



CBMM: Financial Advice for Kernel Memory Managers

Mark Mansi, Bijan Tabatabai, Michael M. Swift



SCAIL



Kernel Memory Management

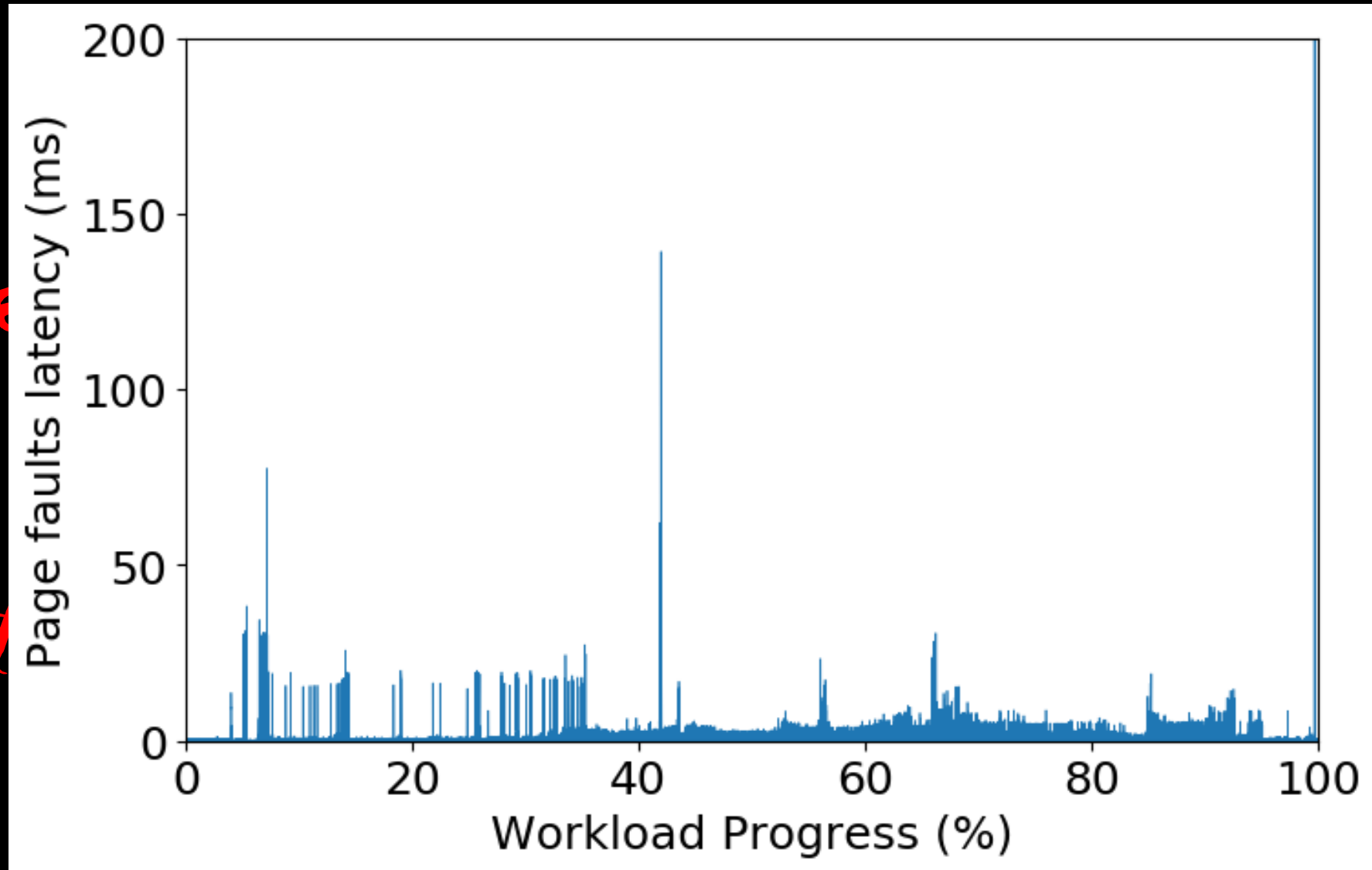
Kernel Memory Management



Kernel Memory Management



Kernel Memory Management



page

tion

ges

Kernel Memory Management



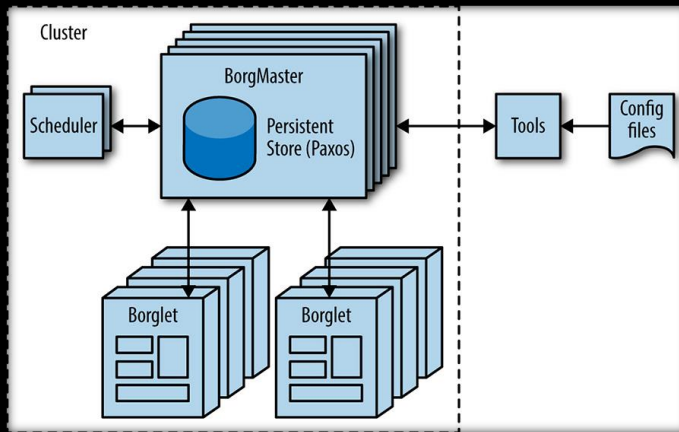
page

tion

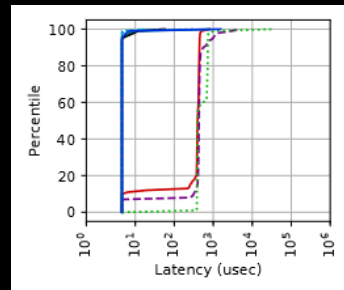
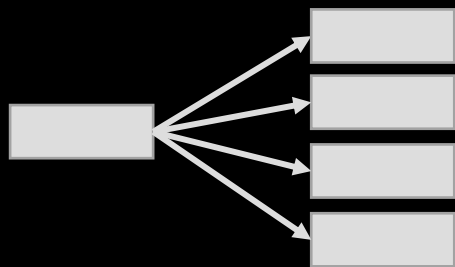
ges

Modern 1st-party Datacenter Workloads

Modern 1st-party Datacenter Workloads

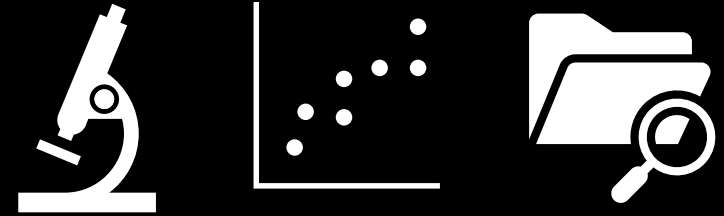
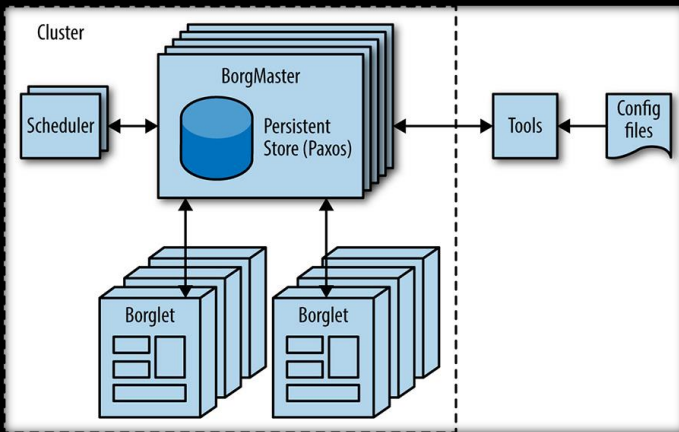


Diverse, concurrent workloads



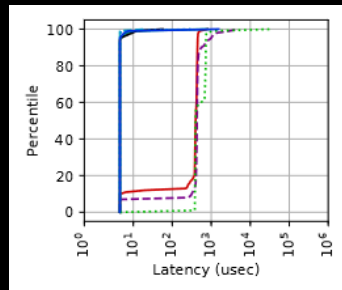
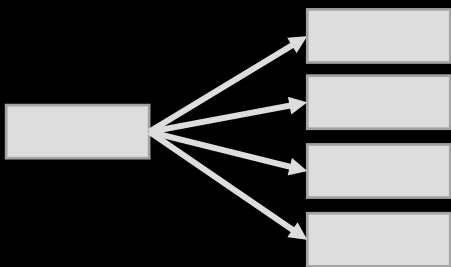
Tail Latency Requirements

Modern 1st-party Datacenter Workloads



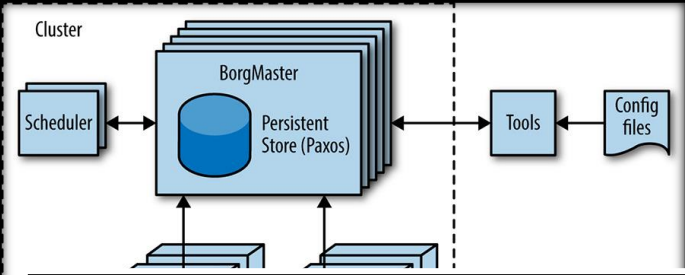
Diverse, concurrent workloads

Highly controlled, replicated



Tail Latency Requirements

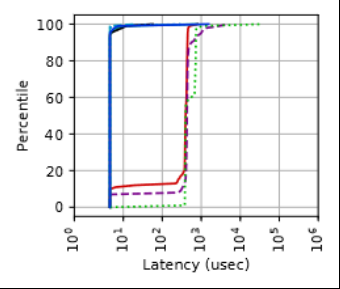
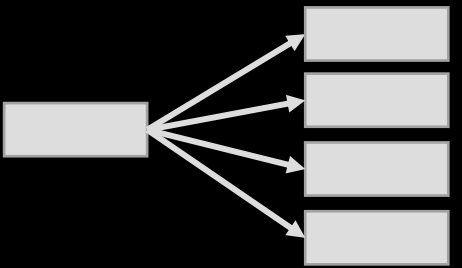
Modern 1st-party Datacenter Workloads



Current MM + 1st-party DC workloads = ???

Diverse, concurrent workloads

Highly controlled, replicated

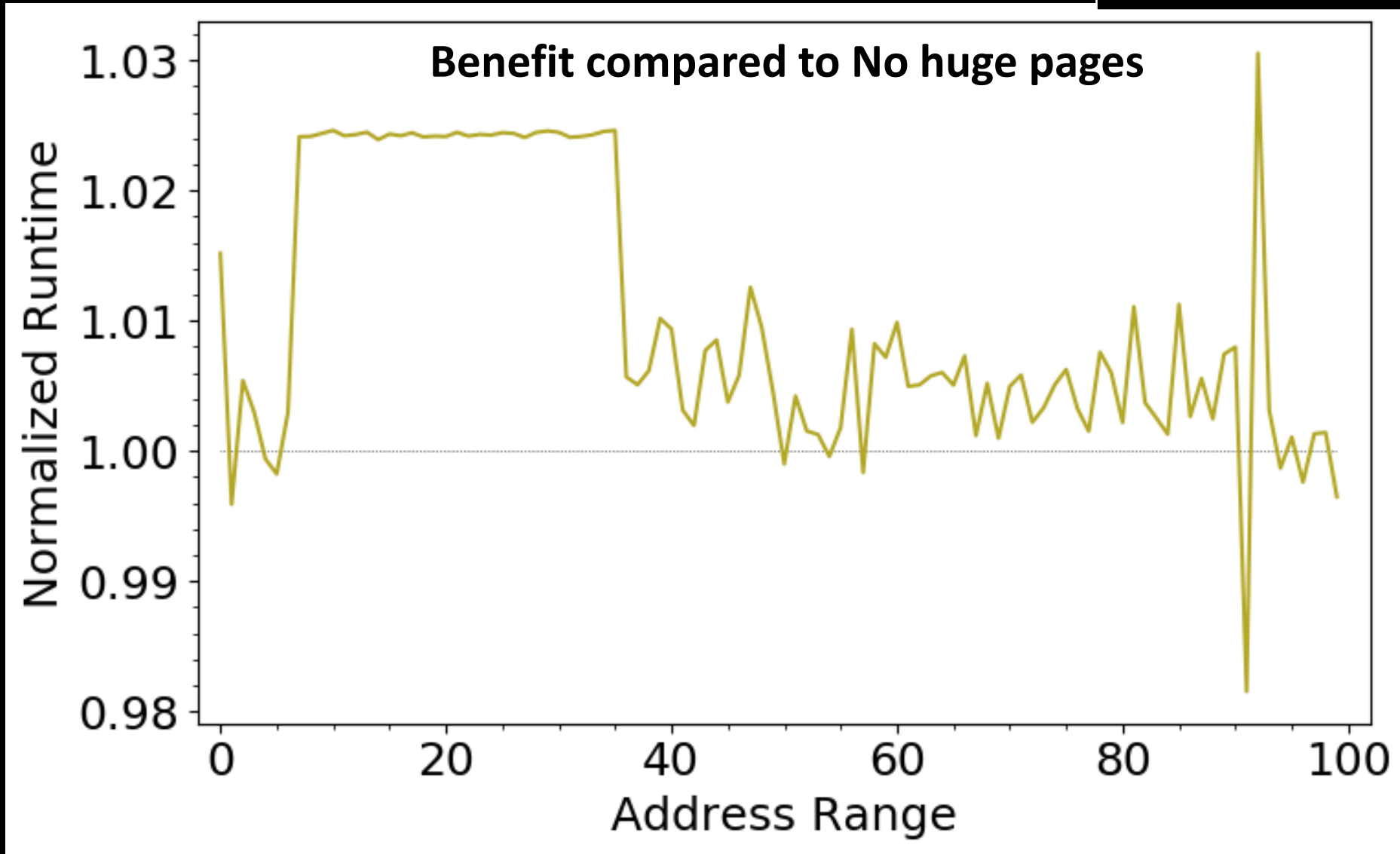


Tail Latency Requirements

1. Low-Quality Information

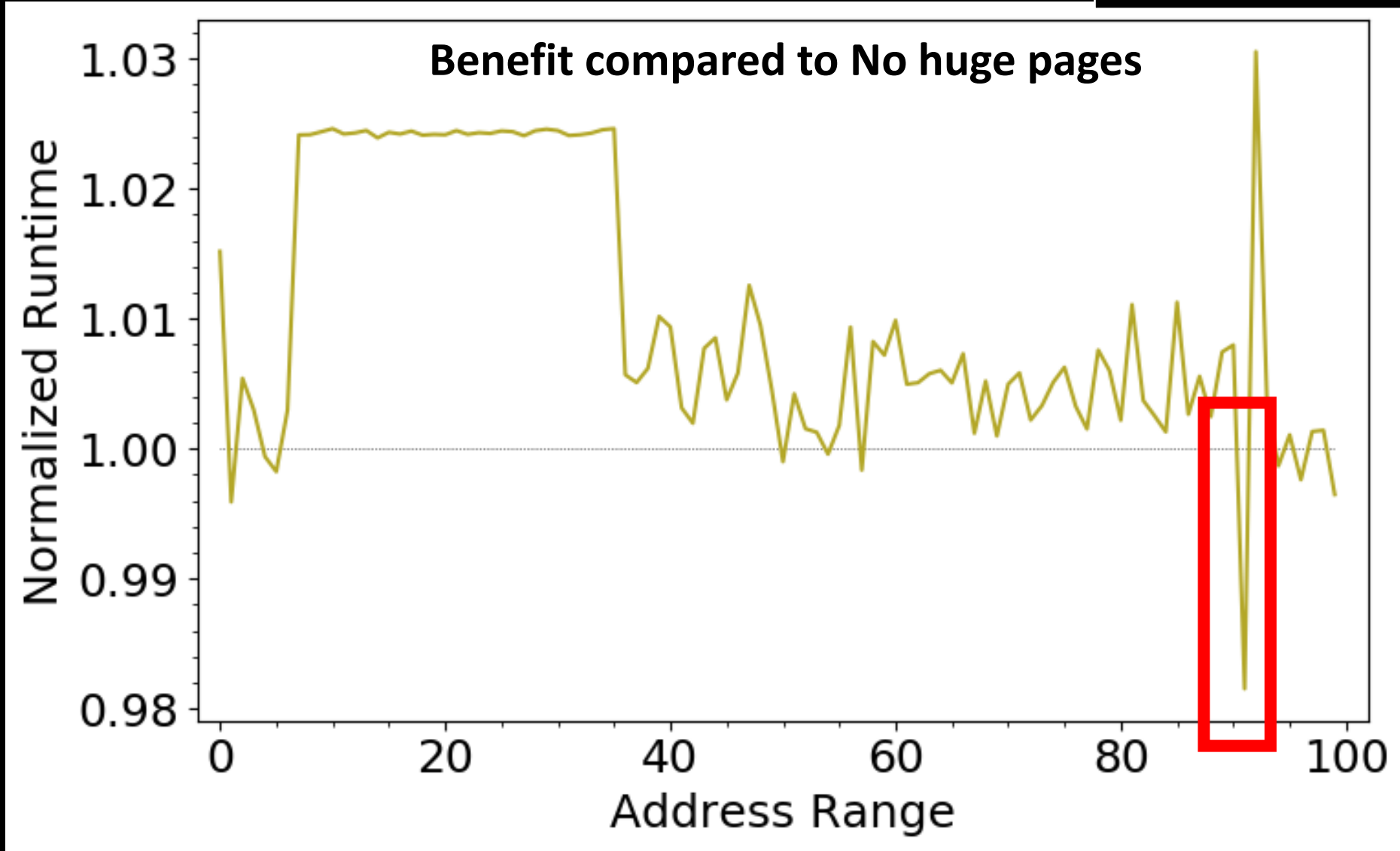
SPEC 2017 xz
Scaled up to 192GB
Data compression

1. Low-Quality Information



1. Low-Quality Information

SPEC 2017 xz
Scaled up to 192GB
Data compression



1. Low-Quality Information

1. Low-Quality Information

```
ls /proc/sys/vm/  
admin_reserve_kbytes      dirty_bytes                hugetlb_...  
block_dump                dirty_expire_centisecs    laptop_m...  
compact_memory            dirty_ratio                legacy_v...  
compact_unevictable_allowed dirty_writeback_centisecs lowmem_...  
compaction_proactiveness dirtytime_expire_seconds  max_map...  
dirty_background_bytes    drop_caches                min_free...  
dirty_background_ratio    extfrag_threshold         mmap_min...
```

System-wide tunables & stats

1. Low-Quality Information

```
ls /proc/sys/vm/
admin_reserve_kbytes      dirty_bytes              hugetlb_
block_dump                dirty_expire_centisecs  laptop_m
compact_memory            dirty_ratio              legacy_
compact_unevictable_allowed dirty_writeback_centisecs lowmem_
compaction_proactiveness  dirtytime_expire_seconds max_map_
dirty_background_bytes    drop_caches              min_free
dirty_background_ratio    extfrag_threshold        mmap_min
```

System-wide tunables & stats

Session: VM/Memory

ASPLOS'19, April 13–17, 2019, Providence, RI, USA

HawkEye: Efficient Fine-grained OS Support for Huge Pages

Ashish Panwar
Indian Institute of Science
ashishpanwar@iisc.ac.in

Sorav Bansal
Indian Institute of Technology Delhi
sbansal@iitd.ac.in

K. Gopinath
Indian Institute of Science
gopi@iisc.ac.in

Abstract

Effective huge page management in operating systems is necessary for mitigation of address translation overheads. However, this continues to remain a difficult area in OS de-

into focus [32, 49, 59, 61, 63]. Modern architectures implementing large multi-level TLBs and page-walk caches, all supporting multiple page sizes [35, 40], require careful OS design to determine suitable page sizes for different workloads

Hardware Performance Counters

1. Low-Quality Information

```
ls /proc/sys/vm/
admin_reserve_kbytes      dirty_bytes               hugetlb_
block_dump                dirty_expire_centisecs   laptop_m
compact_memory           dirty_ratio               legacy_
compact_unevictable_allowed dirty_writeback_centisecs lowmem_
compaction_proactiveness dirtytime_expire_seconds max_map
dirty_background_bytes   drop_caches              min_free
dirty_background_ratio   extfrag_threshold       mmap_mir
```

WRONG
GRANULARITY

HawkEye: Efficient Fine-grained OS Support for Huge Pages

Ashish Panwar
Indian Institute of Science
ashishpanwar@iisc.ac.in

Sorav Bansal
Indian Institute of Technology Delhi
sbansal@iitd.ac.in

K. Gopinath
Indian Institute of Science
gopi@iisc.ac.in

Abstract

Effective huge page management in operating systems is necessary for mitigation of address translation overheads. However, this continues to remain a difficult area in OS de-

into focus [32, 49, 59, 61, 63]. Modern architectures implementing large multi-level TLBs and page-walk caches, all supporting multiple page sizes [35, 40], require careful OS design to determine suitable page sizes for different workloads

Hardware Performance Counters

1. Low-Quality Information

```
ls /proc/sys/vm/
admin_reserve_kbytes      dirty_bytes                hugetlb_
block_dump                dirty_expire_centisecs    laptop_
compact_memory           dirty_ratio                legacy_
compact_unevictable_allowed dirty_writeback_centisecs lowmem_
compaction_proactiveness dirtytime_expire_seconds max_map_
dirty_background_bytes   drop_caches                min_free
dirty_background_ratio   extfrag_threshold         mmap_mi
```

```
#include <sys/mman.h>
```

```
int madvise(void *addr, size_t length, int advice);
```

madvise

WRONG
GRANULARITY

HawkEye: Efficient Fine-grained OS Support for Huge Pages

Ashish Panwar
Indian Institute of Science
ashishpanwar@iisc.ac.in

Sorav Bansal
Indian Institute of Technology Delhi
sbansal@iitd.ac.in

K. Gopinath
Indian Institute of Science
gopi@iisc.ac.in

Abstract

Effective huge page management in operating systems is necessary for mitigation of address translation overheads. However, this continues to remain a difficult area in OS de-

into focus [32, 49, 59, 61, 63]. Modern architectures implementing large multi-level TLBs and page-walk caches, all supporting multiple page sizes [35, 40], require careful OS design to determine suitable page sizes for different workloads

Hardware Performance Counters

1. Low-Quality Information

```
ls /proc/sys/vm/  
admin_reserve_kbytes      dirty_bytes                hugetlb_...  
block_dump                dirty_expire_centisecs    laptop_...  
compact_memory           dirty_ratio                legacy_...  
compact_unevictable_allowed dirty_writeback_centisecs lowmem_...  
compaction_proactiveness dirtytime_expire_seconds max_map_...  
dirty_background_bytes   drop_caches               min_free...  
dirty_background_ratio   extfrag_threshold        mmap_mir...
```

```
#inclu
```

HARD TO USE WELL

```
int ma... advice);
```

madvise

**WRONG
GRANULARITY**

HawkEye: Efficient Fine-grained OS Support for Huge Pages

Ashish Panwar
Indian Institute of Science
ashishpanwar@iisc.ac.in

Sorav Bansal
Indian Institute of Technology Delhi
sbansal@iitd.ac.in

K. Gopinath
Indian Institute of Science
gopi@iisc.ac.in

Abstract

Effective huge page management in operating systems is necessary for mitigation of address translation overheads. However, this continues to remain a difficult area in OS de-

into focus [32, 49, 59, 61, 63]. Modern architectures implementing large multi-level TLBs and page-walk caches, all supporting multiple page sizes [35, 40], require careful OS design to determine suitable page sizes for different workloads

Hardware Performance Counters

1. Low-Quality Information

```

4 ls /proc/sys/vm/
admin_reserve_kbytes      dirty_bytes              hugetlb_
block_dump                dirty_expire_centisecs  laptop_
compact_memory            dirty_ratio              legacy_
compact_unevictable_allowed dirty_writeback_centisecs lowmem_
compaction_proactiveness dirtytime_expire_seconds max_map_
dirty_background_bytes    drop_caches              min_free
dirty_background_ratio    extfrag_threshold        mmap_mi
    
```

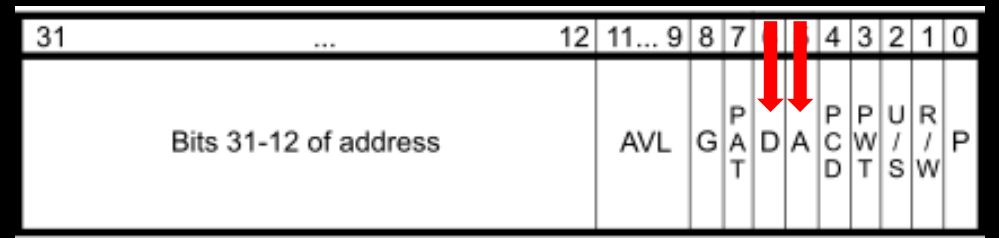
```

#include <sys/mman.h>
int main(int argc, char** argv, struct timeval* tv, void* advice);
    
```

HARD TO USE WELL

madvise

WRONG GRANULARITY



Page table A/D bits

HawkEye: Efficient Fine-grained OS Support for Huge Pages

Ashish Panwar
Indian Institute of Science
ashishpanwar@iisc.ac.in

Sorav Bansal
Indian Institute of Technology Delhi
sbansal@iitd.ac.in

K. Gopinath
Indian Institute of Science
gopi@iisc.ac.in

Abstract

Effective huge page management in operating systems is necessary for mitigation of address translation overheads. However, this continues to remain a difficult area in OS de-

sign into focus [32, 49, 59, 61, 63]. Modern architectures implementing large multi-level TLBs and page-walk caches, all supporting multiple page sizes [35, 40], require careful OS design to determine suitable page sizes for different workloads

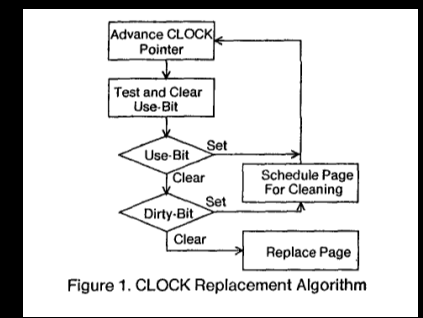
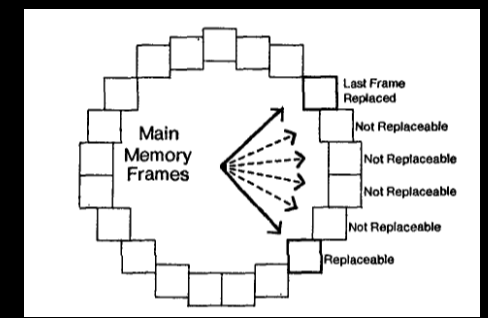


Figure 1. CLOCK Replacement Algorithm

Hardware Performance Counters

Page fault freq/location

1. Low-Quality Information

```

4 ls /proc/sys/vm/
admin_reserve_kbytes      dirty_bytes              hugetlb_
block_dump                dirty_expire_centisecs  laptop_
compact_memory            dirty_ratio              legacy_
compact_unevictable_allowed dirty_writeback_centisecs lowmem_
compaction_proactiveness dirtytime_expire_seconds max_map_
dirty_background_bytes    drop_caches              min_free
dirty_background_ratio    extfrag_threshold        mmap_mi
    
```

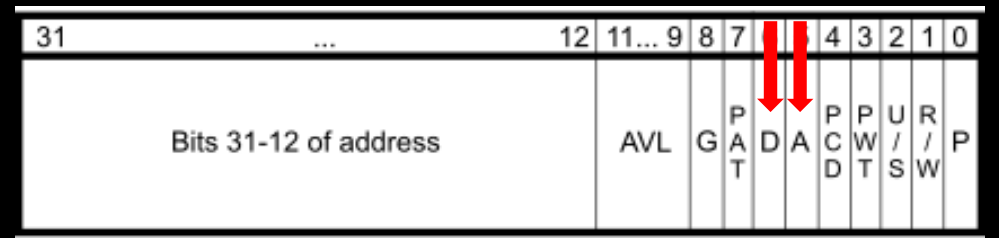
```

#include <sys/mman.h>
int main(int argc, char** argv, struct timeval* tv, void* advice);
    
```

HARD TO USE WELL

madvise

WRONG GRANULARITY



Page table A/D bits
EXPENSIVE

HawkEye: Efficient Fine-grained OS Support for Huge Pages

Ashish Panwar
Indian Institute of Science
ashishpanwar@iisc.ac.in

Sorav Bansal
Indian Institute of Technology Delhi
sbansal@iitd.ac.in

K. Gopinath
Indian Institute of Science
gopi@iisc.ac.in

Abstract

Effective huge page management in operating systems is necessary for mitigation of address translation overheads. However, this continues to remain a difficult area in OS de-

sign into focus [32, 49, 59, 61, 63]. Modern architectures implementing large multi-level TLBs and page-walk caches, all supporting multiple page sizes [35, 40], require careful OS design to determine suitable page sizes for different workloads

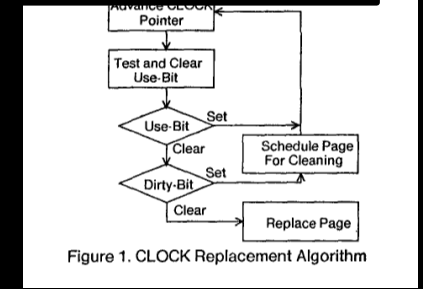
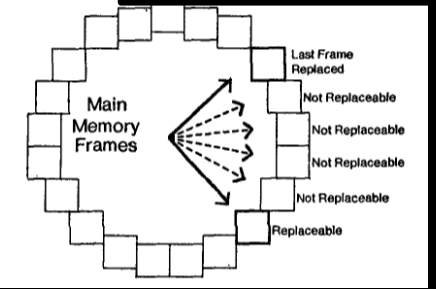


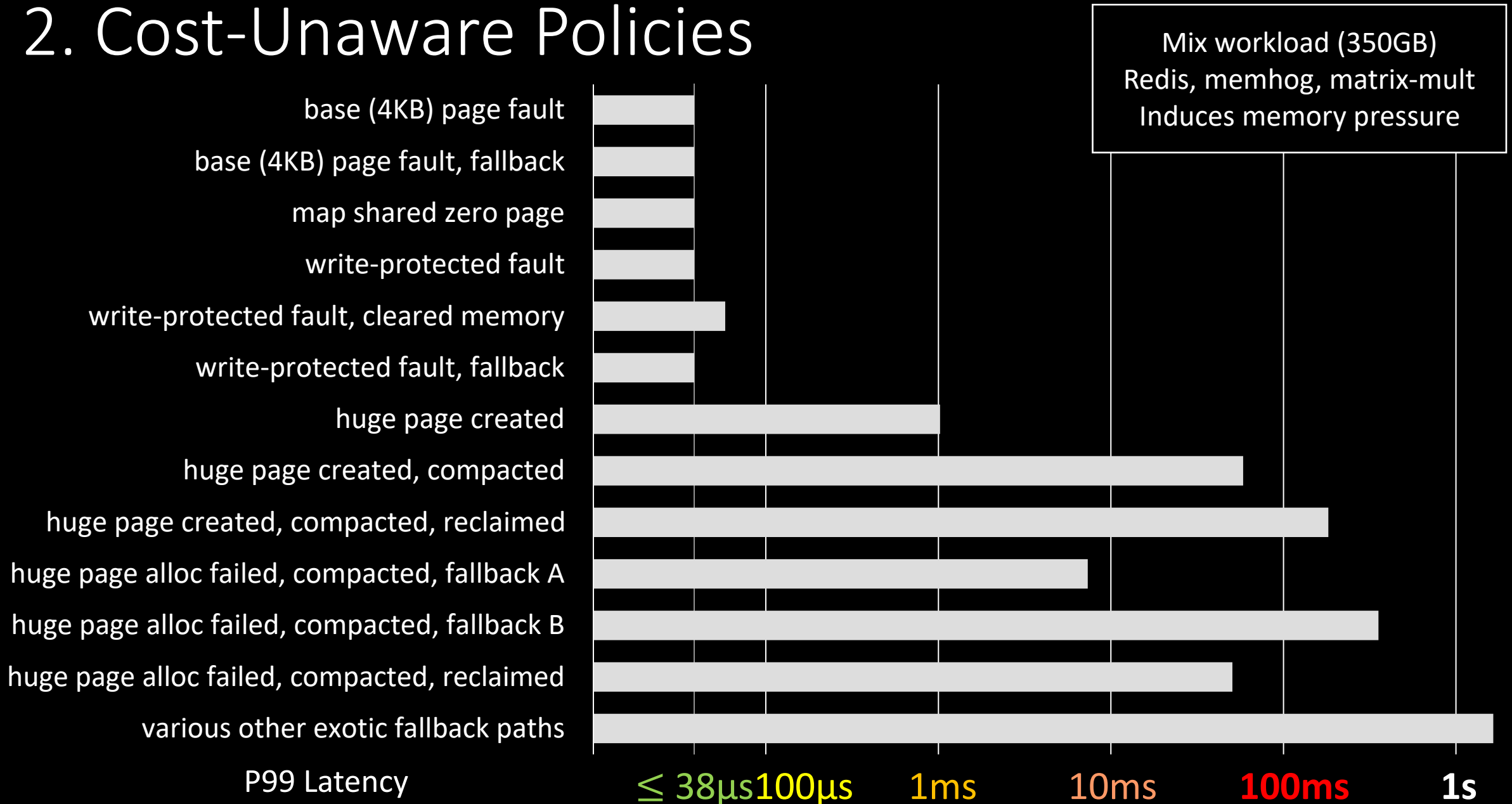
Figure 1. CLOCK Replacement Algorithm

Hardware Performance Counters

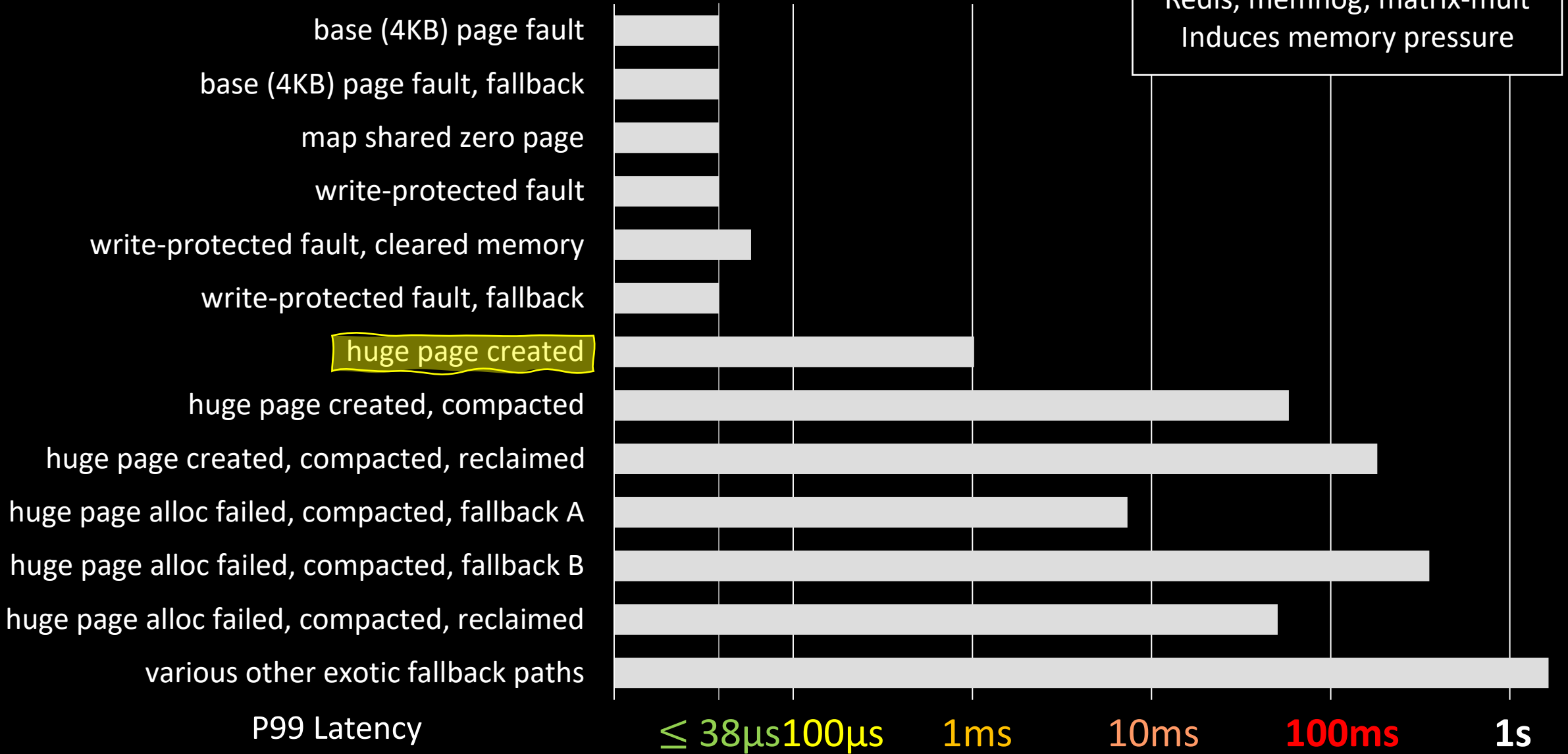
Page fault freq/location

2. Cost-Unaware Policies

2. Cost-Unaware Policies

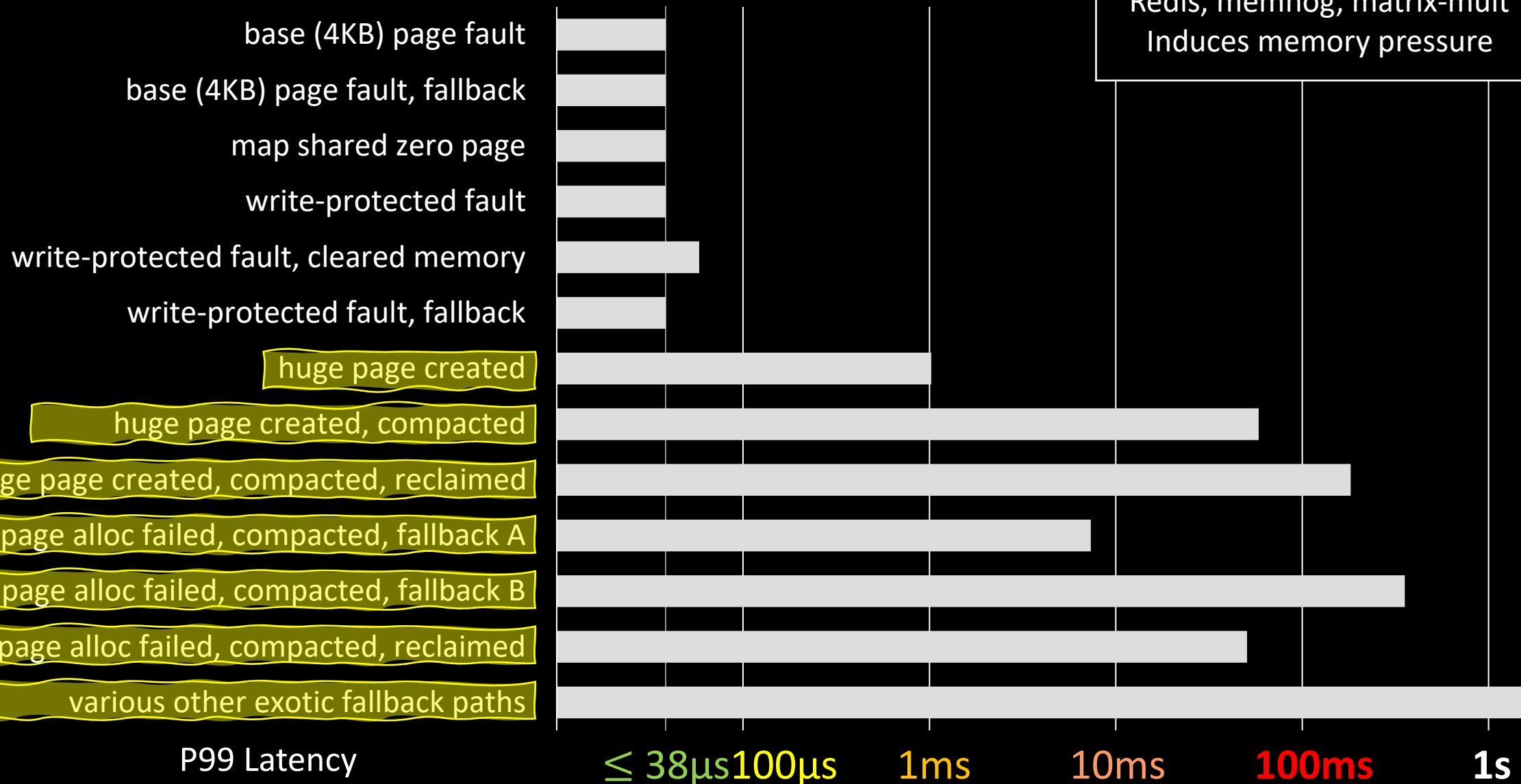


2. Cost-Unaware Policies

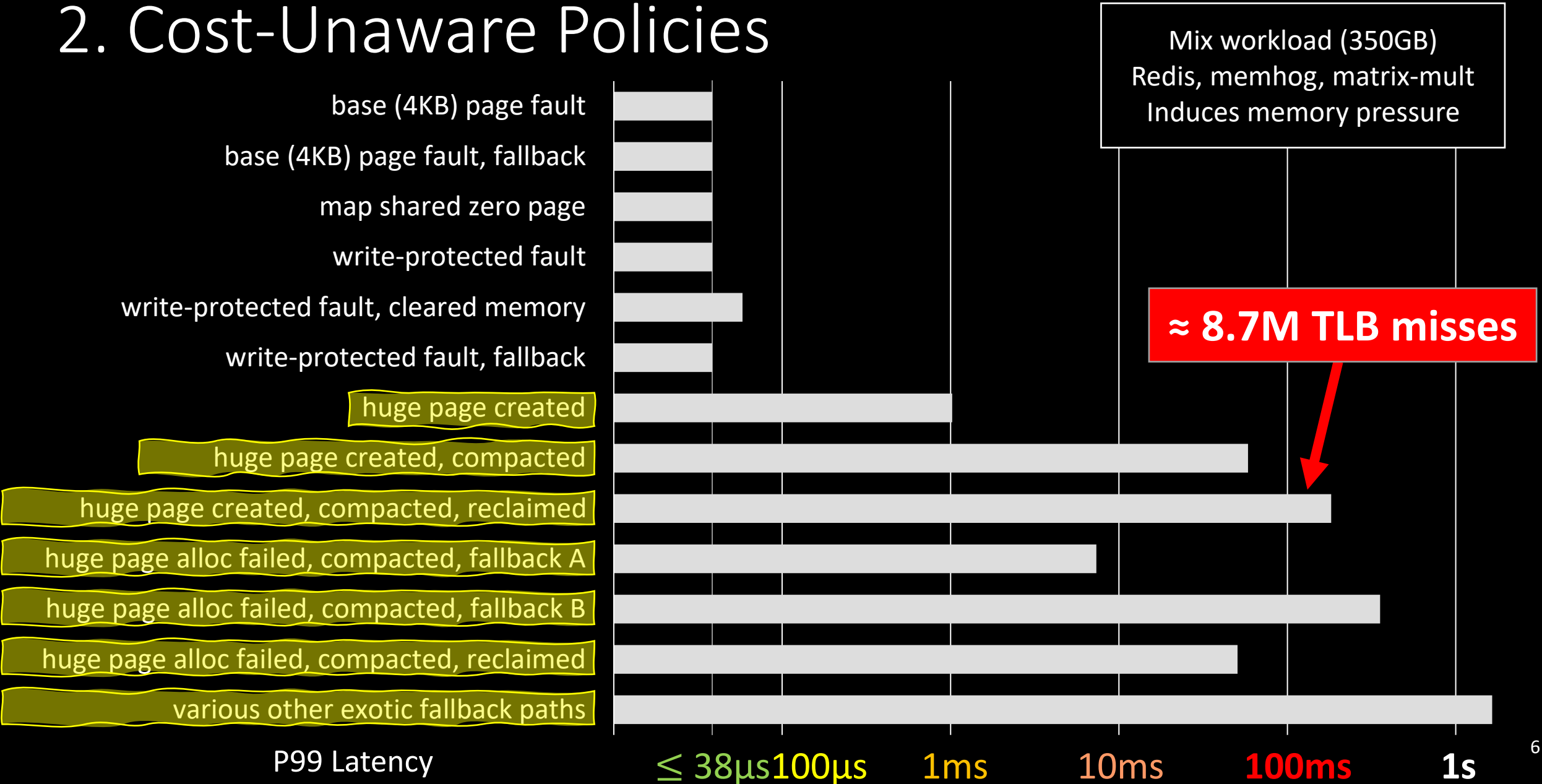


2. Cost-Unaware Policies

Mix workload (350GB)
Redis, memhog, matrix-mult
Induces memory pressure

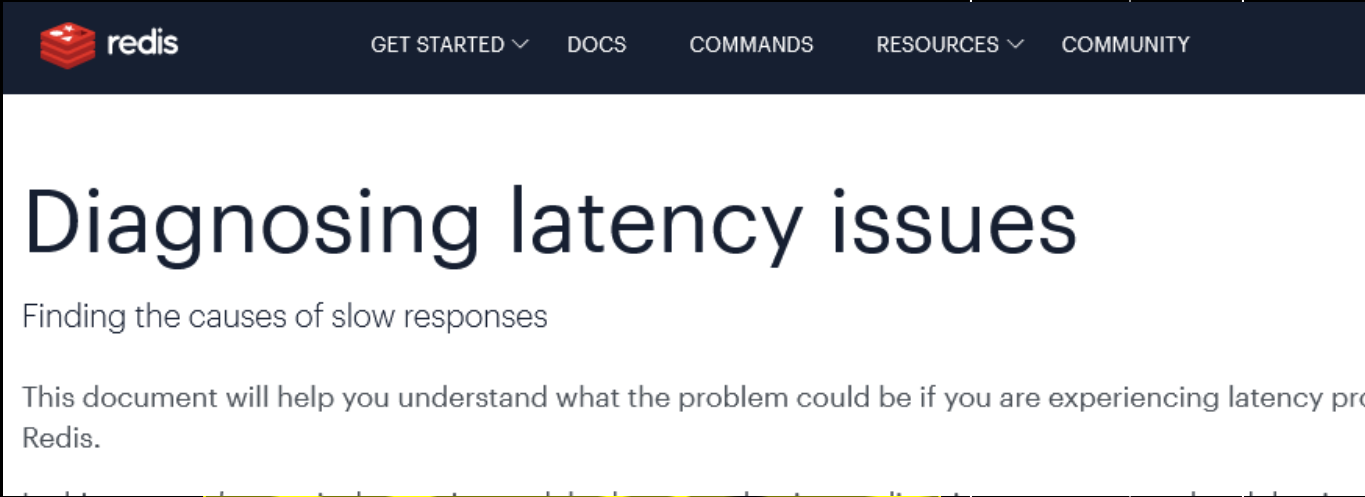


2. Cost-Unaware Policies



2. Cost-Unaware Policies

base (4KB) page fault



The screenshot shows the Redis website navigation bar with links for GET STARTED, DOCS, COMMANDS, RESOURCES, and COMMUNITY. The main heading is "Diagnosing latency issues" with a subtitle "Finding the causes of slow responses". Below this, it states: "This document will help you understand what the problem could be if you are experiencing latency problems with Redis."

huge page created, compacted

huge page created, compacted, reclaimed

PingCAP alloc failed, compacted, fallback Δ

Transparent Huge Pages: Why We Disable It for Databases

2020-12-10 Engineering

100 Latency $\leq 50\mu s$ $100\mu s$

Couchbase Server / Install & Upgrade / Deployment Guidelines / Disable THP

Disabling Transparent Huge Pages (THP)

Transparent huge pages (THP) is a memory management system that is enabled by default in most Linux distributions. THP must be disabled in order for Couchbase Server to function correctly on Linux.



Docs Home → Develop Applications → MongoDB Manual

Disable Transparent Huge Page

Transparent Huge Pages (THP) is a Linux memory management system that uses Lookaside Buffer (TLB) lookups on machines with large amounts of memory.

with THP enabled, b

2.10 Disabling Transparent Huge

Starting with Red Hat Enterprise Linux 6, Oracle Linux 6, SUSE 11 and Oracle Linux 5.11, THP is enabled at runtime. However, Transparent HugePages might cause delays in memory access. Oracle recommends that you disable Transparent HugePages on all Oracle Database servers to avoid performance issues.

Although Transparent HugePages are similar to HugePages, they are not preallocated at startup. Oracle recommends that you use standard HugePages.

3. Scattered implementations

Kernel

```
asm_cost_delta);
    should_do =
    vm_decide(rem, cost_delta);

    if (should_do) {
        ret = create_huge_page(vmf);
        if (!ret & VM_FAULT_FALLBACK)
            return ret;
    } else {
        // (naive) Entry is already present.
        pud_t orig_pud = *vmf.pud;
        barrier();
        if (pud_trans_huge(orig_pud) ||
            pud_devaprotect_pud) {
            // RMA case for anonymous PDEs
            would go here */
            if (dirty &&
                !pud_write(orig_pud)) {
                orig_pud;
                ret = wp_huge_page(vmf,
                    orig_pud);
                if (!ret &
                    VM_FAULT_FALLBACK)
                    return handle_get_fault(vmf) |
                        VM_FAULT_PAGE;
            }
        }
    }
}

// ret =
transparent_lbr_fake_fault()
// return ret;
goto escape_pud;
}

p4t = pud_lock(m, vmf.pud);
// We use the "live" pud here.
if (pud_present(*vmf.pud) &&
    is_huge_trap_enabled(vmf.vma->vm_mm,
        vmf.address))
    // *vmf.pud =
    pud_shraverse("VMF_PUD") // TODO mark uncommitt
    | else if (pud_present(*vmf.pud))
        *vmf.pud = pud_unreserve(*vmf.pud);
    spin_unlock(p4t);

escape_pud:
vmf.pud = pud_alloc(m, p4d, address);
if (!vmf.pud)
    return VM_FAULT_OOM;

retry_pud:
if (pud_same(*vmf.pud) &&
    transparent_hugepage_enabled(vma, address))
    // ret =
transparent_lbr_fake_fault()
// return ret;
goto escape_pud;
}

p4t = pud_lock(m, vmf.pud);
// We use the "live" pud here.
if (pud_present(*vmf.pud) &&
    is_huge_trap_enabled(vmf.vma->vm_mm,
        vmf.address))
    // *vmf.pud =
    pud_shraverse("VMF_PUD") // TODO mark uncommitt
    | else if (pud_present(*vmf.pud))
        *vmf.pud = pud_unreserve(*vmf.pud);
    spin_unlock(p4t);

escape_pud:
vmf.pud = pud_alloc(m, p4d, address);
if (!vmf.pud)
    return VM_FAULT_OOM;

retry_pud:
if (pud_same(*vmf.pud) &&
    transparent_hugepage_enabled(vma, address))
    // Address: user virtual address
    * pfn: location to store found PFN
    *
    * Only IO mappings and raw PFN mappings are
    allowed.
    * Return: zero and the pfn at @pfn on success,
    -ve otherwise.
    */
int follow_pfn(struct vm_area_struct *vma,
    unsigned long *pfn)
{
    int ret = -EINVAL;
    spinlock_t *p4t;
    p4t_t *p4tep;

    if (!!(vma->vm_flags & VM_IO |
        VM_PFNMAP))
        return ret;

    ret = follow_pfn(vma->vm_mm, address,
        &p4tep, &p4t);
    if (!ret)
        return ret;
    if (!ret)
        return ret;
    *pfn = get_pfn(*p4tep);
    pfn_unmap_unlock(*p4tep, p4t);
    return 0;
}
EXPORT_SYMBOL(follow

* Address: user virtual address
* pfn: location to store found PFN
*
* Only IO mappings and raw PFN mappings are
allowed.
* Return: zero and the pfn at @pfn on success,
-ve otherwise.
*/
int follow_pfn(struct vm_area_struct *vma,
    unsigned long *pfn)
{
    int ret = -EINVAL;
    spinlock_t *p4t;
    p4t_t *p4tep;

    if (!!(vma->vm_flags & VM_IO |
        VM_PFNMAP))
        return ret;

    ret = follow_pfn(vma->vm_mm, address,
        &p4tep, &p4t);
    if (!ret)
        return ret;
    if (!ret)
        return ret;
    *pfn = get_pfn(*p4tep);
    pfn_unmap_unlock(*p4tep, p4t);
    return 0;
}
EXPORT_SYMBOL(follow
```



3. Scattered implementations

Kernel

```
asm_cost_delta);
    should_do =
    mm_decidelem_cost_delta);
    if (should_do)
        hope pud(vmf);
        if (!
            FAULT_FALLBACK)
        }
        // (reason
        pud_t orig_pud = *vmf.pud;
        handle();
        if (pud_trans_huge(orig_pud) {
            pud_decompress_pud);
        }
        // RMA case for anonymous PDE
        would go here */
        if (dirty &&
            !pud_write(orig_pud)) {
            orig_pud;
            ret = wp_huge_pud(vmf,
                orig_pud);
            if (!ret &
                VM_FAULT_FALLBACK)
                return handle_gpte_fault(vmf);
            VM_FAULT_BADPUD;
        }
    }
}

// ret =
transparent_lbr_fake_fault();
// return ret;
goto escape_pud;
}
p4t = pud
// We use
if (pud
    &&
    is_hugep_trap_enabled(vma->vm_mm,
        vmf.address))
    // *vmf.pud =
    pud_shraverse(*vmf.pud) // XXXX mark uncommen
    // also if (pud_present(*vmf.pud)
    {
        *vmf.pud = pud_unreserve(*vmf.pud);
        spin_unlock(pt1);
    }
    escape_pud;
    vmf.pud = pud
    if (!ret)
        return VM_FAULT_BADPUD;
    retry_pud;
    if (pud_same(*vmf.pud) &&
        transparent_hugepage_enabled(vma, address))
}

// ret =
transparent_lbr_fake_fault();
// return ret;
goto escape_pud;
}
p4t = pud
// We use
if (pud
    &&
    is_hugep_trap_enabled(vma->vm_mm,
        vmf.address))
    // *vmf.pud =
    pud_shraverse(*vmf.pud) // XXXX mark uncommen
    // also if (pud_present(*vmf.pud)
    {
        *vmf.pud = pud_unreserve(*vmf.pud);
        spin_unlock(pt1);
    }
    escape_pud;
    vmf.pud = pud_alloc(mm, p4d, address);
    if (!vmf.pud)
        return VM_FAULT_BADPUD;
    retry_pud;
    if (pud_same(*vmf.pud) &&
        transparent_hugepage_enabled(vma, address))
}

/* Address: user virtual address
 * @pfn: location to store found PFN
 *
 * Only IO mappings and raw PFN mappings are
 * allowed.
 *
 * Return: zero and the pfn at @pfn on success,
 * -ve otherwise.
 */
int follow_pfn(struct vm_area_struct *vma,
    unsigned long addr,
    unsigned long
}
int ret = -EINVAL;
spinlock_t *pt1;
pte_t *ptep;
if (!!(vma->vm_flags & VM_IO |
    VM_PFNMAP))
    return ret;
ret = follow_pfn(vma->vm_mm, address,
    &ptep, &pfn);
if (ret)
    return ret;
*pfn = pte_pfn(*ptep);
pte_unmap_unlock(ptep, pt1);
return 0;
}
EXPORT_SYMBOL(follow

/* Address: user virtual address
 * @pfn: location to store found PFN
 *
 * Only IO mappings and raw PFN mappings are
 * allowed.
 *
 * Return: zero and the pfn at @pfn on success,
 * -ve otherwise.
 */
int follow_pfn(struct vm_area_struct *vma,
    unsigned long address,
    unsigned long *pfn)
{
    int ret = -EINVAL;
    spinlock_t *pt1;
    if (!!(vma->vm_flags & VM_IO |
        VM_PFNMAP))
        return ret;
    ret = follow_pfn(vma->vm_mm, address,
        &ptep, &pfn);
    if (ret)
        return ret;
    *pfn = pte_pfn(*ptep);
    pte_unmap_unlock(ptep, pt1);
    return 0;
}
EXPORT_SYMBOL(follow
```

3. Scattered implementations

```
markm ~/linux/mm
$ grep CONFIG_TRANSPARENT_HUGEPAGE
page_alloc.c
page_idle.c
vmscan.c
huge_memory.c
Makefile
rmap.c
mempolicy.c
memcontrol.c
hmm.c
gup.c
vmstat.c
page_io.c
memory.c
pgtable-generic.c
advise.c
swap.c
migrate.c
```

Huge page implementation!

Roadmap

Roadmap

Three challenges to consistent kernel MM behavior:

1. Low-quality information
2. Cost-unaware policies
3. Scattered implementations

Roadmap

Three challenges to consistent kernel MM behavior:

1. Low-quality information
2. Cost-unaware policies
3. Scattered implementations

Cost-benefit Memory Management (CBMM)

- Userspace cost < userspace benefit
- Centralized policy implementation
- Profiles augment kernel info

 github.com/multifacet/cbmm-artifact



Roadmap

Three challenges to consistent kernel MM behavior:

1. Low-quality information
2. Cost-unaware policies
3. Scattered implementations

Cost-benefit Memory Management (CBMM)

- Userspace cost < userspace benefit
- Centralized policy implementation
- Profiles augment kernel info

 github.com/multifacet/cbmm-artifact



Results

- Improve soft page fault tail latency by orders of magnitude
- Match or improve performance, especially if fragmented

Roadmap

Three challenges to consistent kernel MM behavior:

1. Low-quality information
2. Cost-unaware policies
3. Scattered implementations

➔ Cost-benefit Memory Management (CBMM)

- Userspace cost < userspace benefit
- Centralized policy implementation
- Profiles augment kernel info

 github.com/multifacet/cbmm-artifact



Results

- Improve soft page fault tail latency by orders of magnitude
- Match or improve performance, especially if fragmented

A red ribbon graphic with a white outline, featuring a central rectangular section and two pointed ends. The text "CBMM: KEY IDEA" is centered within the central section.

CBMM: KEY IDEA

A red ribbon graphic with a white outline, featuring a central rectangular section and two pointed ends. The text "CBMM: KEY IDEA" is written in white, bold, uppercase letters across the central section.

CBMM: KEY IDEA

All MM operations have a cost and a benefit to userspace.

A red ribbon graphic with a white outline, featuring a central rectangular section and two pointed ends that taper to the left and right. The text "CBMM: KEY IDEA" is written in white, bold, uppercase letters across the central section.

CBMM: KEY IDEA

All MM operations have a cost and a benefit to userspace.

Ex: Copy-on-write

Benefit: processor cycles not spent copying memory

Cost: processor cycles spent on extra page faults

CBMM: KEY IDEA

All MM operations have a cost and a benefit to userspace.

Ex: Copy-on-write

Benefit: processor cycles not spent copying memory

Cost: processor cycles spent on extra page faults

Ensure that cost < benefit.



CBMM Design: Overview

Kernel

Estimator



```
mm_cout_dirtal);
    should do a
    we decide (mm_cout_dirtal);

    if (!should_do) {
        set = create_huge_pud(vmf);
        if (!test & VM_FAULT_FALLBACK)
            return set;
    } else {
        // (main) Entry is already present.
        pud_t orig_pud = *vmf.pud;
        barrier();
        if (pud_testing_huge(orig_pud))
            pud_dewmap(orig_pud);
        /* WMA case for anonymous PUDs
        would go here */
        if (dirty &&
            !pud_writes(orig_pud)) {
            set = wp_huge_pud(vmf,
                orig_pud);
            if (!test &
                (vmf.flags & VM_FAULT_WRITE))
                return set;
            vmf.pud = pud_unreserve(*vmf.pud);
            spin_unlock(ptl);
            escape_pud:
            vmf.pud = pud_alloc(mm, p4d, address);
            if (!test & VM_FAULT_OOM)
                return VM_FAULT_OOM;
        }
        return set;
    }
}

/* Returns: user virtual address
 * @pfn: location to store found PFN
 * Only IO mappings and raw PFN mappings are
 * allowed.
 * Returns: zero and the pfn at @pfn on success,
 * -ve otherwise.
 */
int follow_pfn(struct vm_area_struct *vma,
    unsigned long address,
    unsigned long *pfn)
{
    int ret = -EINVAL;
    spinlock_t *ptl;
    pte_t *ptep;

    if ((vma->vm_flags & VM_IO) &
        VM_SHARED))
        return ret;
    ret = follow_pte(vma->vm_mm, address,
        &ptep, &ptl);
    if (test)
        return ret;
    *pfn = pfn_pte(*ptep);
    pte_unmap_unlock(ptep, ptl);
    return 0;
}
EXPORT_SYMBOL(follow_pfn);

/* Returns: user virtual address
 * @pfn: location to store found PFN
 * Only IO mappings and raw PFN mappings are
 * allowed.
 * Returns: zero and the pfn at @pfn on success,
 * -ve otherwise.
 */
int follow_pfn(struct vm_area_struct *vma,
    unsigned long address,
    unsigned long *pfn)
{
    int ret = -EINVAL;
    spinlock_t *ptl;
    pte_t *ptep;

    if ((vma->vm_flags & VM_IO) &
        VM_SHARED))
        return ret;
    ret = follow_pte(vma->vm_mm, address,
        &ptep, &ptl);
    if (test)
        return ret;
    *pfn = pfn_pte(*ptep);
    pte_unmap_unlock(ptep, ptl);
    return 0;
}
EXPORT_SYMBOL(follow_pfn);
```

Centralized MM policies


CBMM Design: Overview

Kernel

Estimator



Cost-Benefit Models



Workload Profiles

```
mm_cst_delta)
should_go =
mm_decide(mm_cst_delta)
    if (should_go) {
        ret = create_huge_pod(vmf);
        if (ret & VM_FAULT_FALLBACK)
            return ret;
    }
    // (marks) Entry is already present.
    pod_t orig_pod = *vmf.pod;
    barrier();
    if (pod_ksize_huge(orig_pod))
        pod_dump(orig_pod);
    /* NB: case for anonymous PIDs
would go here */
    if (dirty &&
        !pod_write(orig_pod)) {
        ret = wp_huge_pod(vmf,
            orig_pod);
        if (ret &
            VM_FAULT_FALLBACK)
            return handle_pte_fault(vmf);
        VM_FAULT_PAGE;
    }
}

ret =
    1; // 2020 warn uncomment
    return ret;
    to escape_pod;

vmf.pod =
    vmf.pod; // 2020 warn uncomment
    if (pod_present(*vmf.pod))
        *vmf.pod = pod_unreserve(*vmf.pod);
    spin_unlock(pci);

escape_pod:
    *vmf.pod = pod_alloc(mm, pfd, address);
    vmf.pod;
    return VM_FAULT_OOM;

retry_pod:
    if (pod_name(*vmf.pod) &&
        !transparent_hugepage_enabled(vma, address))
        goto vmf.pod;

pci = pod_lock(mm, vmf.pod);
    "line" pod here.
    mci(*vmf.pod)
    *vma->vm_mm;

/* Address: user virtual address
 * @pfn: location to store found PPN
 *
 * Returns: zero and the pfn at @pfn on success,
 * -ve otherwise.
 */
int follow_pfn(struct vm_area_struct *vma,
    unsigned long address,
    unsigned long *pfn)
{
    int ret = -EINVAL;
    spinlock_t *pfn;
    if (!pfn)
        return ret;
    if ((vma->vm_flags & VM_IO) &
        VM_PFNMAP))
        return ret;
    ret = follow_pte(vma->vm_mm, address,
        &pfn, &pci);
    if (ret)
        return ret;
    *pfn = pfn;
    pte_unmap_unlock(pfn, pci);
    return 0;
}
EXPORT_SYMBOL(follow
```

Centralized MM policies

Consult Estimator for each policy decision

Cost-Benefit Models

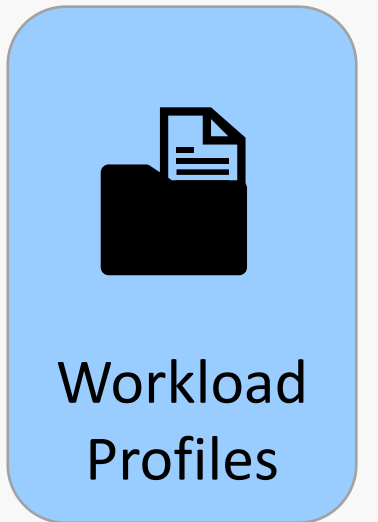
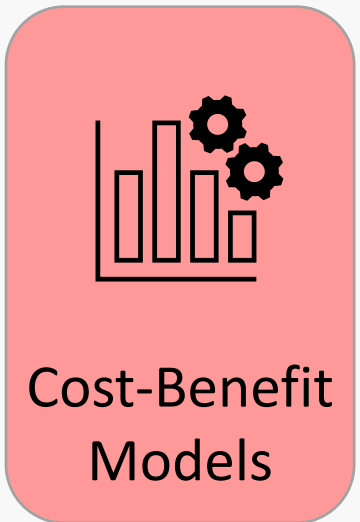
Units: userspace cycles

Workload Profiles

CBMM Design: Overview

Kernel

Estimator



```
mm_cost_delta)
should_go =
mm_decide(mm_cost_delta)
if (should_go)
    ret = create_huge_page(vmf);
    if (!ret & VM_FAULT_FALLBACK)
        return ret;
} else {
    // (mark) Entry is already present.
    pod_t orig_pod = *vmf.pod;
    barrier();
    if (pod_khuge_huge(orig_pod))
        pod_dump(orig_pod);
    /* NB: case for anonymous PIDs
would go here */
    if (dirty &&
        !pod_write(orig_pod))
        ret = wp_huge_page(vmf,
            orig_pod);
    if (!ret &
        VM_FAULT_FALLBACK)
        return handle_pte_fault(vmf);
    VM_FAULT_PAGE;
}

ret =
    get_fake_fault();
return ret;
to_escape_pod;

vmf.pod =
    pod_lock(vm, vmf.pod);
we use the "live" pod here.
(pod_present(*vmf.pod)
    &&
    !vma_enabled(vma->vm_mm,
        *vmf.pod)
    &&
    !vmf.pod) // 2020 mark comment
    like if (pod_present(*vmf.pod))
        *vmf.pod = pod_unreserve(*vmf.pod);
    spin_unlock(pci);
}

escape_pod:
    *vmf.pod = pod_alloc(mm, pfd, address);
    vmf.pod;
return VM_FAULT_OOM;

retry_pod:
    if (pod_name(*vmf.pod) &&
        transparent_hugepage_enabled(vma, address))
        return vmf.pod;

pci = pod_lock(vm, vmf.pod);
"live" pod here.
mci(*vmf.pod)
    &&
    !vma->vm_mm,
    vmf.pod);
// 2020 mark comment
pod_present(*vmf.pod);
pod_unreserve(*vmf.pod);
pci);
}

vmf.pod = pod_alloc(mm, pfd, address);
return VM_FAULT_OOM;
}

*vmf.pod = pod_unreserve(*vmf.pod);
spin_unlock(pci);
}

+ Address: user virtual address
+ pfn: location to store found PTE
+
+ Returns: zero and the pfn at pfn on success,
+ve otherwise.
+
int follow_pfn(struct vm_area_struct *vma,
    unsigned long address,
    unsigned long *pfn)
{
    int ret = -EINVAL;
    spinlock_t *pfn;
    if (!pfn)
        return ret;
    if ((vma->vm_flags & VM_IO)
        VM_PFNMAP))
        return ret;
    ret = follow_pte(vma->vm_area, address,
        &pfn, &ret);
    if (ret)
        return ret;
    *pfn = pfn;
    pte_unmap_unlock(pfn, pci);
    return 0;
}
EXPORT_SYMBOL(follow
```

Centralized MM policies

Consult Estimator for each policy decision

Cost-Benefit Models

Units: userspace cycles

Workload Profiles

Creating and Implementing Models

Estimator



Cost-Benefit
Models

Creating and Implementing Models

Estimator



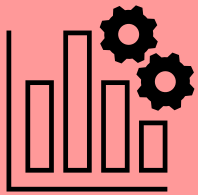
Cost-Benefit
Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
)  
{  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

Model

Creating and Implementing Models

Estimator



Cost-Benefit
Models

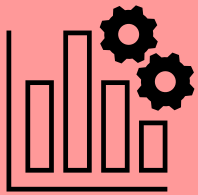
```
mm_estimate_t my_model(  
    mm_op_t *op  
)  
{  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

MM operation info

Model

Creating and Implementing Models

Estimator



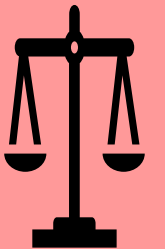
Cost-Benefit
Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
)  
{  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

MM operation info

Model

Cost, Benefit
Estimates



Creating and Implementing Models

Estimator



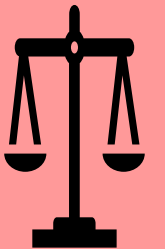
Cost-Benefit
Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
)  
{  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

MM operation info
Kernel/HW State

Model

Cost, Benefit
Estimates



Creating and Implementing Models

Estimator



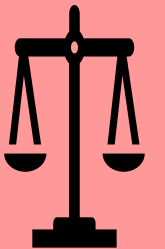
Cost-Benefit
Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
)  
{  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

MM operation info
Kernel/HW State
Preloaded Profiles

Model

Cost, Benefit
Estimates



Creating and Implementing Models

Estimator



Cost-Benefit
Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
)  
{  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

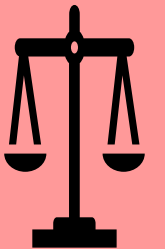
MM operation into
Kernel/HW State
Preloaded Profiles



Model



Cost-Benefit
Estimates



Ex: Huge Page Model

Implementing Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
) {  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

MM operation into
Kernel/HW State
Preloaded Profiles



Model



Cost Benefit
Estimates



Ex: Huge Page Model

COST

Have Huge Pages?

N

1s

Implementing Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
) {  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

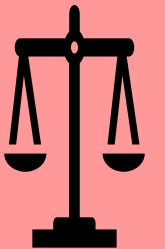
MM operation into
Kernel/HW State
Preloaded Profiles



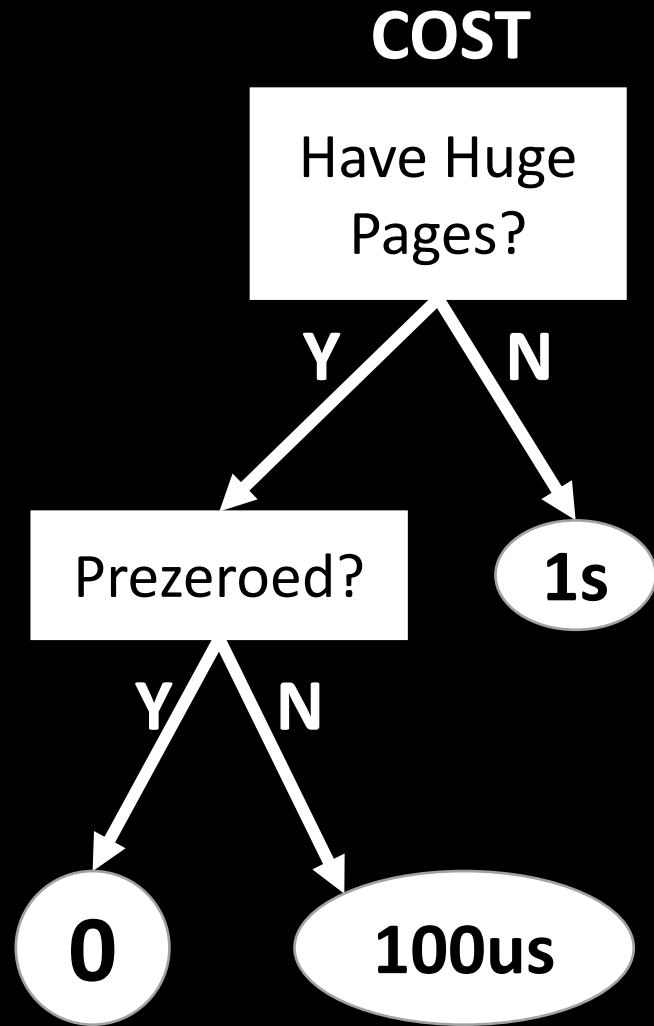
Model



Cost Benefit
Estimates



Ex: Huge Page Model



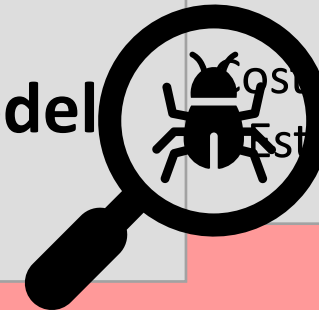
Implementing Models

```
mm_estimate_t my_model(  
    mm_op_t *op  
) {  
    ... // Logic  
  
    return mm_estimate_t {  
        .cost = ...,  
        .benefit = ...,  
    };  
}
```

MM operation into
Kernel/HW State
Preloaded Profiles



Model



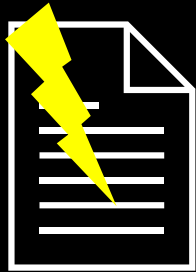
Cost Benefit
Estimates



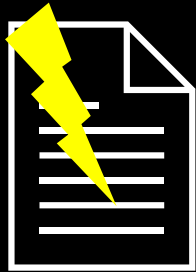
Example: Huge Page Cost-Benefit Model

Example: Huge Page Cost-Benefit Model

Page fault: use base or huge pages?



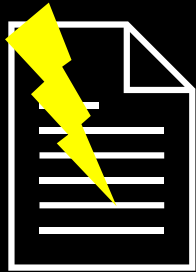
Example: Huge Page Cost-Benefit Model



Page fault: use base or huge pages?

Huge avoids 10us of TLB misses?

Example: Huge Page Cost-Benefit Model

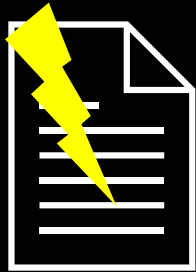


Page fault: use base or huge pages?

Huge avoids 10us of TLB misses?

Cost estimate: 100us -> Use base pages

Example: Huge Page Cost-Benefit Model



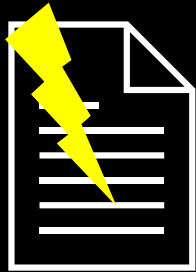
Page fault: use base or huge pages?

Huge avoids 10us of TLB misses?

Cost estimate: 100us -> Use base pages

Cost estimate: 0 -> Use huge pages

Example: Huge Page Cost-Benefit Model



Page fault: use base or huge pages?

Huge avoids ^{???}10us of TLB misses?

Cost estimate: 100us -> Use base pages

Cost estimate: 0 -> Use huge pages

Profile Generation and Maintenance

Estimator



Cost-Benefit
Models

Profile Generation and Maintenance

Estimator



Cost-Benefit
Models

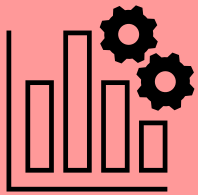


Workload
Profiles

Address Range	Benefit
0x0 to 0x7fcc76400000	243632025
0x7fcd83400000 to 0x7fce09c00000	229284
0x7ffbbf400000 to 0x7ffc45c00000	1311889877
...	

Profile Generation and Maintenance

Estimator



Cost-Benefit
Models



Workload
Profiles

Address Range	Benefit
0x0 to 0x7fcc76400000	243632025
0x7fcd83400000 to 0x7fce09c00000	229284
0x7ffbbf400000 to 0x7ffc45c00000	1311889877
...	

$$A - B = N \text{ cycles}$$



Profile Generation and Maintenance

Estimator



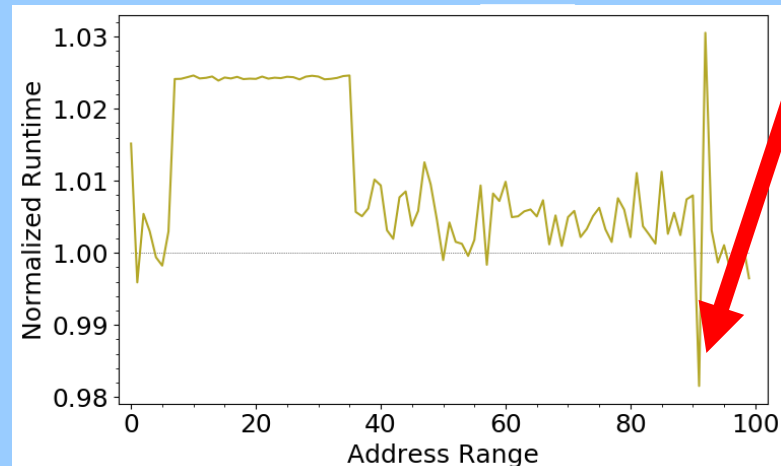
Cost-Benefit
Models



Workload
Profiles

Address Range	Saved usermode cycles
0x0 to 0x7fcc76400000	243632025
0x7fcd83400000 to 0x7fce09c00000	229284
0x7ffbbf400000 to 0x7ffc45c00000	1311889877
...	

$$\text{(Huge)} - \text{(Base)} = 1311889877$$



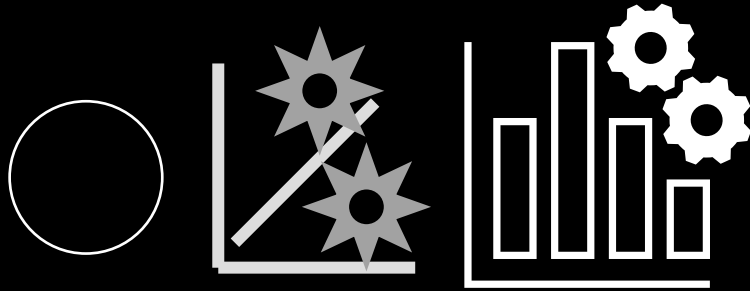
All together...

All together...

Before Deployment

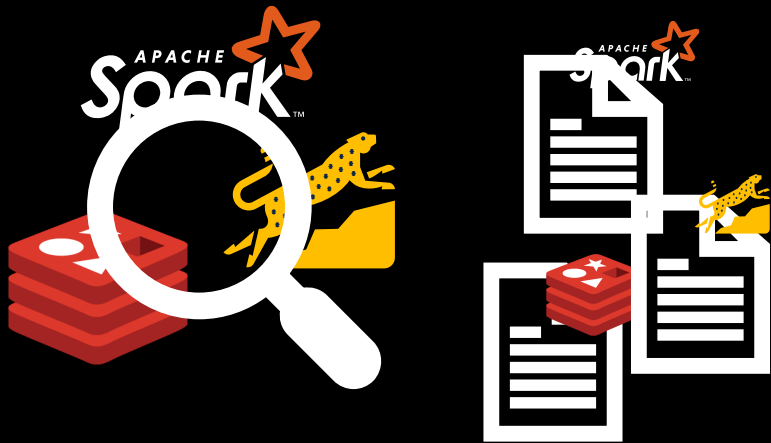
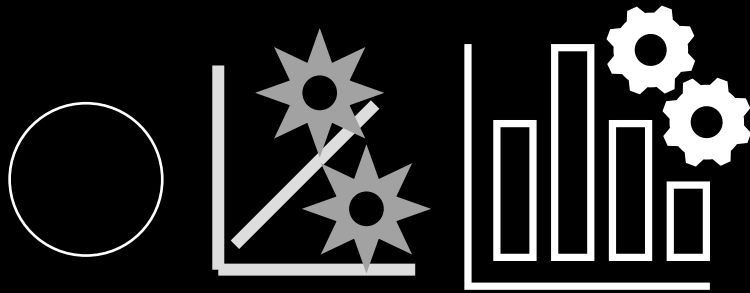
All together...

Before Deployment



All together...

Before Deployment

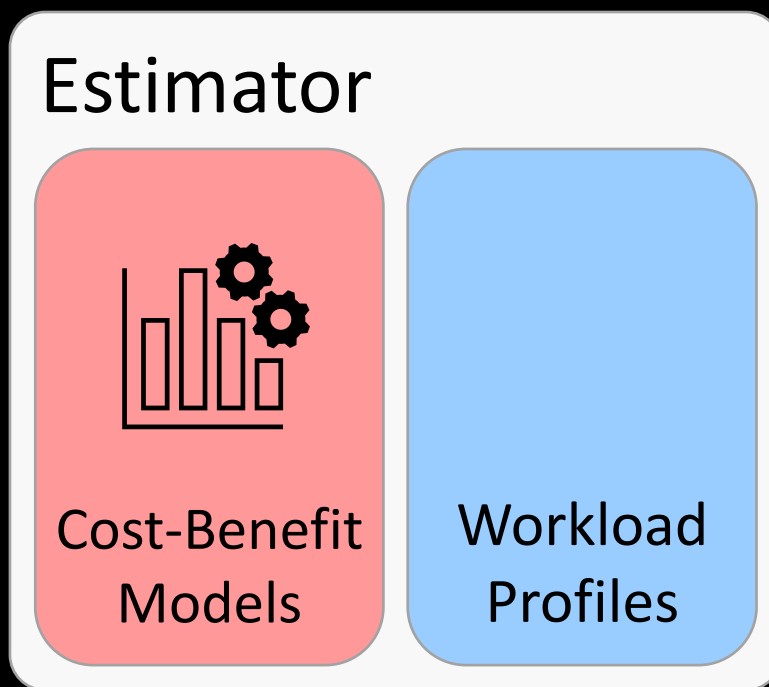


All together...

Before Deployment



Runtime

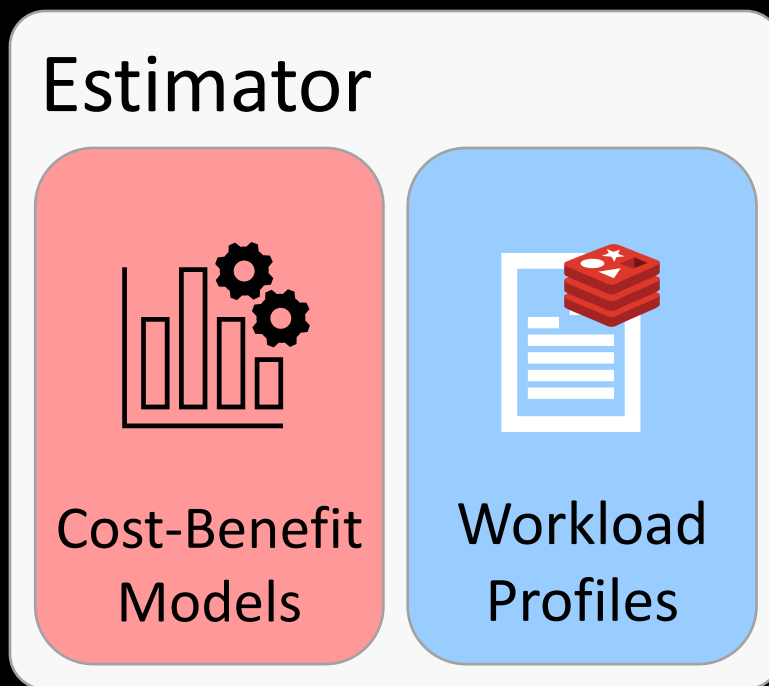


All together...

Before Deployment



Runtime

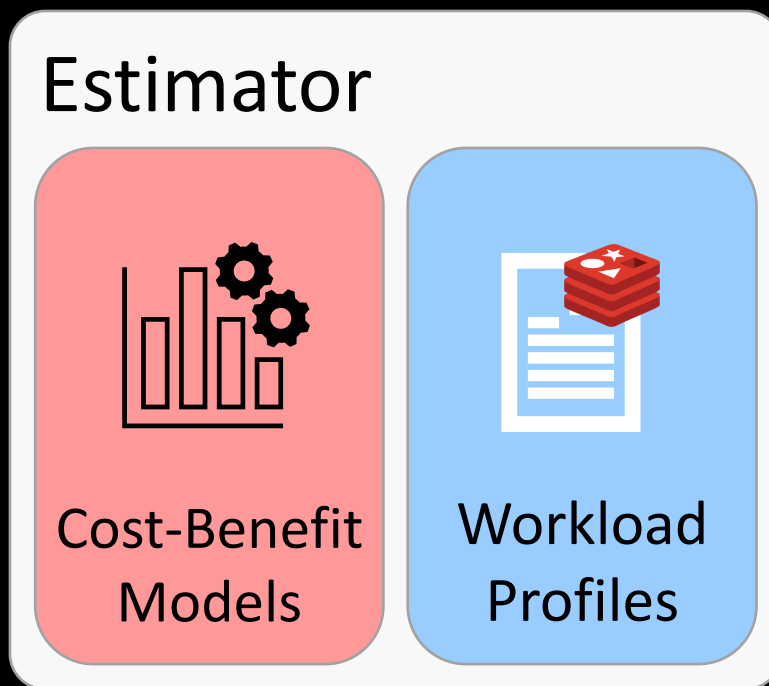


All together...

Before Deployment



Runtime



e.g., `handle_mm_fault()`
use huge page?

```
void __cpuinit(mm_init())
{
    should_do =
    mm_should_do(mm_init());
}

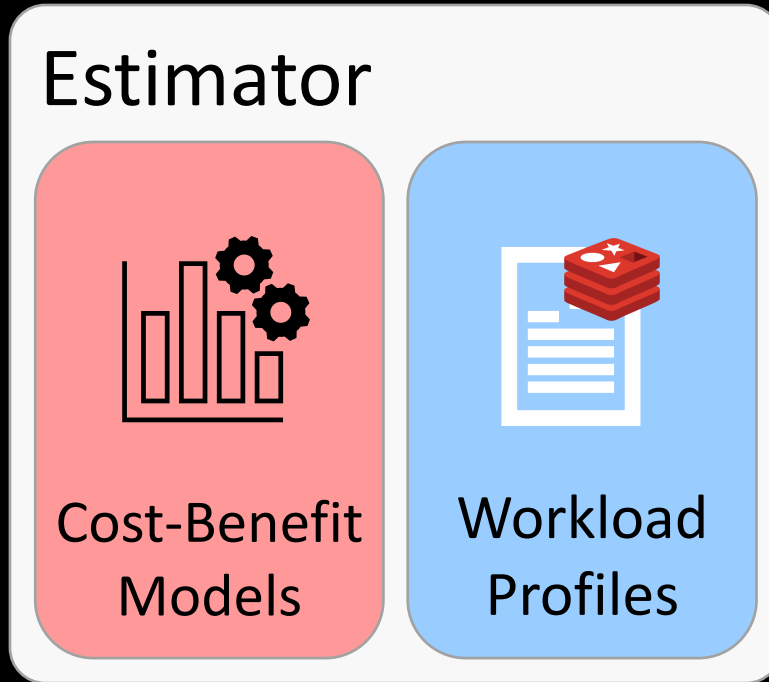
if (should_do) {
    mm_init();
    if (test & MM_FAULT_FALLBACK)
        return (ret);
} else {
    // Invariant: Entry is already present.
    pod_t orig_pod = *pod;
    barrier();
    if (pod_trans_huge_forcing_pod) {
        pod_detach(orig_pod);
        /* MMIO case for anonymous PMDs
        would go here */
        if (test &
            'pod_write(orig_pod)') {
            orig_pod;
            ret = op_huge_pod_init,
            if (test &
                MM_FAULT_FALLBACK)
                return handle_mm_fault(0);
            VM_FAULT_BUG_PAGE;
        }
    }
}
```

All together...

Before Deployment



Runtime



e.g., `handle_mm_fault()`
use huge page?

```
void __cpuinit(mm_init()) {
    should_do =
    mm_decide(mm_init_decision);
    if (should_do) {
        use_huge_page = mm_decide(mm_init_decision);
        return use_huge_page;
    }
    // In some cases, huge page is already present.
    // If so, we should use huge page.
    barrier();
    if (pod_trans_huge_forcing_pod) {
        pod_trans_huge_forcing_pod = 1;
        /* MMIO case for anonymous PMDs
        would go here */
        if (likely) {
            use_huge_page = mm_decide(mm_init_decision);
            return use_huge_page;
        }
        if (likely) {
            use_huge_page = mm_decide(mm_init_decision);
            return use_huge_page;
        }
        return mm_decide(mm_init_decision);
    }
}
```

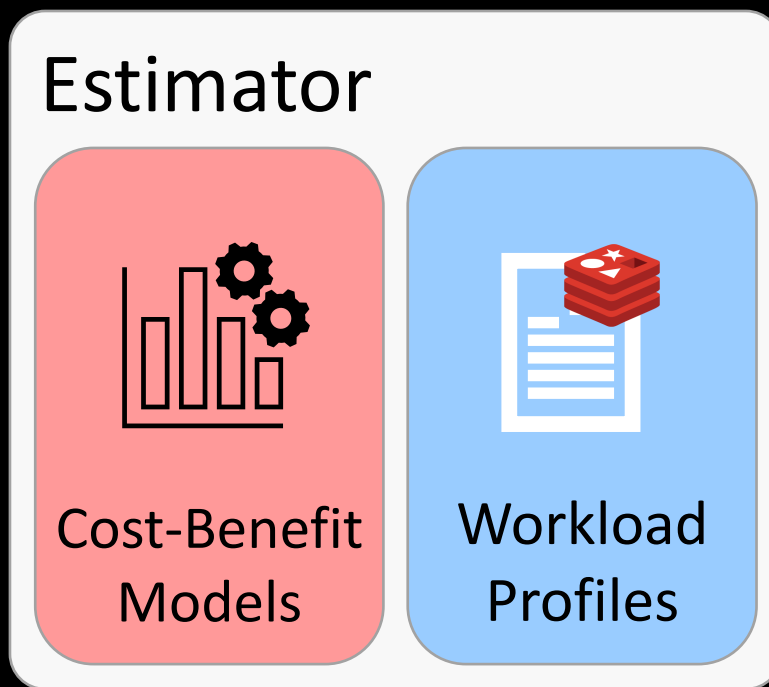


All together...

Before Deployment



Runtime



e.g., `handle_mm_fault()`
use huge page?

```
void __cpuinit(mm_cpu_data) {
    should_do =
    mm_decide(mm_cpu_data);
    if (should_do) {
        mm = mm_cpu_data;
        if (test & MM_FAULT_FALLBACK)
            return (ret);
    }
    // In some Entry is already present.
    pod_t orig_pod = *pod;
    barrier();
    if (pod_trans_huge_forcing_pod) {
        pod_destmap(orig_pod);
        /* MMIO case for anonymous PMDs
        would go here */
        if (dirty &&
            !pod_write(orig_pod)) {
            orig_pod;
            ret = op_huge_podriver;
        }
        if (test &
            MM_FAULT_FALLBACK)
            return handle_mm_fault(&mm);
        VM_FAULT_BUG_PAGE;
    }
}
```

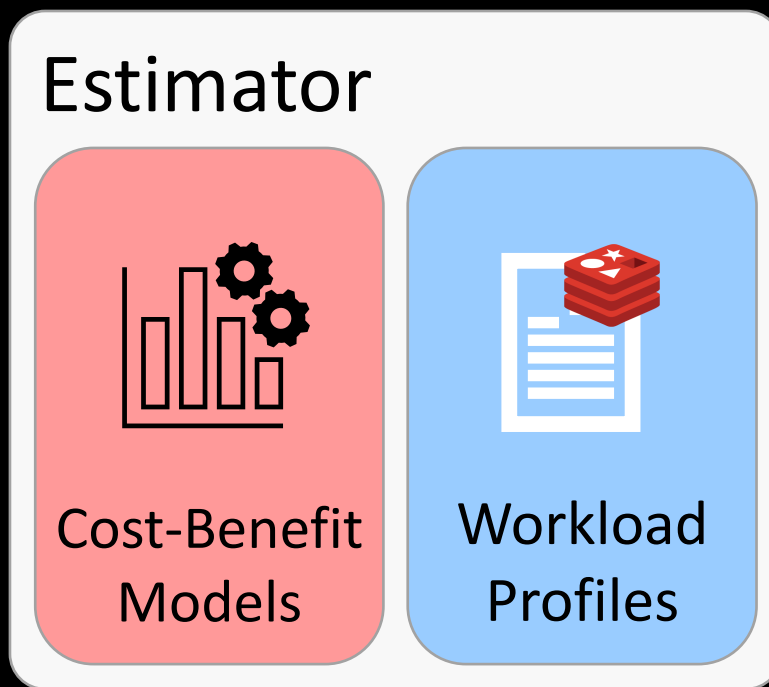


All together...

Before Deployment



Runtime



e.g., `handle_mm_fault()`
use huge page?

```
void __cpuinit(mm_fault) {
    should_do =
    mm_decide(mm_fault);
    if (should_do) {
        // should do huge page
        if (test & VM_FAULT_FALLBACK)
            return (ret);
    }
    // (max) Entry is already present.
    pud_t orig_pud = *pud;
    *pud = orig_pud;
    // MMIO case for anonymous PDBs
    would go here ??
    if (likely &&
        !pud_wise(orig_pud)) {
        ret = pg_huge_pud_alloc(
            orig_pud);
        if (ret &&
            VM_FAULT_FALLBACK)
            return handle_mm_fault(mm_fault);
        VM_FAULT_HUGE_PAGE;
    }
}
```

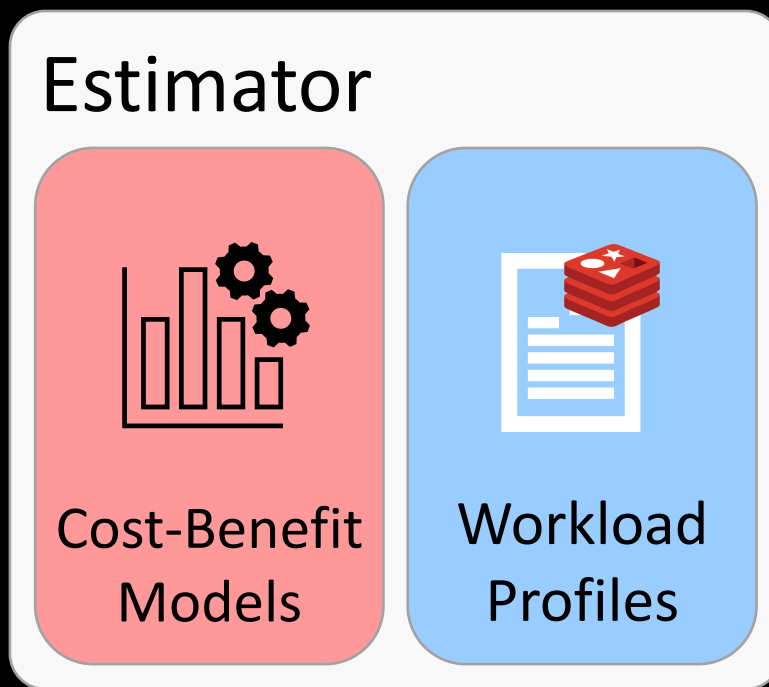


All together...

Before Deployment



Runtime



e.g., `handle_mm_fault()`
use huge page?

```
void __cpuinit(mm_fault) {
    should_do =
    mm_decide(mm_fault);
    if (should_do) {
        // should do huge page
        if (test & VM_FAULT_FALLBACK)
            return (ret);
    }
    // (max) Entry is already present.
    pod_t orig_pod = *pod;
    // (max)
    if (pod_trans_huge_fault) {
        pod_trans_huge_fault();
        /* MMIO case for anonymous PMDs
        would go here */
        if (likely &&
            !pod_wsize(orig_pod)) {
            ret = op_huge_podover,
            orig_pod);
            if (test &
                VM_FAULT_FALLBACK)
                return handle_mm_fault(mm_fault);
            VM_FAULT_HUGE_PAGE;
        }
    }
}
```



Implementation and Evaluation

Implementation and Evaluation

Implementation based on Linux

Implementation and Evaluation

Implementation based on Linux

Eval questions:

- Improved MM op tail latency?

Implementation and Evaluation

Implementation based on Linux

Eval questions:

- Improved MM op tail latency?
- Match performance?

Implementation and Evaluation

Implementation based on Linux

Eval questions:

- Improved MM op tail latency?
- Match performance?

Methodology:

- Baseline: Linux 5.5.8
- With and without fragmentation
- Much more evaluation in paper!

Test Machine Specs

Intel Xeon Silver 4114

10C/20T @ 2.2 GHz

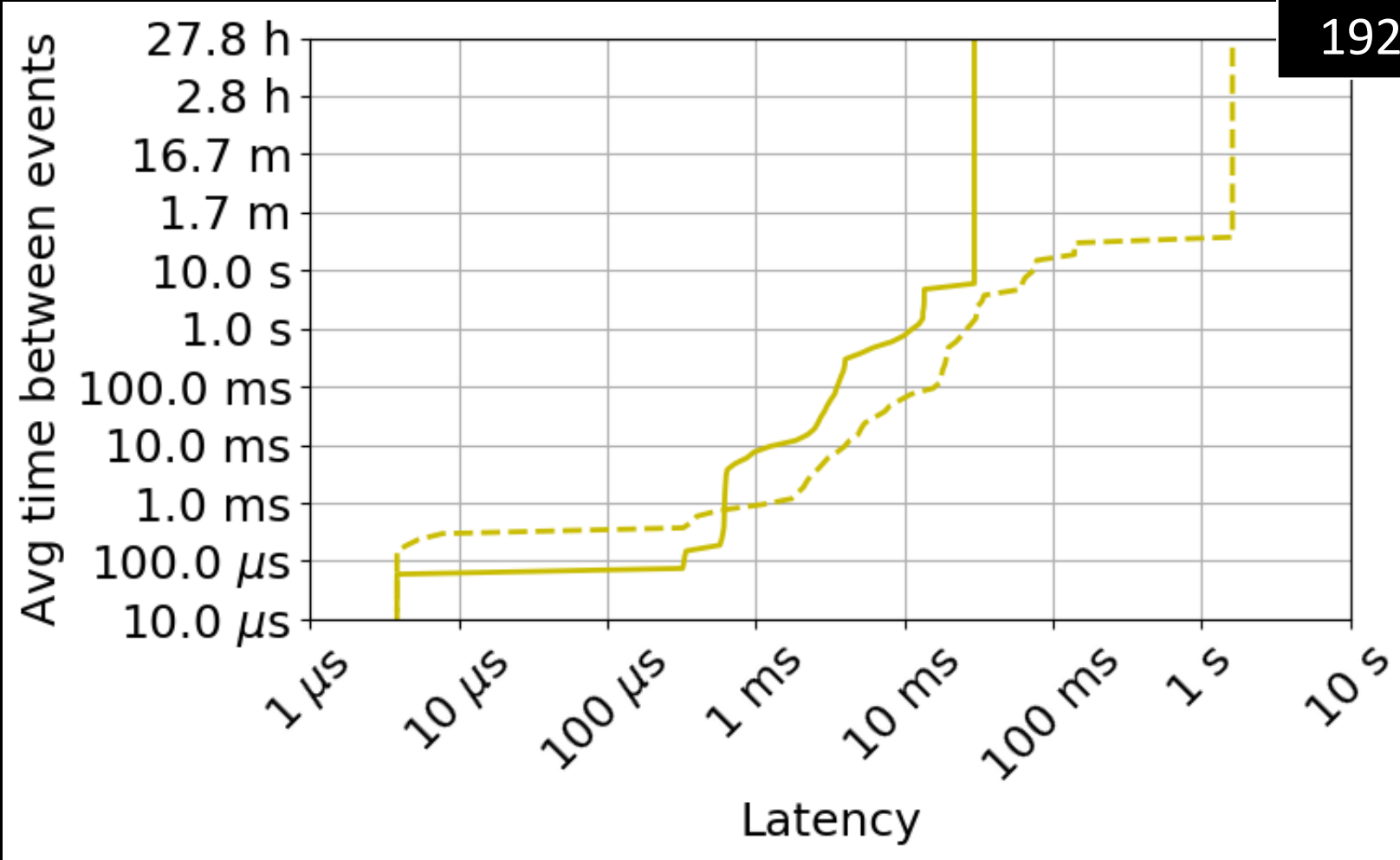
Skylake 2017

192GB ECC DRAM

480GB SSD

Soft Page Fault Tail Latency

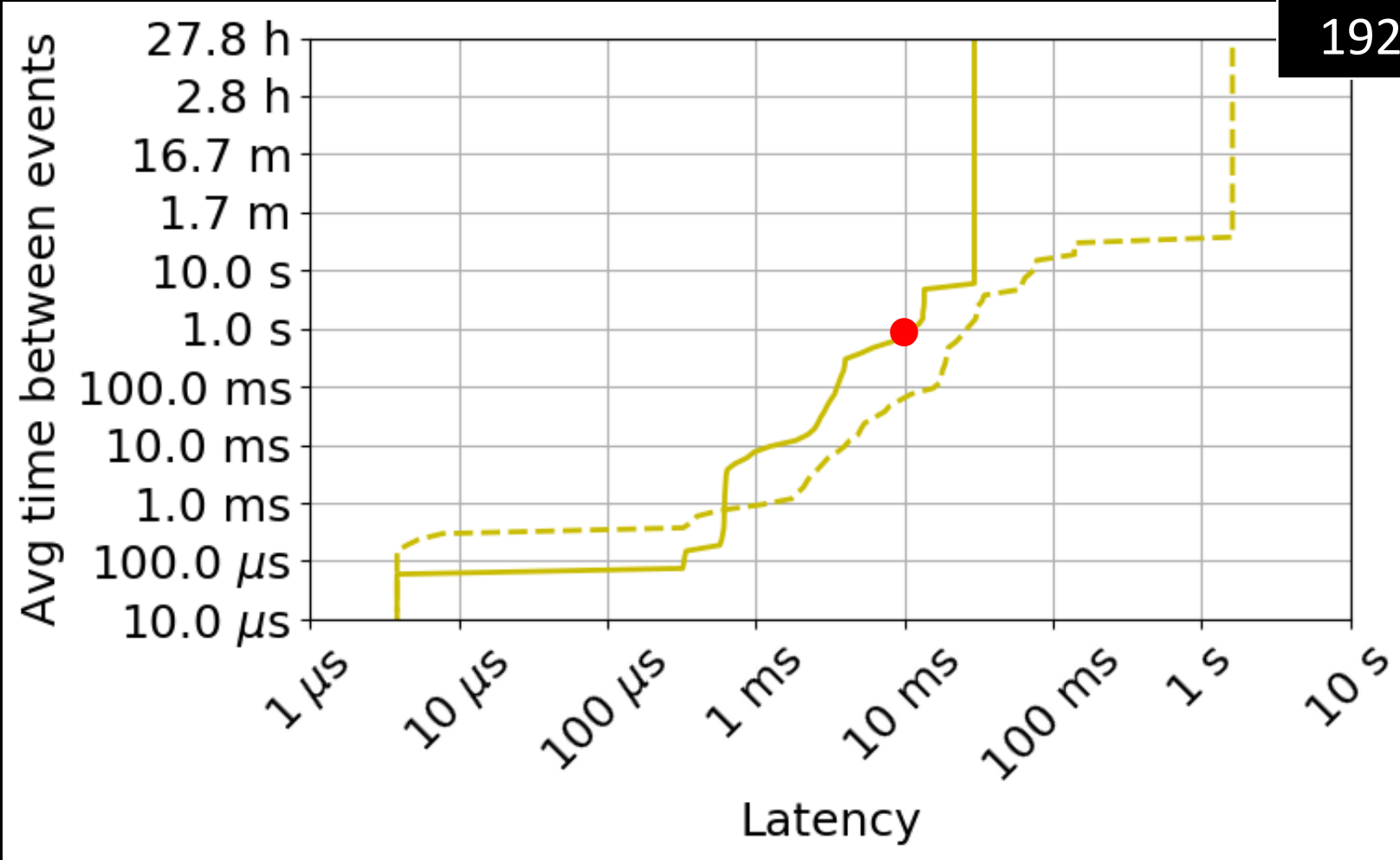
memcached
192GB peak mem



— Linux
- - - Linux, fragmented

Soft Page Fault Tail Latency

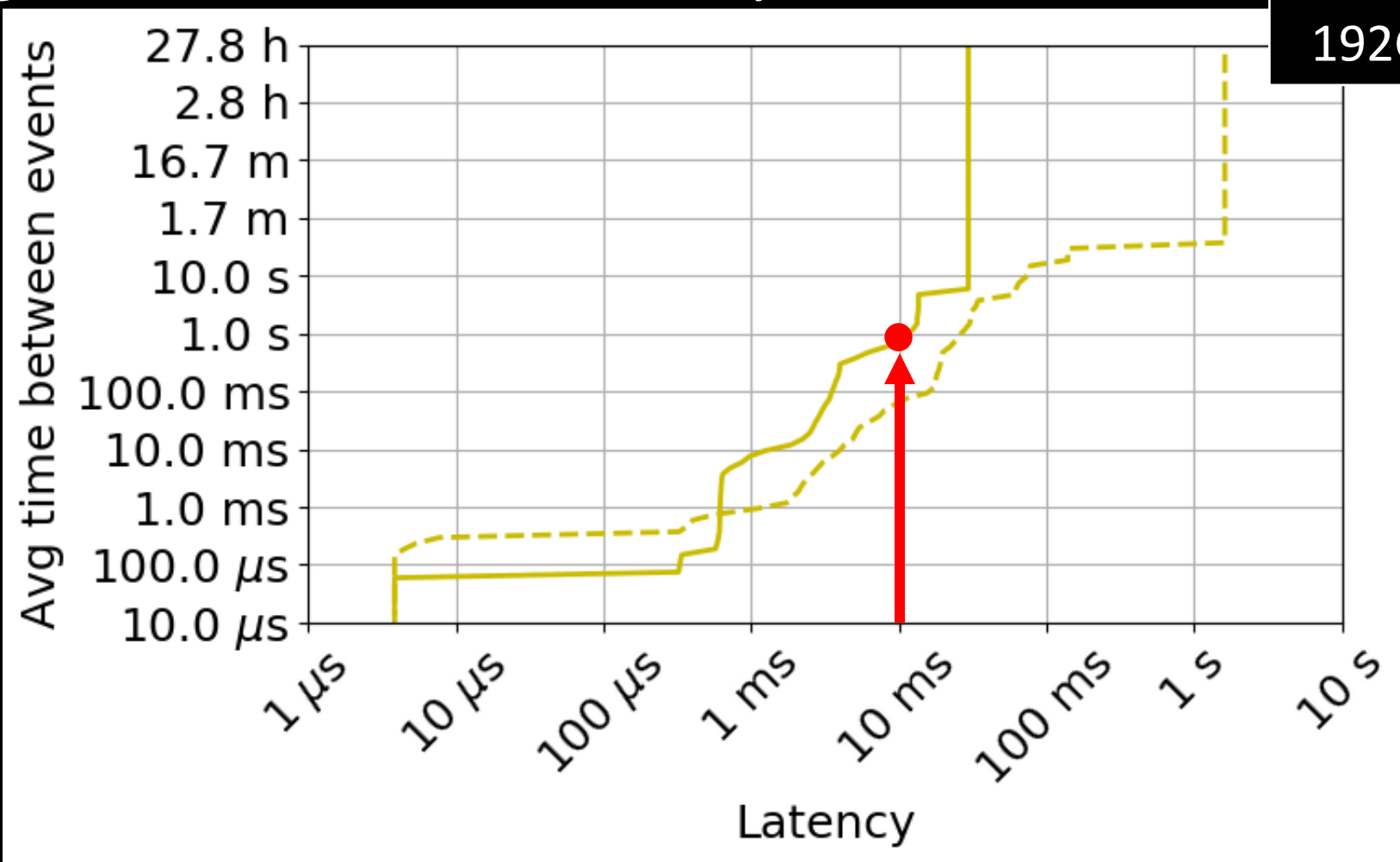
memcached
192GB peak mem



— Linux
- - - Linux,fragmented

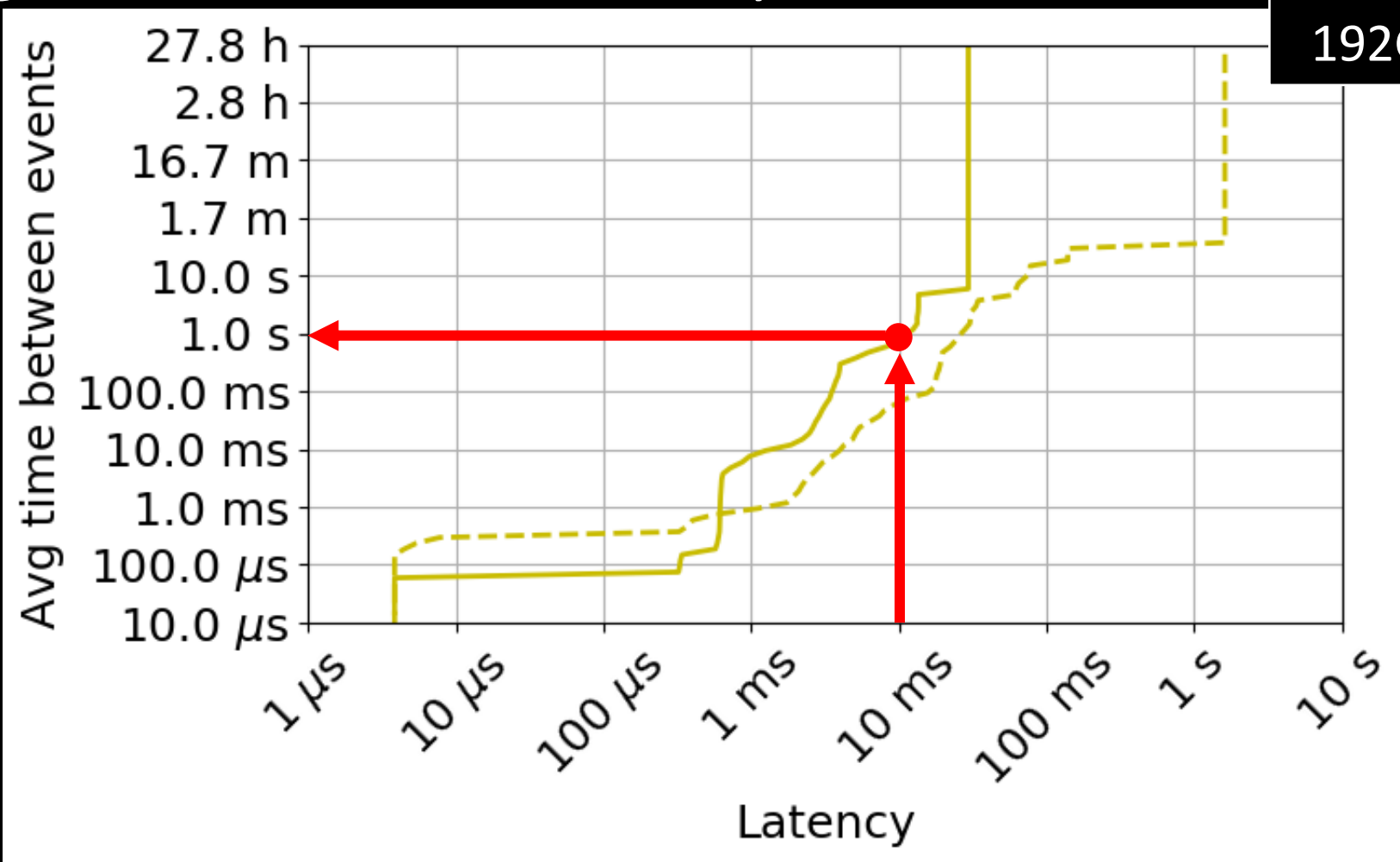
Soft Page Fault Tail Latency

memcached
192GB peak mem



Soft Page Fault Tail Latency

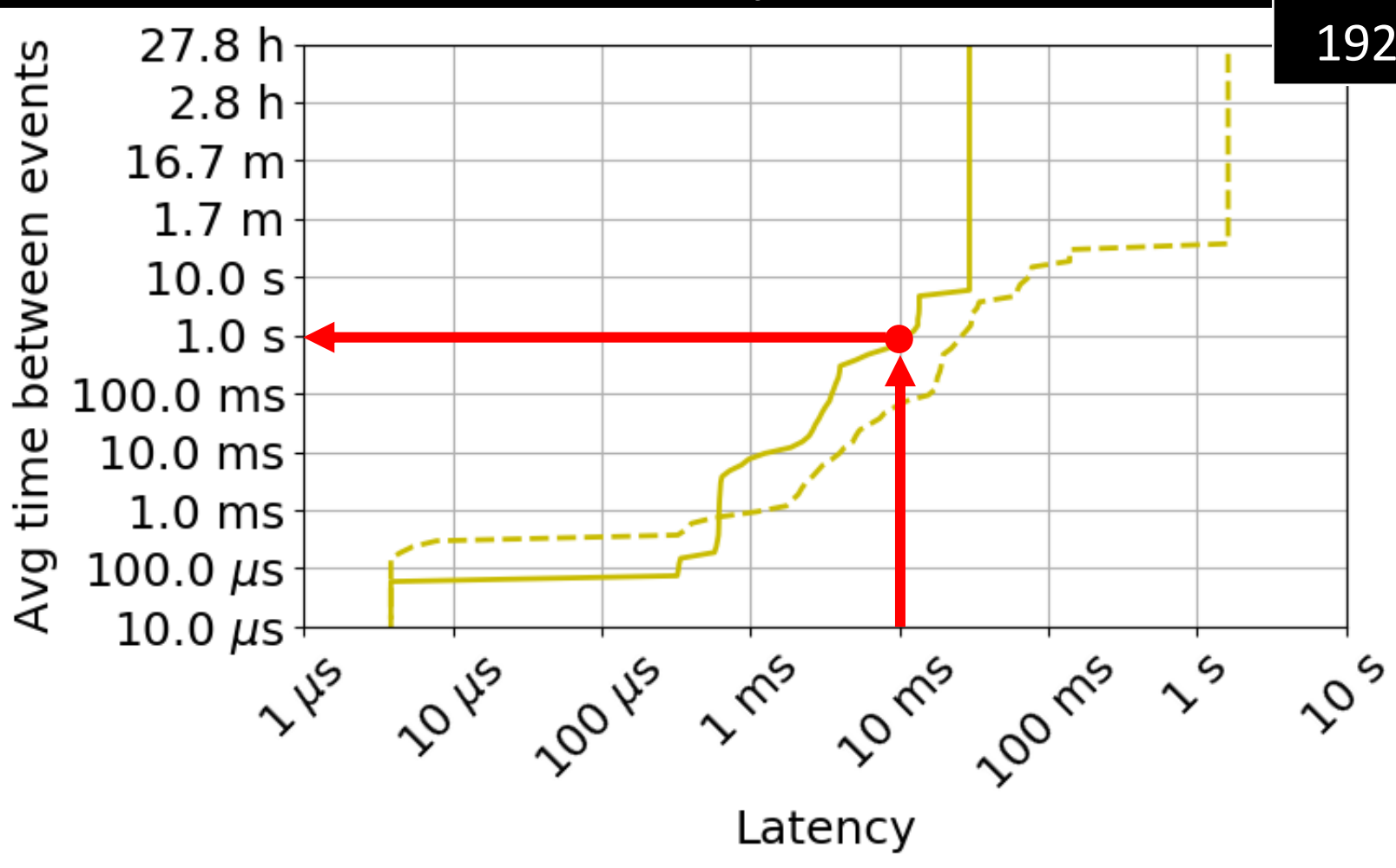
memcached
192GB peak mem



Soft Page Fault Tail Latency

memcached
192GB peak mem

Upper Left
is better

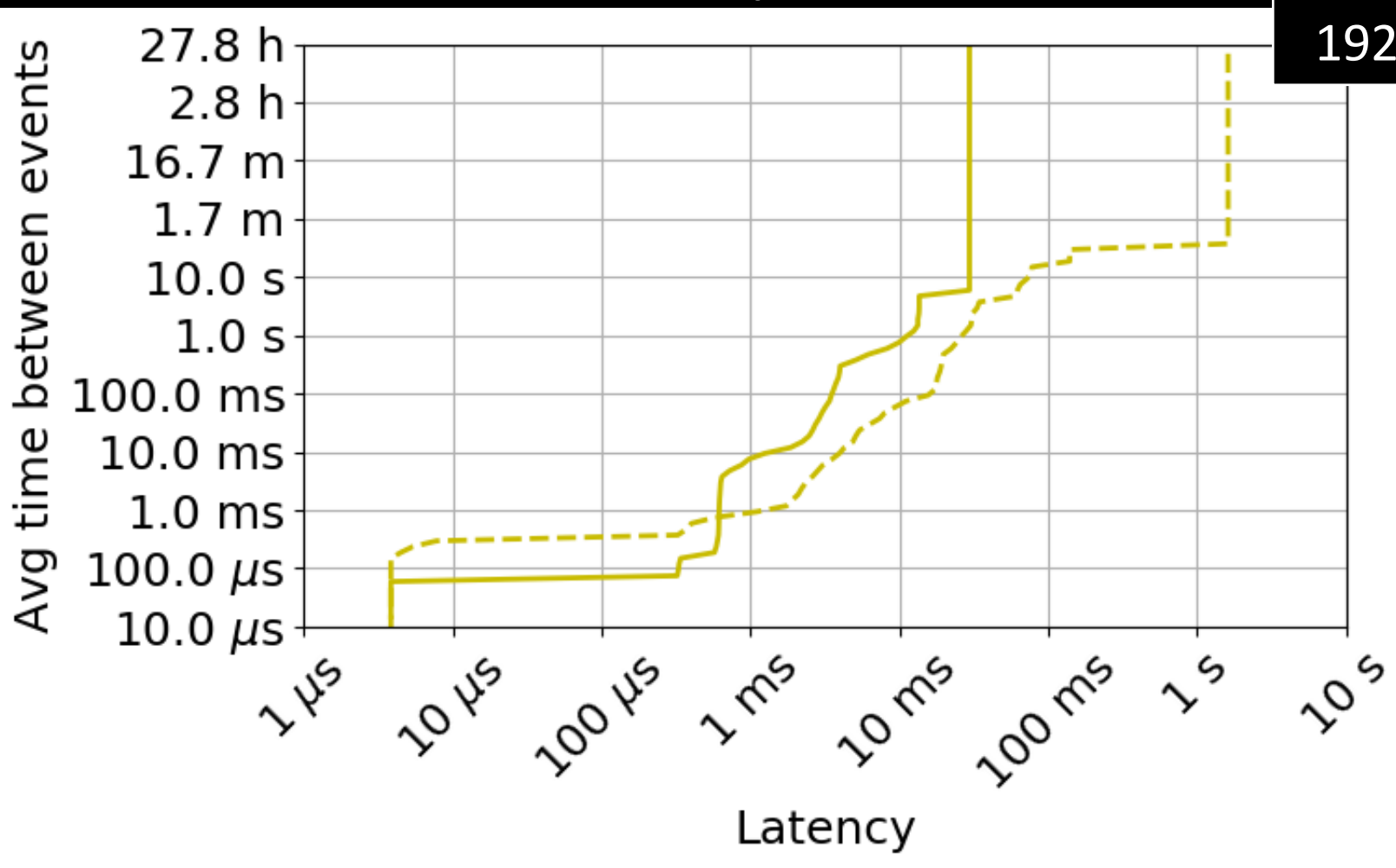


- Linux
- - - Linux,fragmented

Soft Page Fault Tail Latency

memcached
192GB peak mem

Upper Left
is better

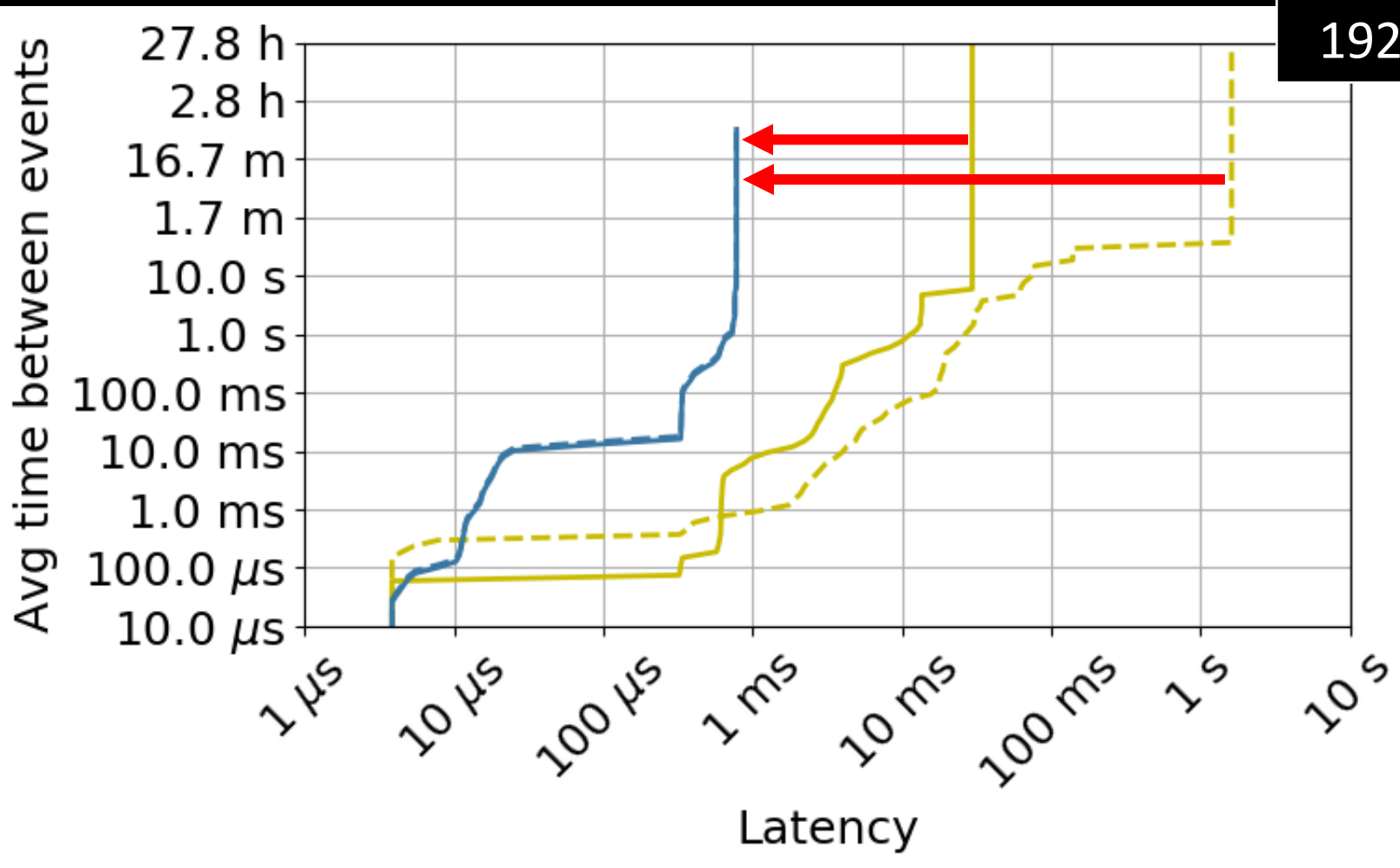


— Linux
- - - Linux, fragmented

Soft Page Fault Tail Latency

memcached
192GB peak mem

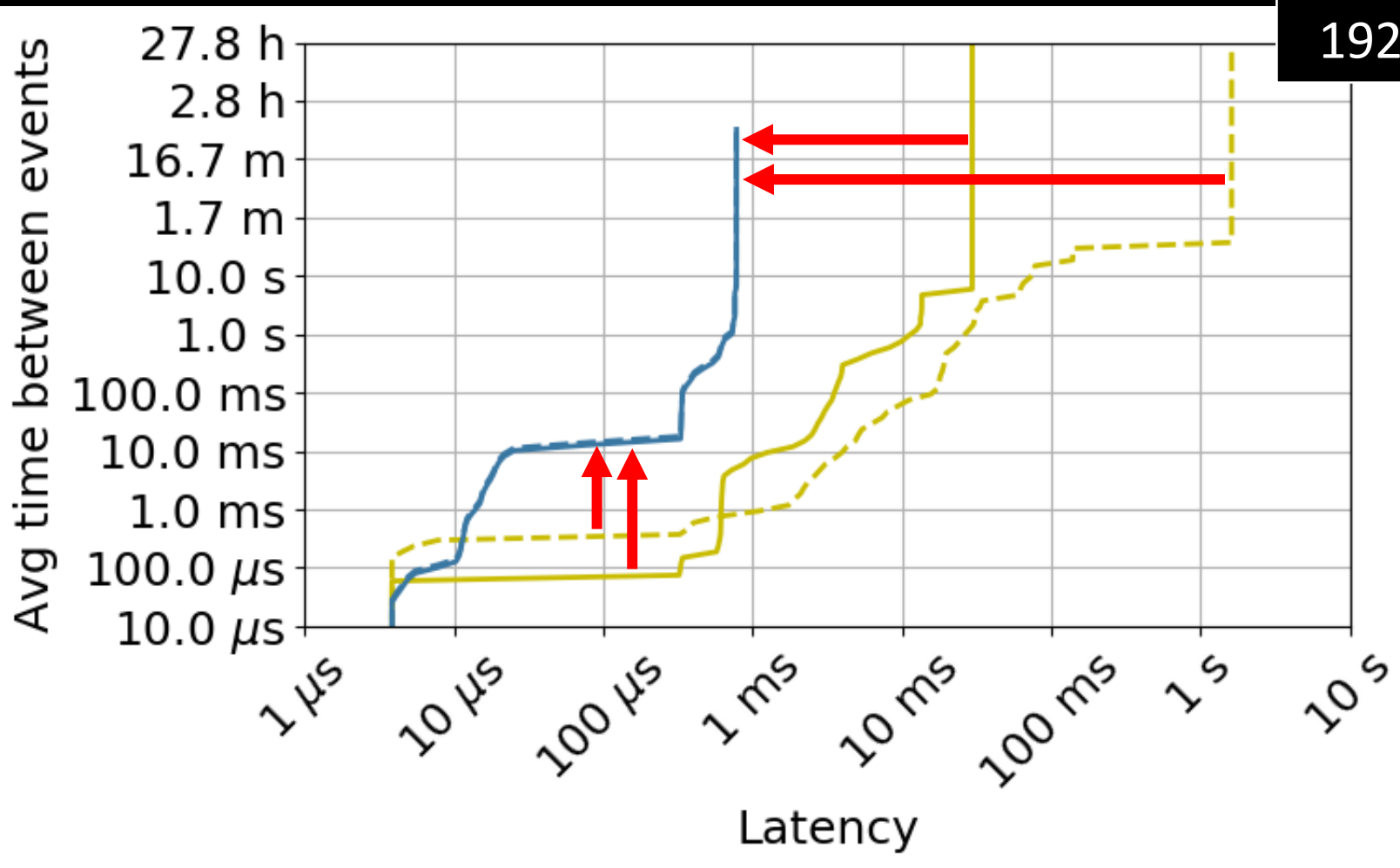
Upper Left
is better



Soft Page Fault Tail Latency

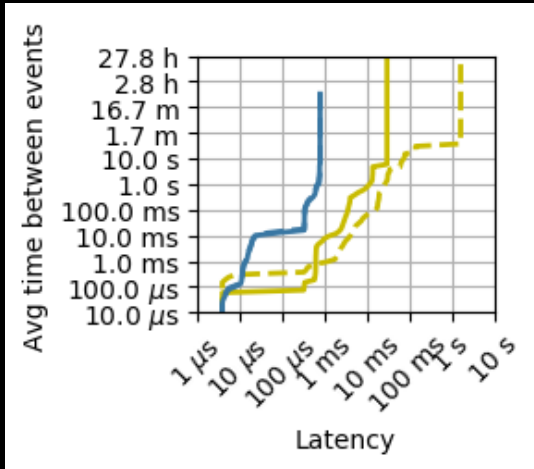
memcached
192GB peak mem

Upper Left
is better

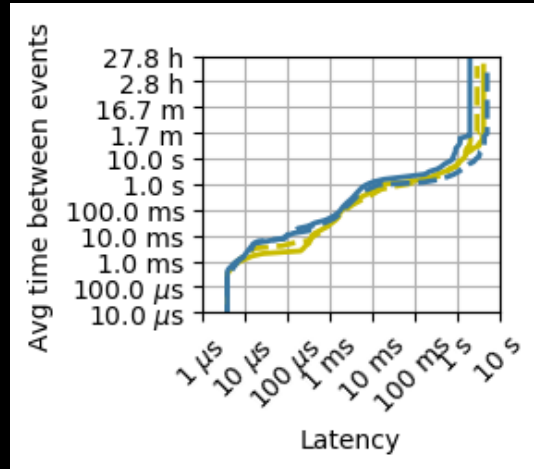


Soft Page Fault Tail Latency

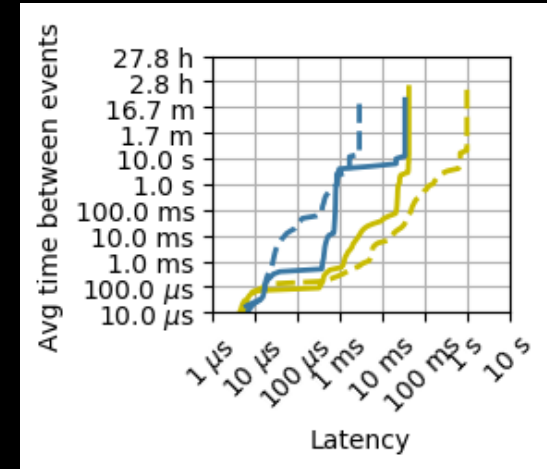
Upper Left
is better



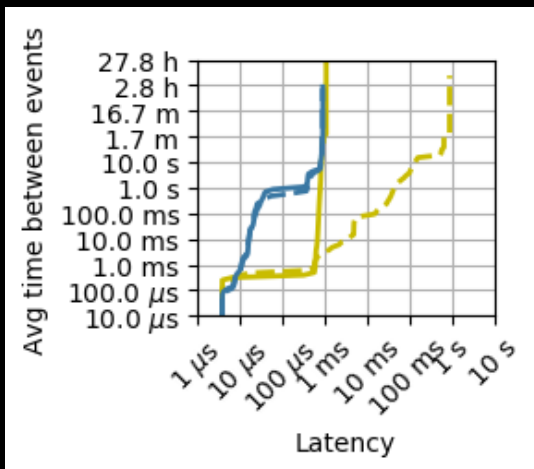
memcached



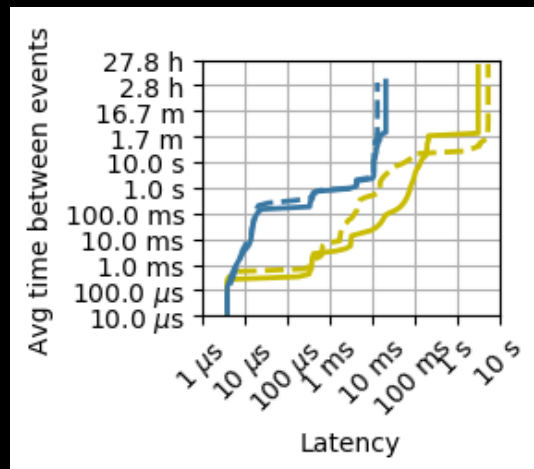
mongodb



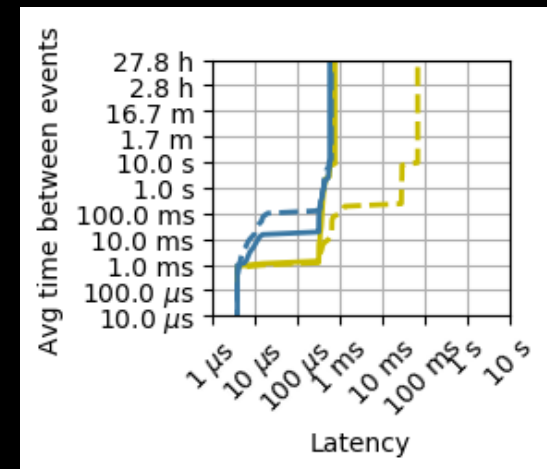
dc-mix



xz



canneal

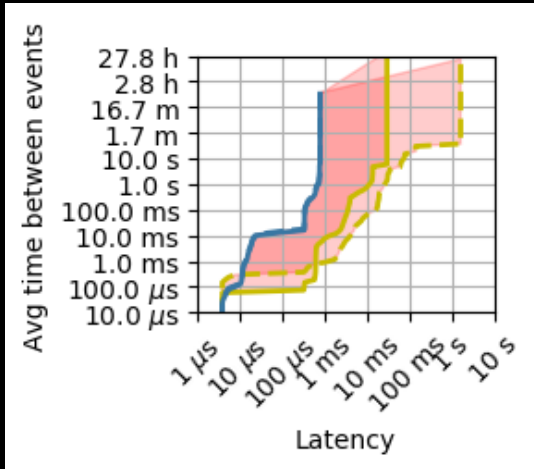


mcf

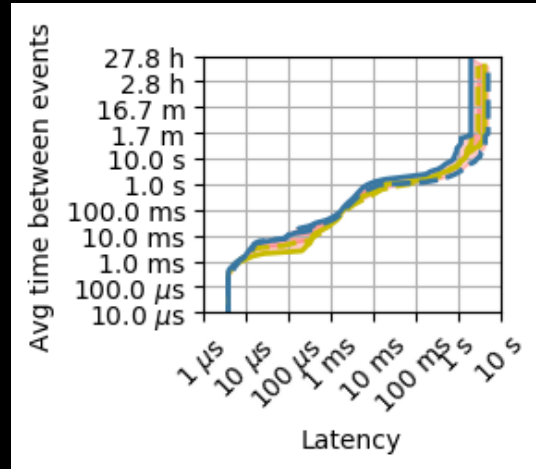


Soft Page Fault Tail Latency

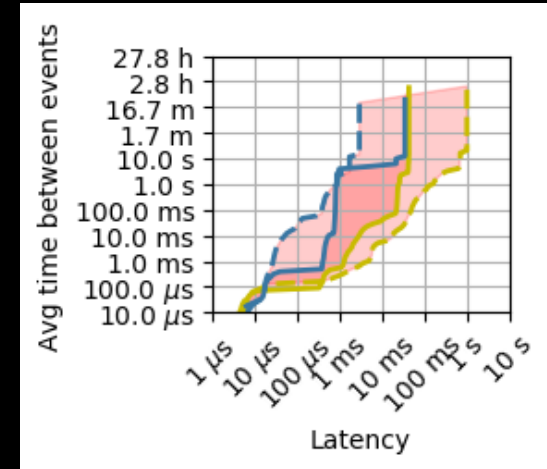
Upper Left
is better



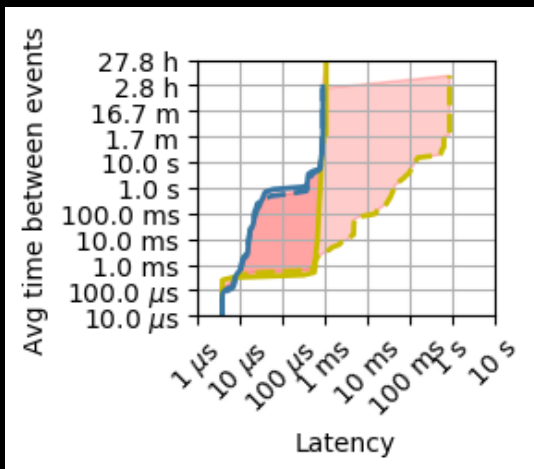
memcached



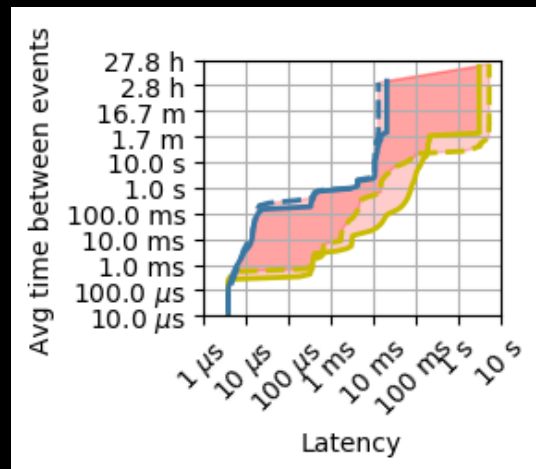
mongodb



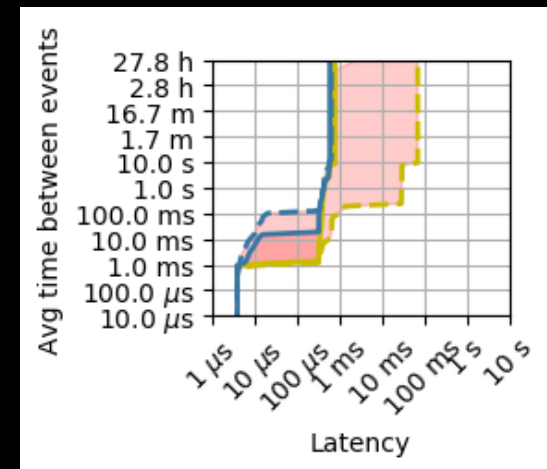
dc-mix



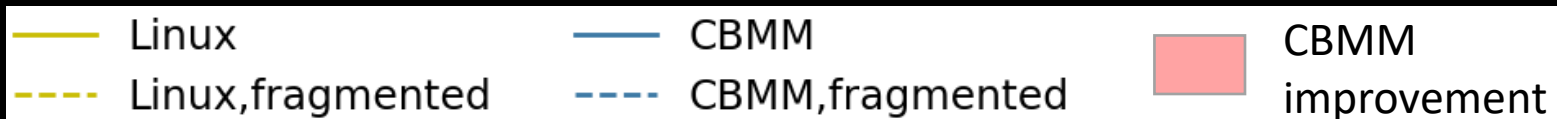
xz



canneal




mcf



Performance (Normalized Workload Runtime)

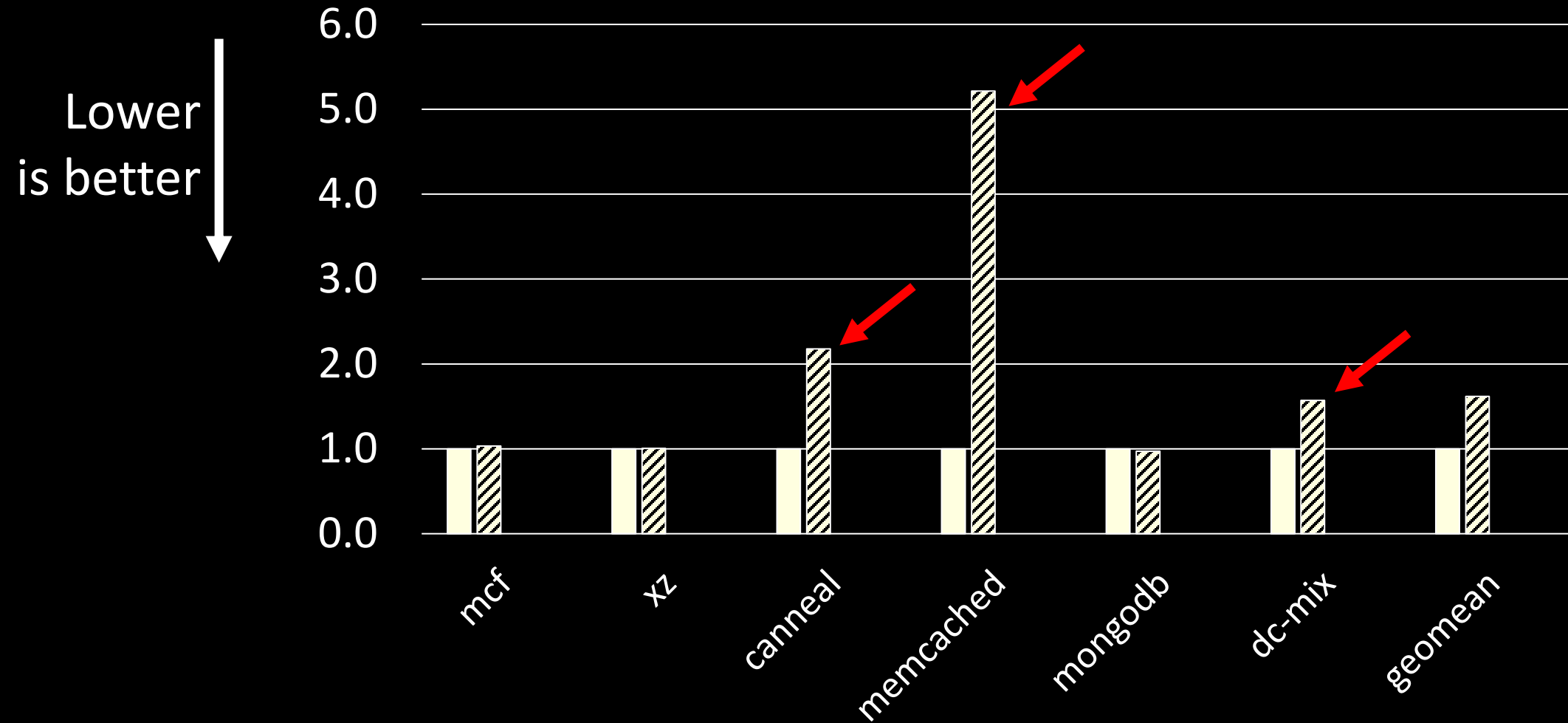
Lower
is better



Performance (Normalized Workload Runtime)

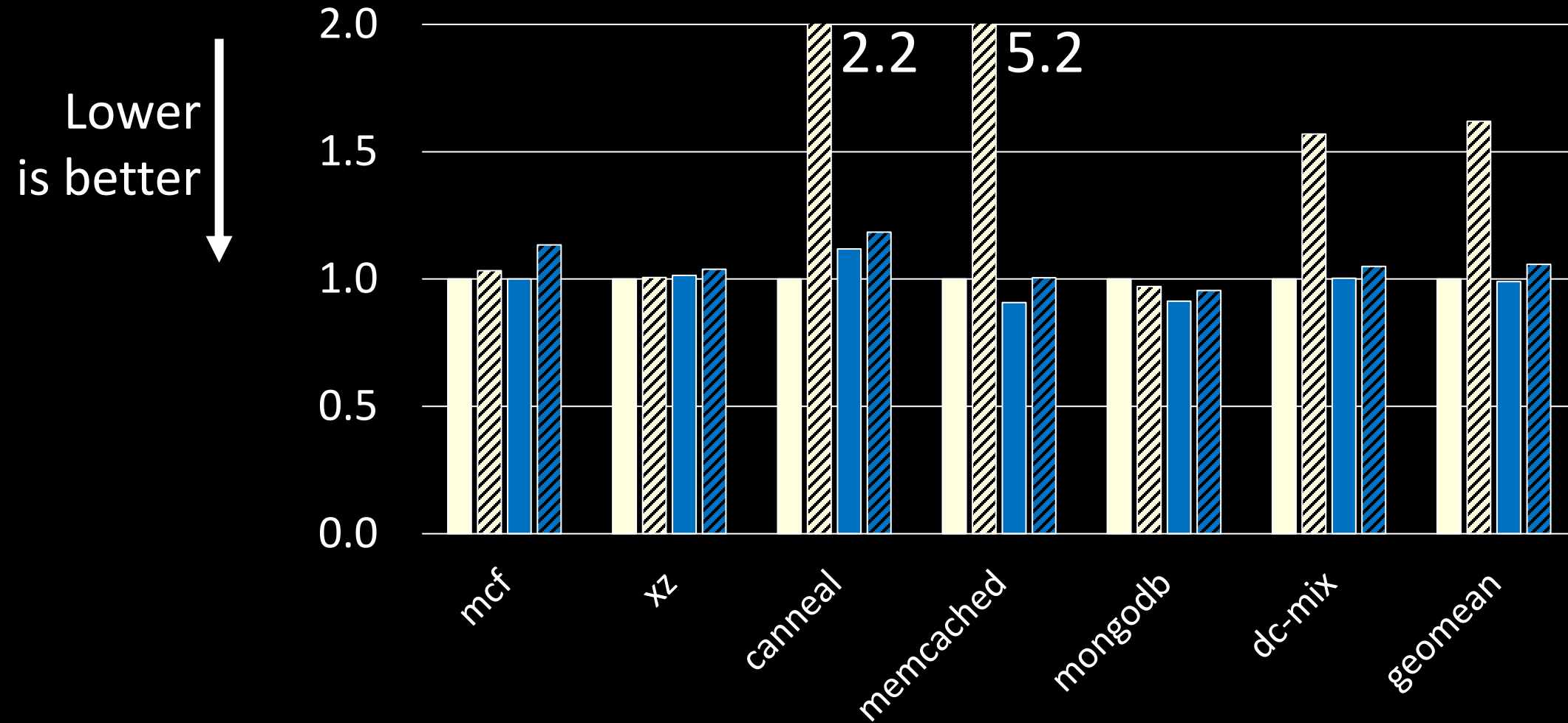
■ Linux, unfragmented

▨ Linux, fragmented



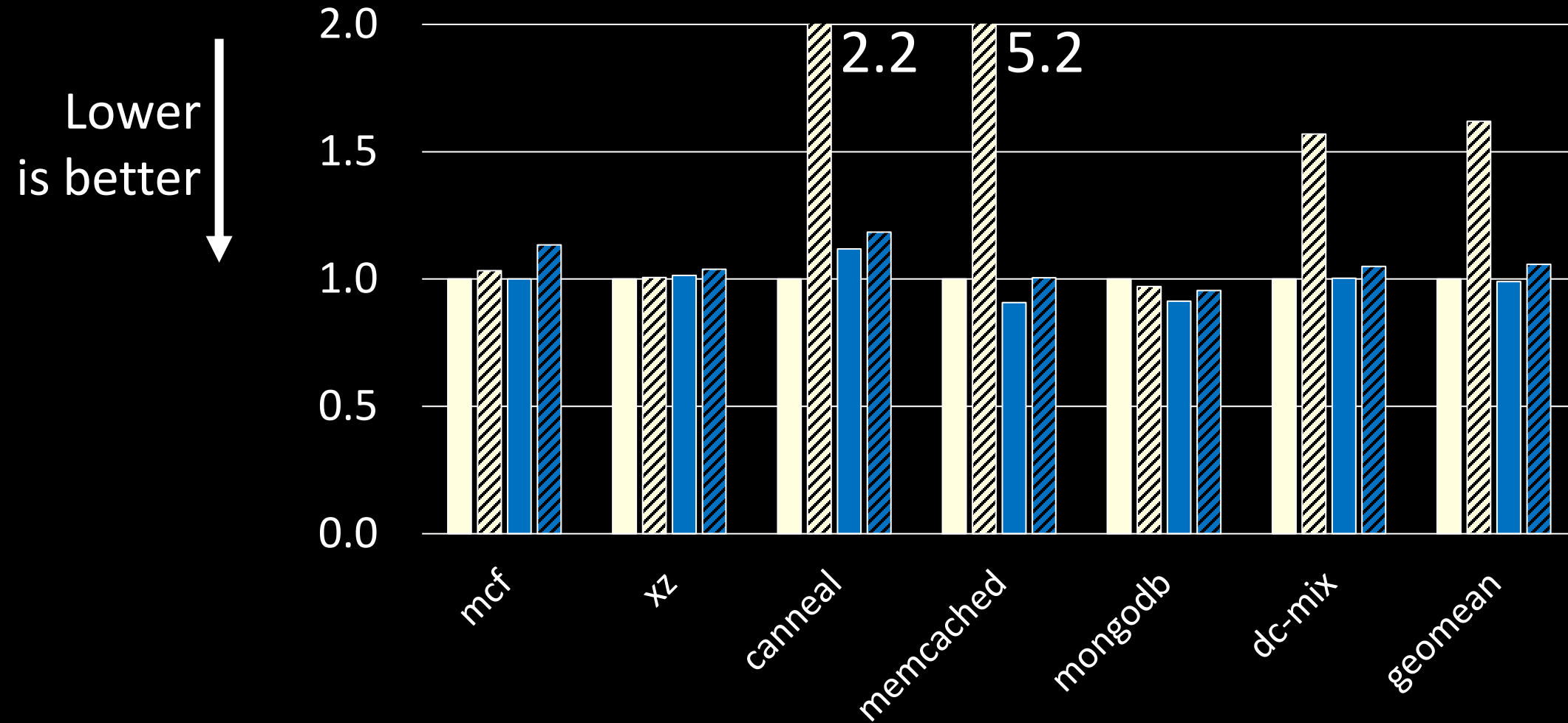
Performance (Normalized Workload Runtime)

■ Linux, unfragmented ▨ Linux, fragmented
■ CBMM, unfragmented ▨ CBMM, fragmented



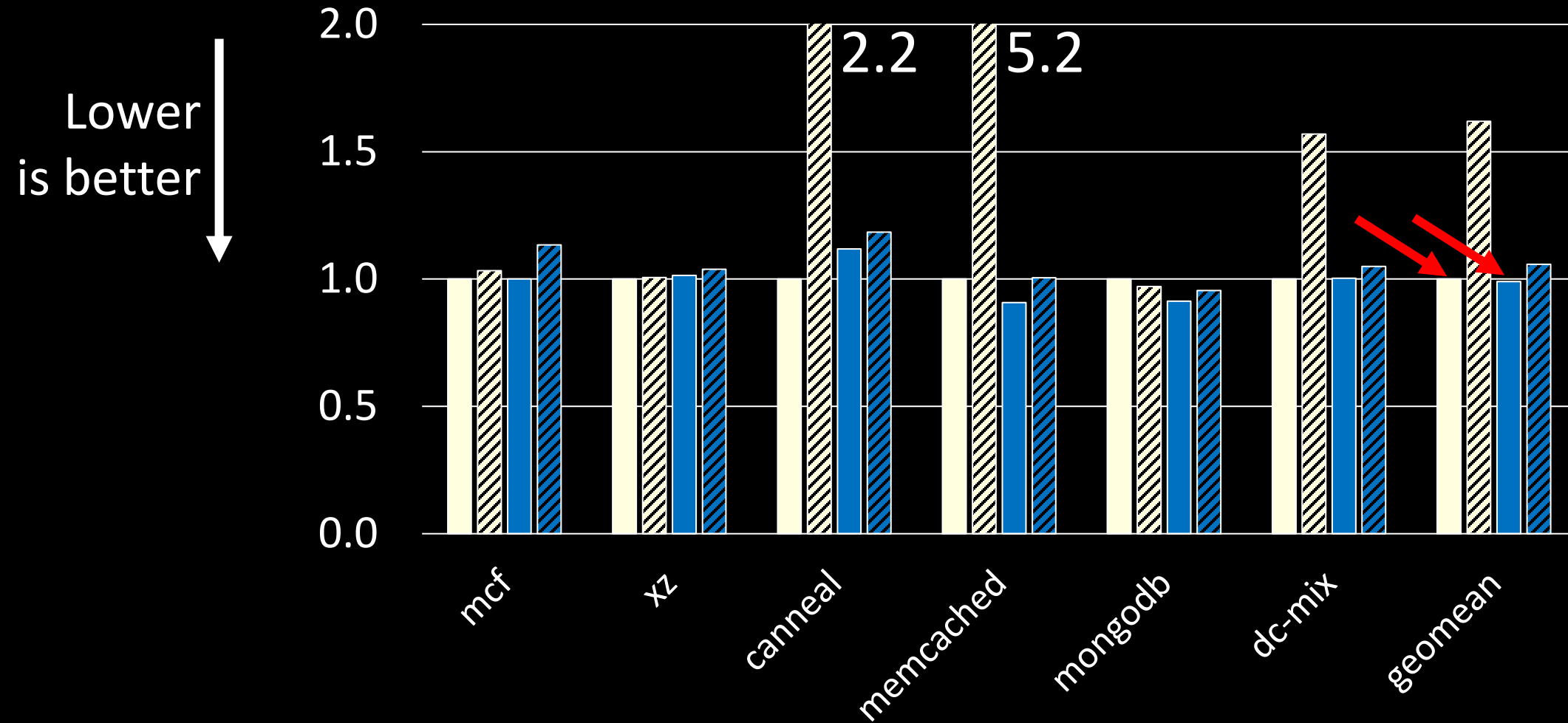
Performance (Normalized Workload Runtime)

■ Linux, unfragmented ▨ Linux, fragmented
■ CBMM, unfragmented ▨ CBMM, fragmented



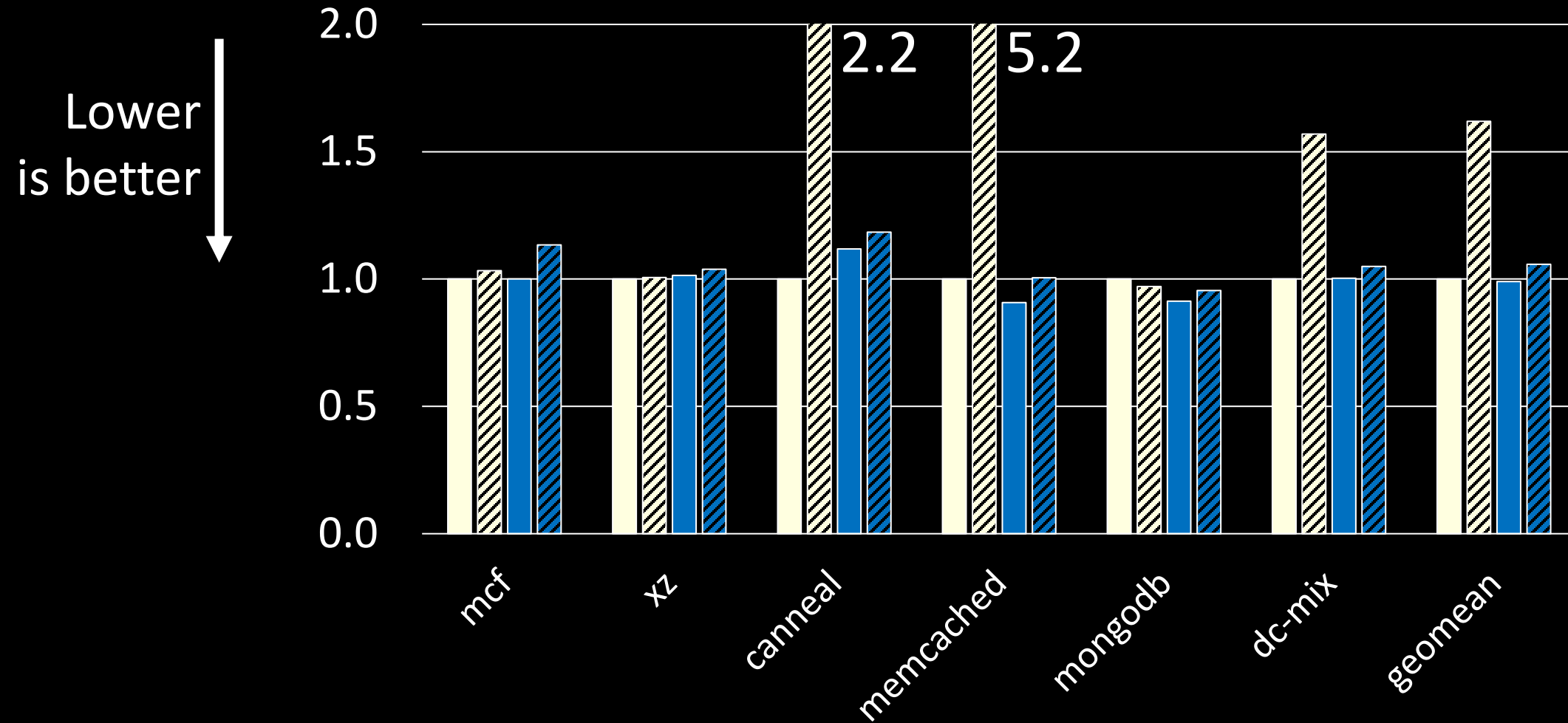
Performance (Normalized Workload Runtime)

■ Linux, unfragmented ▨ Linux, fragmented
■ CBMM, unfragmented ▨ CBMM, fragmented



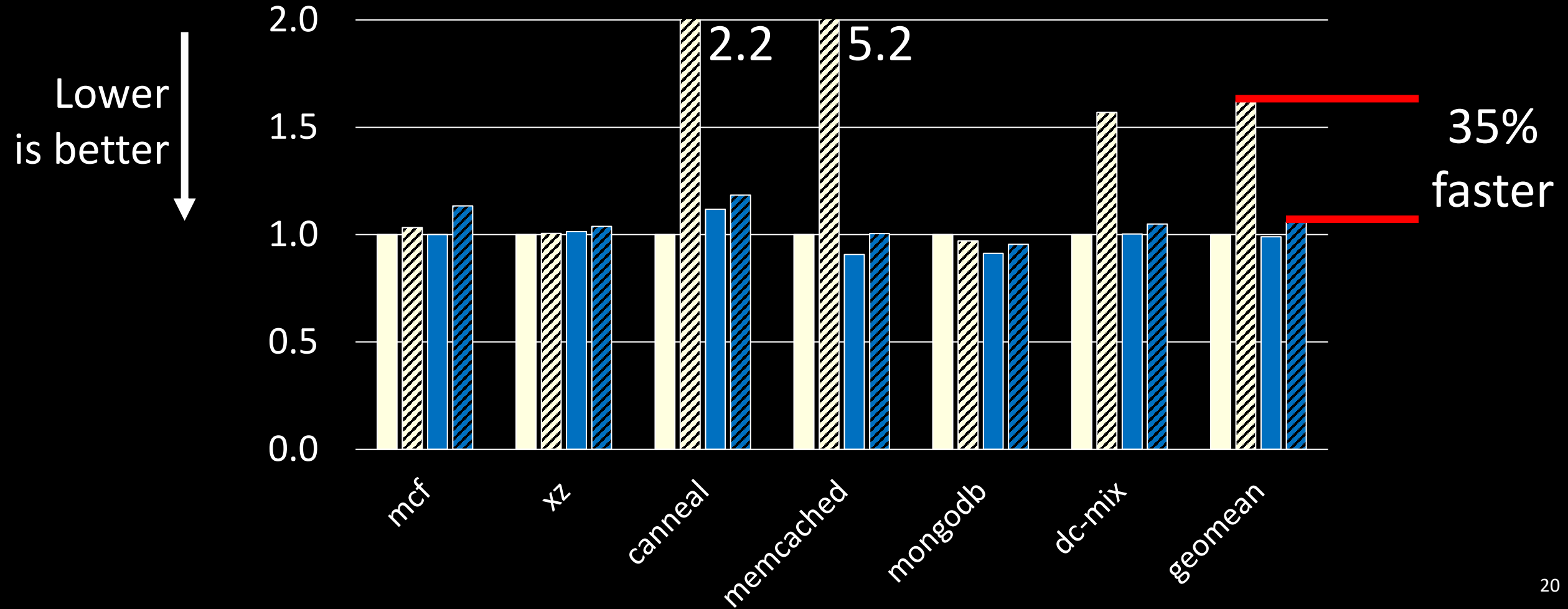
Performance (Normalized Workload Runtime)

■ Linux, unfragmented ▨ Linux, fragmented
■ CBMM, unfragmented ▨ CBMM, fragmented



Performance (Normalized Workload Runtime)

■ Linux, unfragmented ▨ Linux, fragmented
■ CBMM, unfragmented ▨ CBMM, fragmented



Performance (Normalized Workload Runtime)

■ Linux, unfragmented ▨ Linux, fragmented
■ CBMM, unfragmented ▨ CBMM, fragmented

2.0

2.2

5.2

Lower

is bet

25%

ster

Improves tail latency without regressing performance

0.5

0.0

mcf

xz

canneal

memcached

mongodb

dc-mix

geomean

Conclusion

Conclusion

Challenges

CBMM Design

Conclusion

Challenges

Scattered implementation

Cost-unaware policies

Low-quality information

CBMM Design

Centralized policy implementation

Models: userspace cost < benefit

Profiles augment kernel info

Conclusion

Challenges

Scattered implementation

Cost-unaware policies

Low-quality information

CBMM Design

Centralized policy implementation

Models: userspace cost < benefit

Profiles augment kernel info

Results

- Improve soft page fault tail latency, often by 2-3 orders of magnitude
- Competitive performance; 35% faster on fragmented systems on average

Conclusion

Challenges

Scattered implementation
Cost-unaware policies
Low-quality information

Results

- Improve soft page fault tail latency, often by 2-3 orders of magnitude
- Competitive performance; 35% faster on fragmented systems on average

CBMM Design

Centralized policy implementation
Models: userspace cost < benefit
Profiles augment kernel info

Poster!



github.com/multifacet/cbmm-artifact