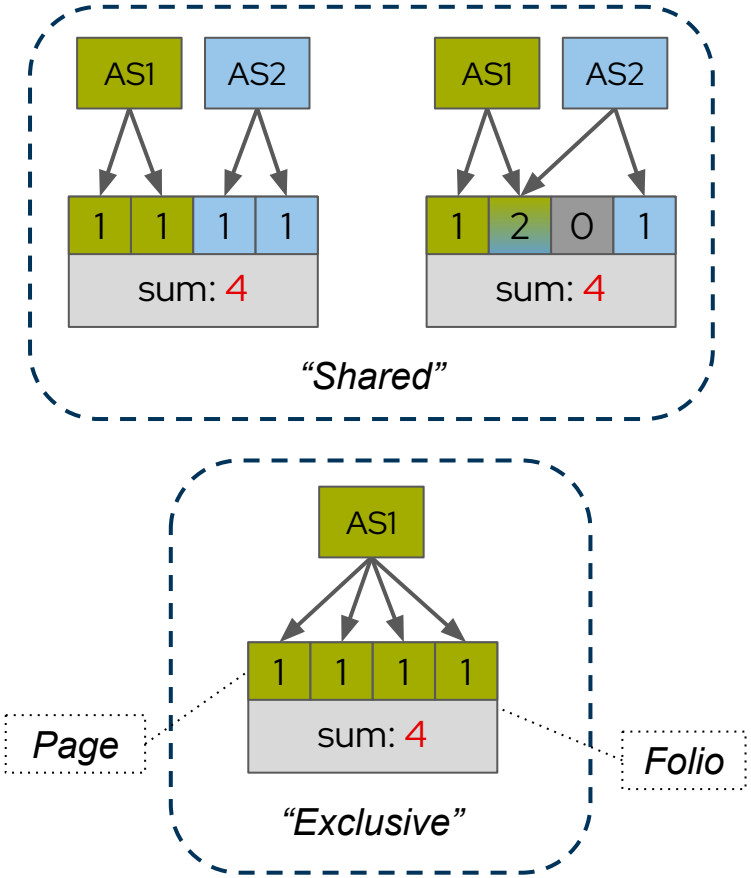
- Correctness: Enforce OS policies, memory statistics
- Performance: Reduce redundant Copy-on-Write operations



& “exclusive vs. shared” ?

➜ **New folio accounting approach required**

Challenge: Page map_count



Dual purpose

1. #Page table entries
2. #Address Spaces (ASes)[1]

Exclusivity

- Exclusive: *page.map_count == 1*
- Shared: *page.map_count > 1*

Aggregating map_counts at folio level

➜ **Insufficient to determine folio exclusivity**

[1] *anonymous pages can only be mapped once per AS*

Approach (1): Pigeonhole Principle

Possible solution: Track #ASes that map a folio

- Requires tracking #mappings per AS per folio
- Linux allows for up to 4M processes

Simplification: **“one vs. multiple”** ASes

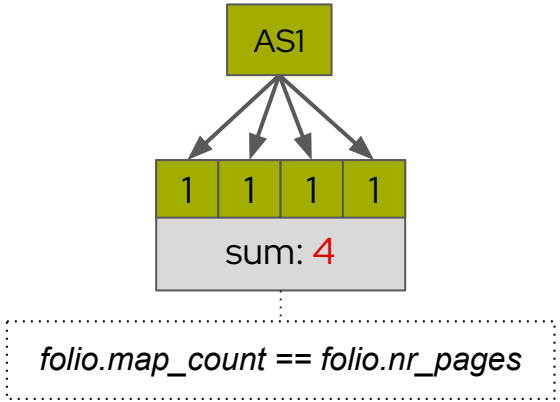
- Sufficient for “exclusive vs. shared”

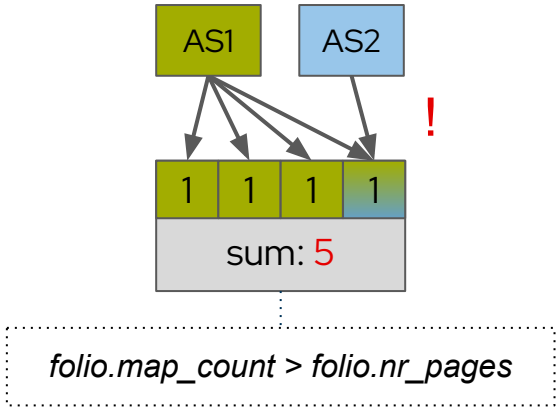
Pigeonhole Principle: *folio.map_count > folio.nr_pages*

- *At least one page mapped by another AS*
  ➜ ***“Shared”***

folio.map_count <= folio.nr_pages:

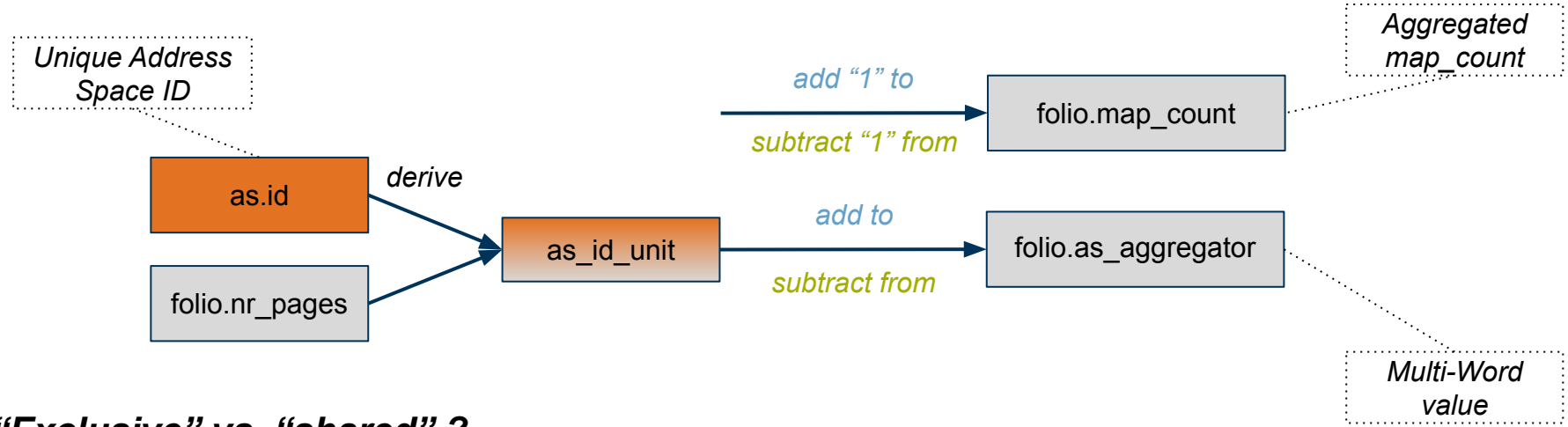
➜ More information required





Approach (2): Specialized Tracking

Adding / **Removing** one mapping (page table entry)



"Exclusive" vs. "shared" ?

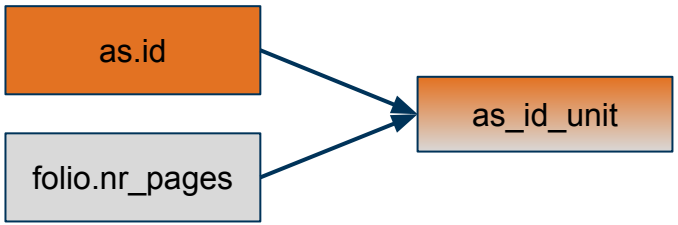
- *folio.as_aggregator == folio.map_count* * as_id_unit

➜ Suitable **as_id_unit** required

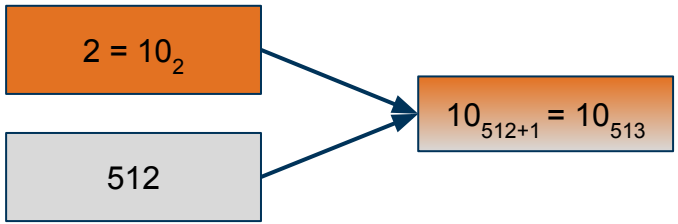
Approach (3): Deriving as_id_unit

Interpret **bit representation** of **as.id** in basis "***folio.nr_pages + 1***"

- *Intuition*: One counter per ***as.id*** bit & compress counters into single value
- No overflow while *folio.map_count* <= *folio.nr_pages*



Example:

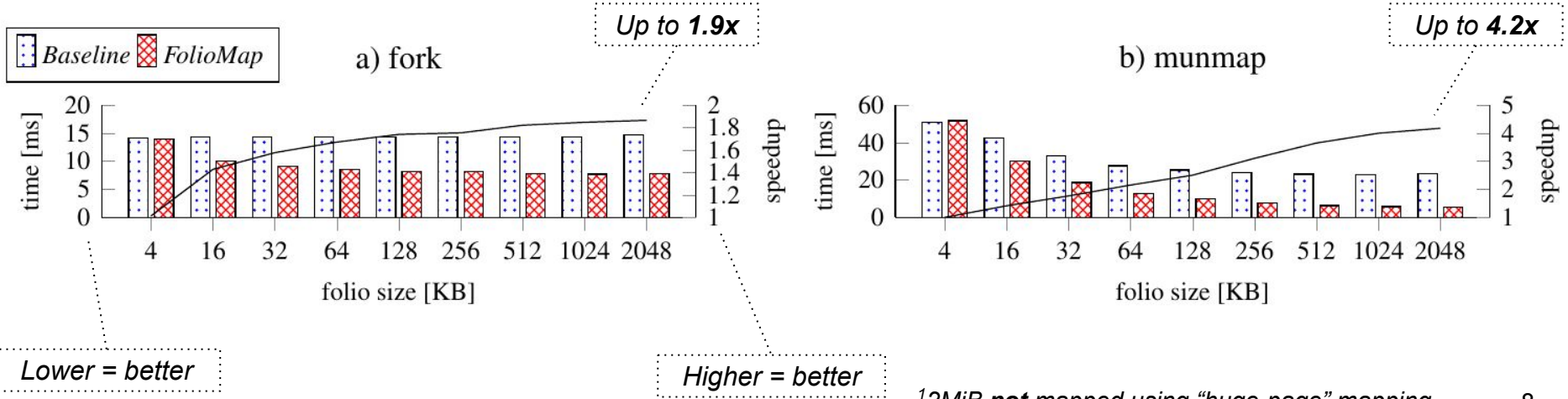


Evaluation (1): OS Primitives

Comparison of our approach (FolioMap) with current approach in Linux (BaseLine)

- BaseLine: Linux 6.7 + **mTHP (multi-sized THP) patches**
- FolioMap: Extension of BaseLine with our changes

Benchmarks allocate 1 GB of folios **of a given size**[1] to then **fork()** or **munmap()**

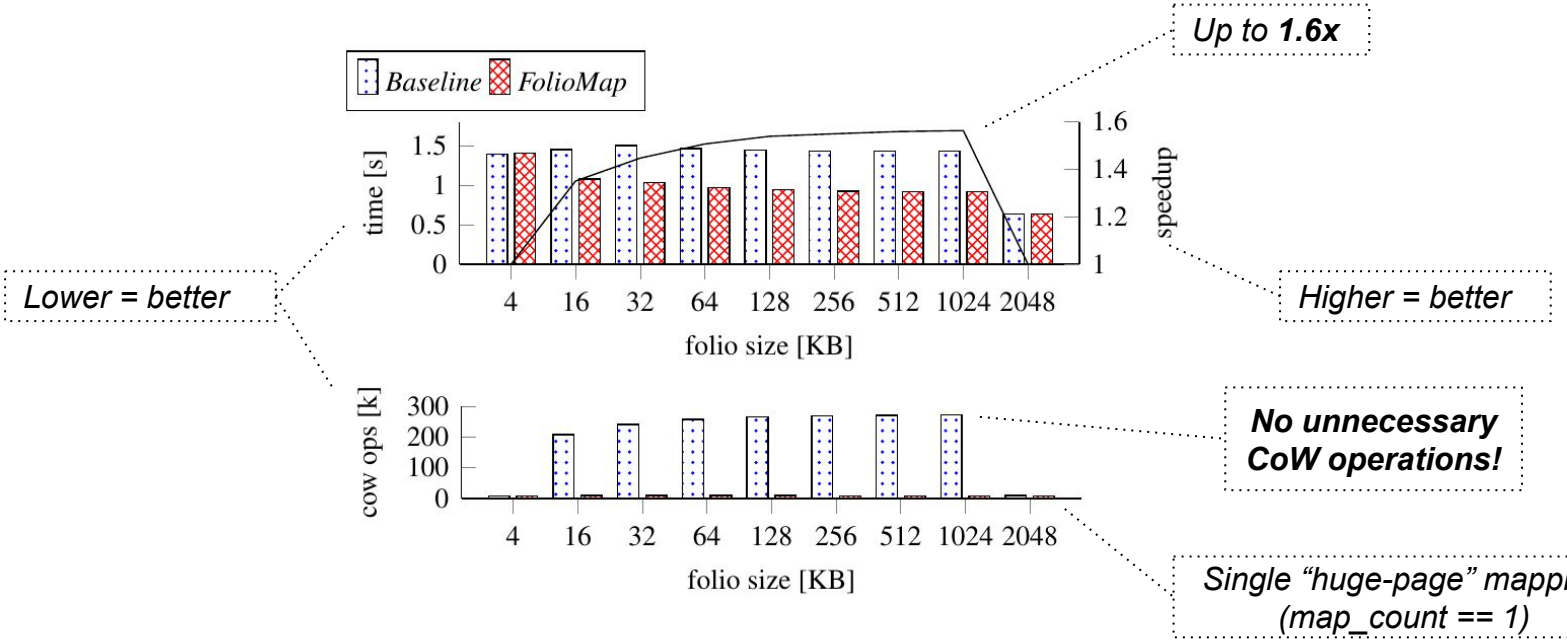


[1] 2MiB **not** mapped using "huge-page" mapping

Evaluation (2): Copy-on-Write impact on Python

Multi-process Python program: Possible **Copy-on-Write *optimization (reuse)***

- BaseLine: Only when folio is mapped with **single page table entry**
- FolioMap: For **all folio sizes**



Conclusions

Linux introduced folio abstraction

- Page table mappings still tracked per page
- Hinders performance and memory savings

Innovative approach for per-folio accounting

- Pigeonhole principle + specialized tracking

Precise and scalable

- Overheads grow sublinearly with folio size
- Significant speedups

Paves the way for more enhanced system performance

- Implementation is getting incrementally upstreamed

David

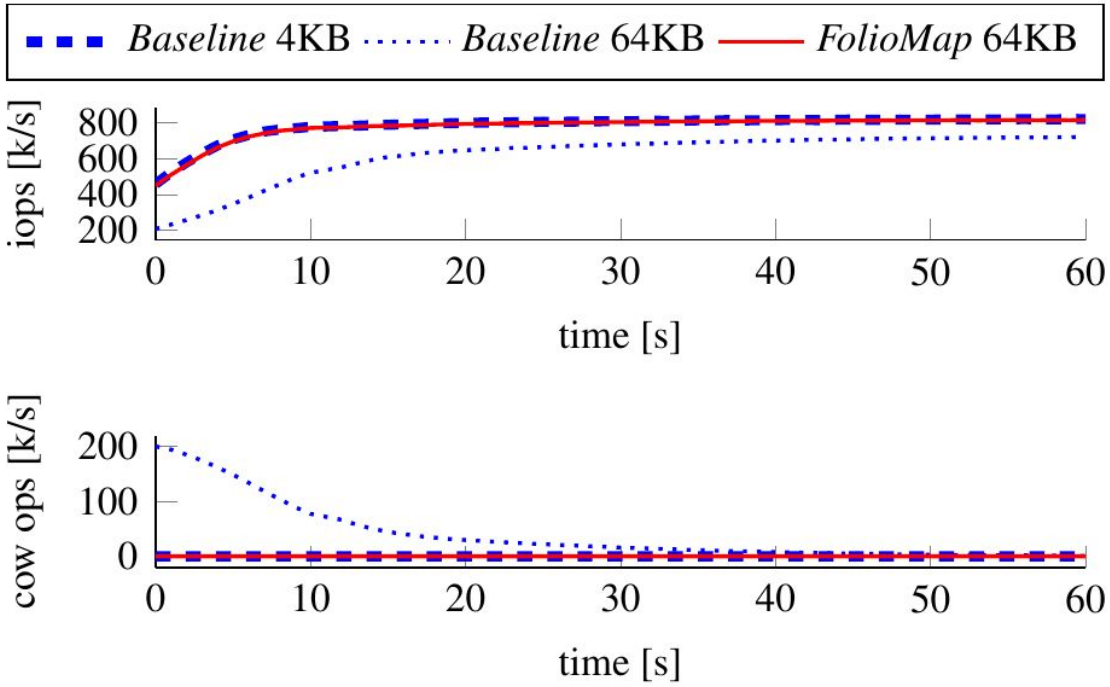
https://www.ce.cit.tum.de/en/caps/staff/david-hildenbrand/

For more details/experiments/results, refer to our paper!

Backup

1. Redis
2. Python Program
3. Pseudocode
4. Example

Backup (1): Redis



Backup (2): Python Program

```
import multiprocessing as mp
import numpy

size = pow(512, 3)
arr = numpy.ones(size)

def fn(range):
     return numpy.sum(arr[range[0]:range[1]])

def multi_process_sum(arr):
     c = int(size / 8)
     ranges = [(i,i + c) for i in range(0, size, c)]

     pool = mp.Pool(processes = 8)
     return int(sum(pool.map(fn, ranges)))

assert(multi_process_sum(arr) == size)
arr[0:size] = 0
assert(multi_process_sum(arr) == 0)
```

Backup (3): Pseudocode

```
def as_id_unit(as, folio):
     binary_unit = bin(as.id)[2:]
     return int(binary_unit, base=folio.nr_pages + 1)

def add_folio_mapping(as, folio):
     folio.map_count += 1
     folio.as_aggregator += as_id_unit(as, folio)

def remove_folio_mapping(as, folio):
     folio.map_count -= 1
     folio.as_aggregator -= as_id_unit(as, folio)

def is_folio_mapped_exclusively(as, folio):
     if folio.map_count > folio.nr_pages:
     return false
     return (as_id_unit(as, folio) *
          folio.map_count == folio.as_aggregator)
```

Backup (4): Example