Superpages

Emmett Witchel
CS380L

Virtualization/Superpages faux quiz *(pick 2, 5 min)*

1. Define PTE replication? Do we still need it?
2. FreeBSD always breaks superpages on a write. What are the alternatives/tradeoffs?
3. Define reference/dirty bit emulation.
4. What is a population map?
5. Why super-pages instead of big segments?

Virtual Memory: Goals…what are they again?

- Abstraction: contiguous, isolated memory
  - Remember overlays?
- Prevent illegal operations
  - Access to others/OS memory
  - Fail fast (e.g. segv on *(NULL))
  - Prevent exploits that try to execute program data
- Sharing mechanism/IPC substrate

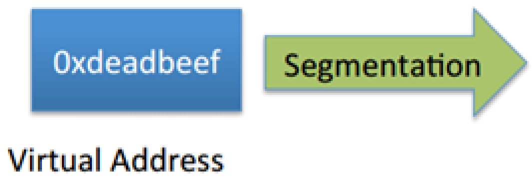
Address Translation with Segments

Address Translation with Segments

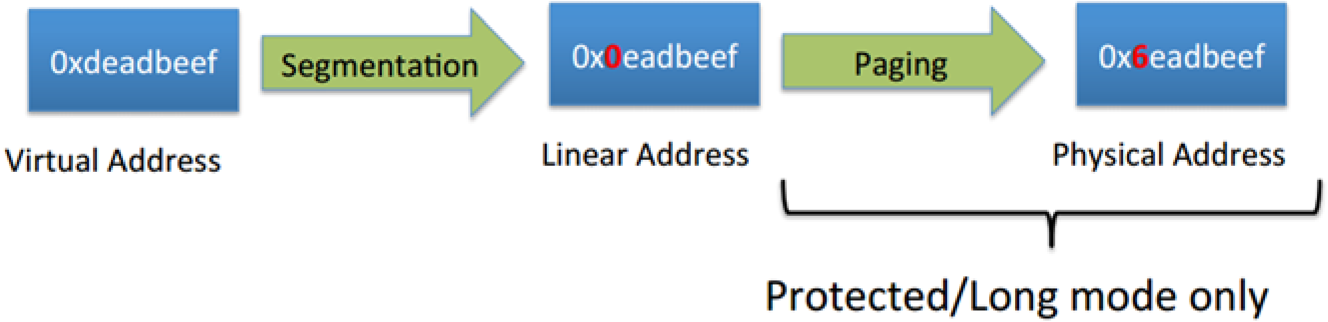
0xdeadbeef

Virtual Address

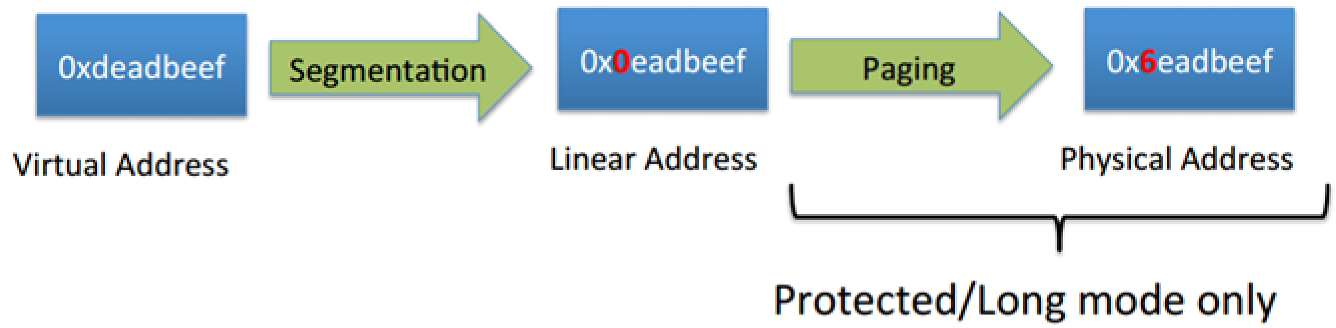
Address Translation with Segments



Address Translation with Segments

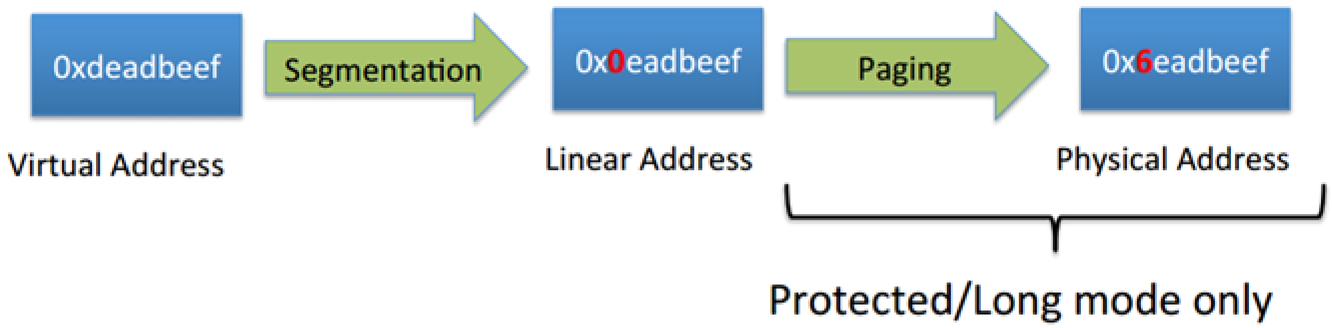


Address Translation with Segments



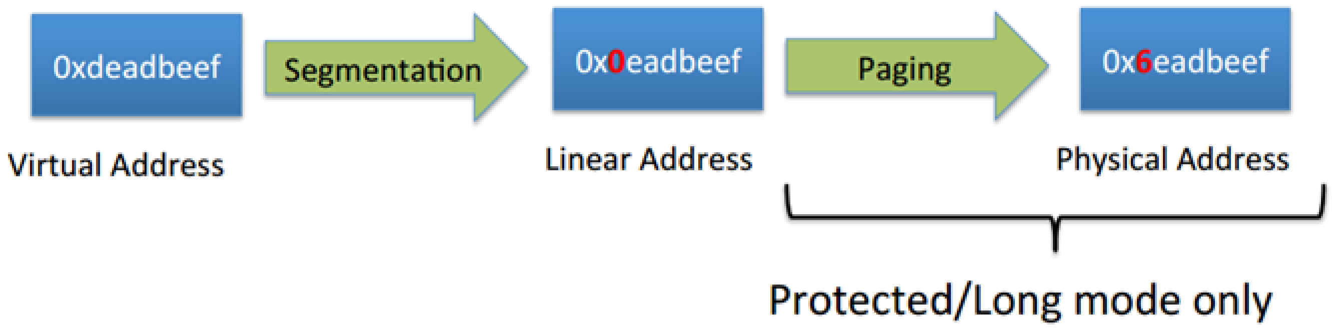
- Segmentation cannot be disabled
    - Can be made a no-op (flat mode)

Address Translation with Segments



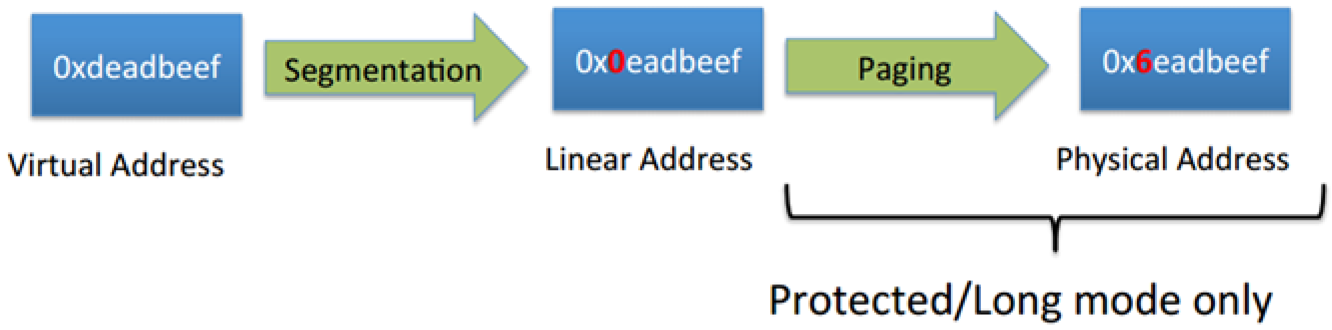
- Segmentation cannot be disabled
  - Can be made a no-op (flat mode)
- Segment = <base,len,type(code,data,stack)>
  - AS → 6 segments: regs cs, ds, ss, es, fs, gs

Address Translation with Segments



- Segmentation cannot be disabled
  - Can be made a no-op (flat mode)
- Segment = <base,len,type(code,data,stack)>
  - AS → 6 segments: regs cs, ds, ss, es, fs, gs
- Programming model: prefix refs with segment

Address Translation with Segments

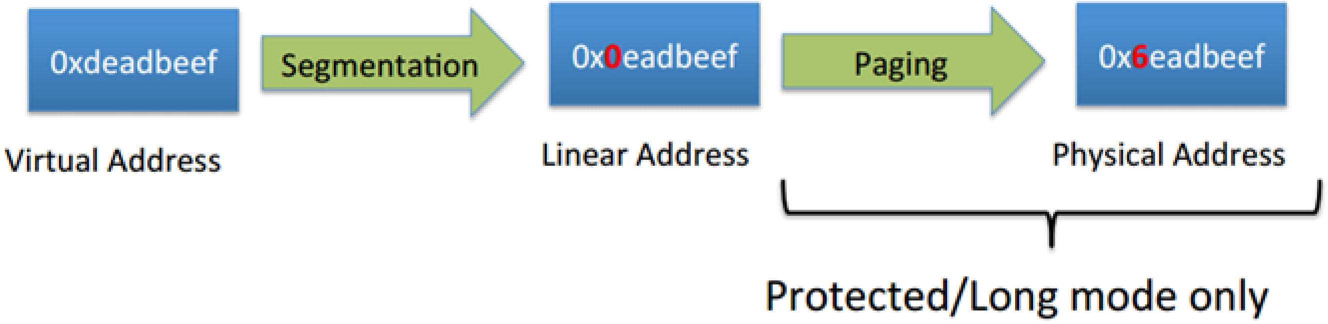


- Segmentation cannot be disabled
  - Can be made a no-op (flat mode)
- Segment = <base,len,type(code,data,stack)>
  - AS → 6 segments: regs cs, ds, ss, es, fs, gs
- Programming model: prefix refs with segment

```
// global int x = 1
int y; // stack
if (x) {
      y = 1;
      printf ("Boo");
} else
      y = 0;
```

```
ds:x = 1; // data
ss:y; // stack
if (ds:x) {
      ss:y = 1;
      cs:printf(ds:"Boo");
} else
      ss:y = 0;
```

Address Translation with Segments



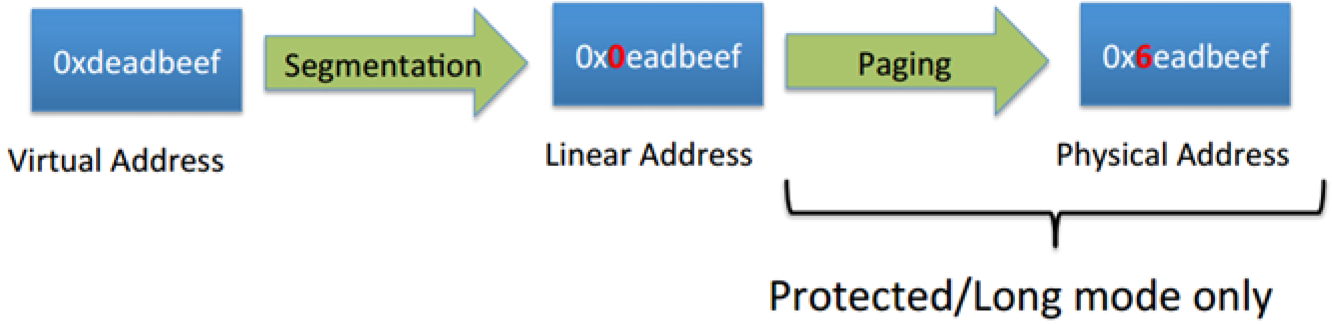
Q: How to make segmentation a no-op?

- Segmentation cannot be disabled
  - Can be made a no-op (flat mode)
- Segment = <base,len,type(code,data,stack)>
  - AS → 6 segments: regs cs, ds, ss, es, fs, gs
- Programming model: prefix refs with segment

```
// global int x = 1
int y; // stack
if (x) {
      y = 1;
      printf ("Boo");
} else
      y = 0;
```

```
ds:x = 1; // data
ss:y; // stack
if (ds:x) {
      ss:y = 1;
      cs:printf(ds:"Boo");
} else
      ss:y = 0;
```

Address Translation with Segments



Q: How to make segmentation a no-op?

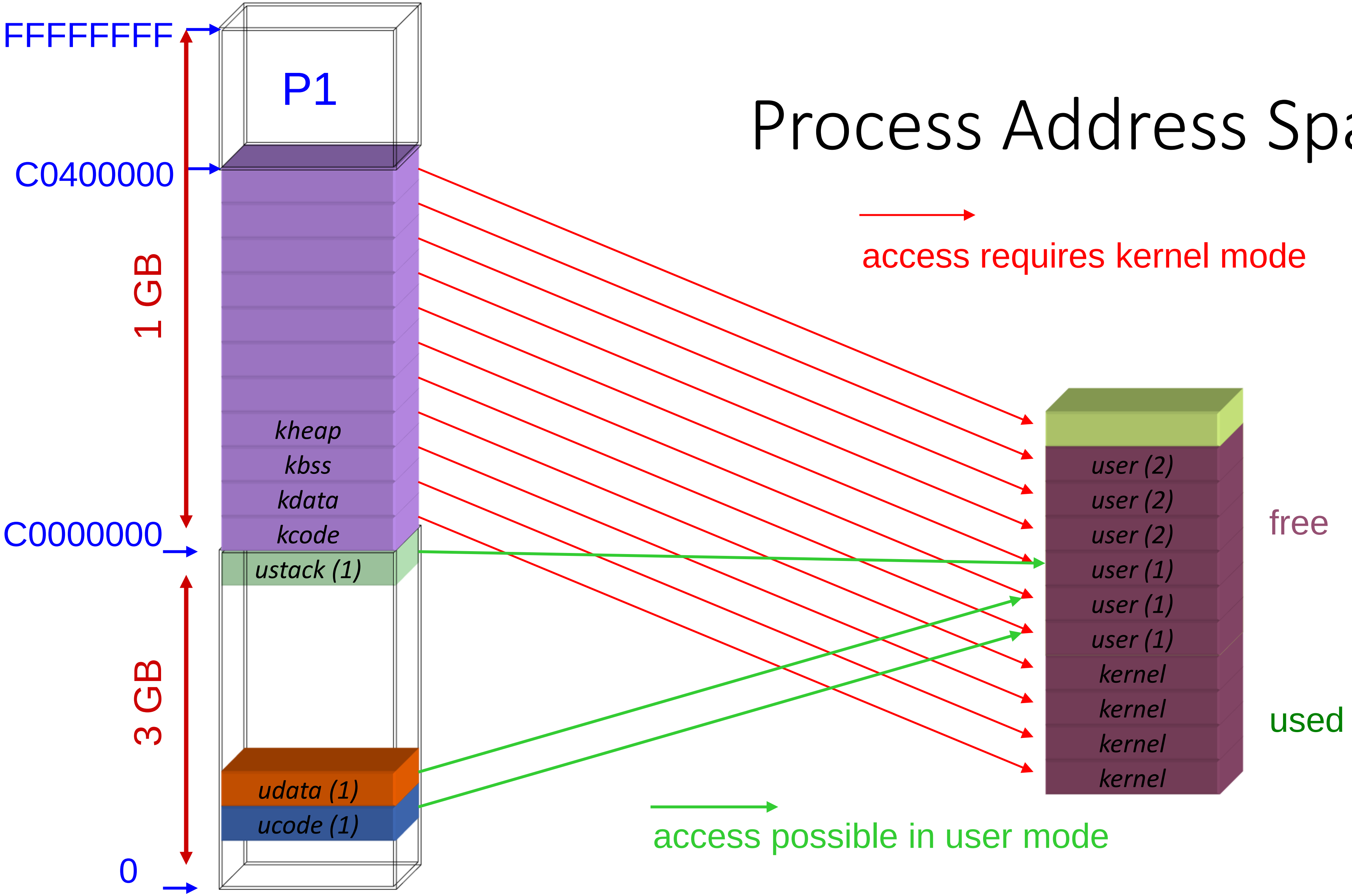
Q: How to implement TLS?

- Segmentation cannot be disabled
  - Can be made a no-op (flat mode)
- Segment = <base,len,type(code,data,stack)>
  - AS → 6 segments: regs cs, ds, ss, es, fs, gs
- Programming model: prefix refs with segment

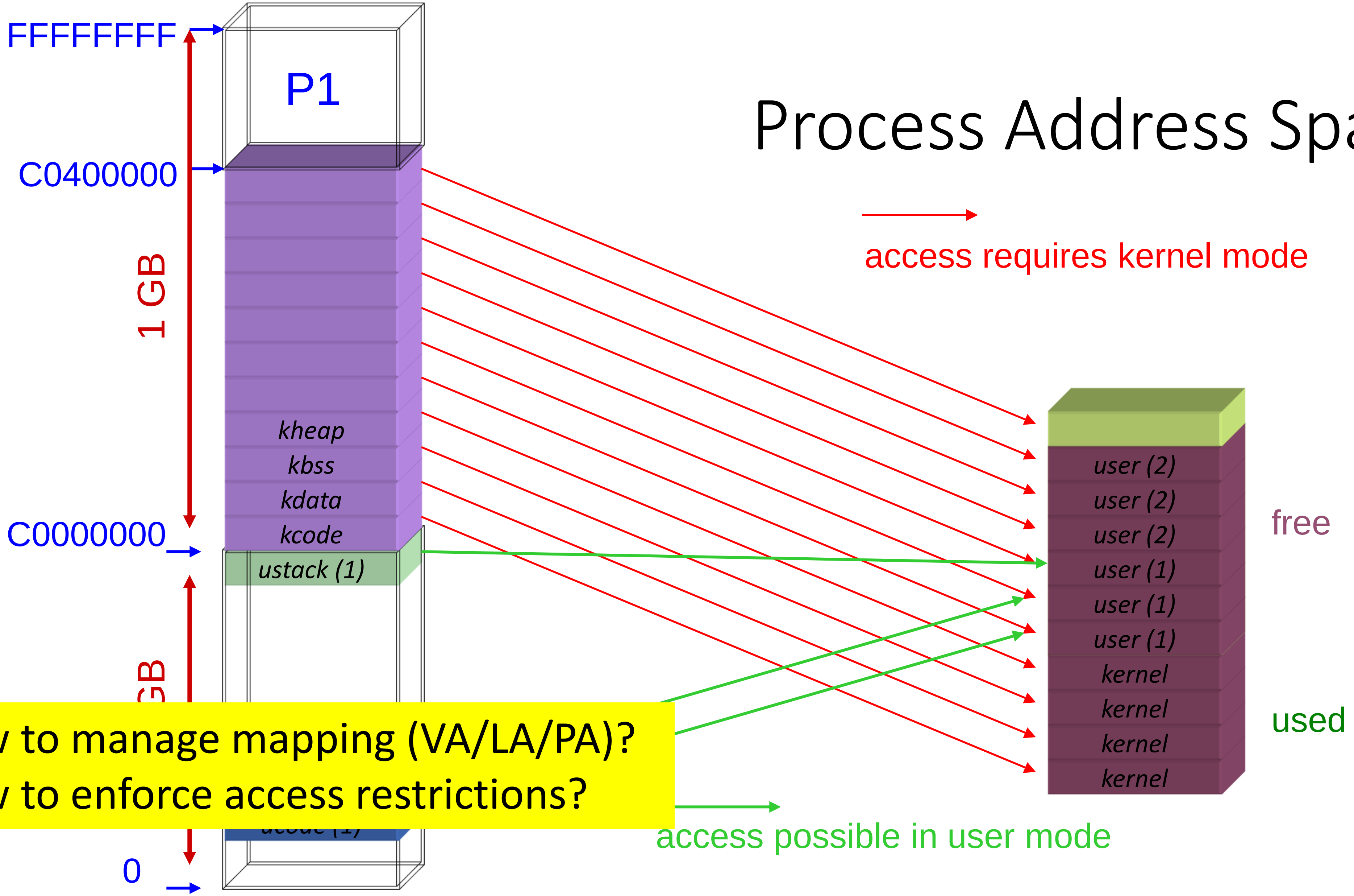
```
// global int x = 1
int y; // stack
if (x) {
      y = 1;
      printf (“Boo”);
} else
      y = 0;
```

```
ds:x = 1; // data
ss:y; // stack
if (ds:x) {
      ss:y = 1;
      cs:printf(ds:“Boo”);
} else
      ss:y = 0;
```

Process Address Space



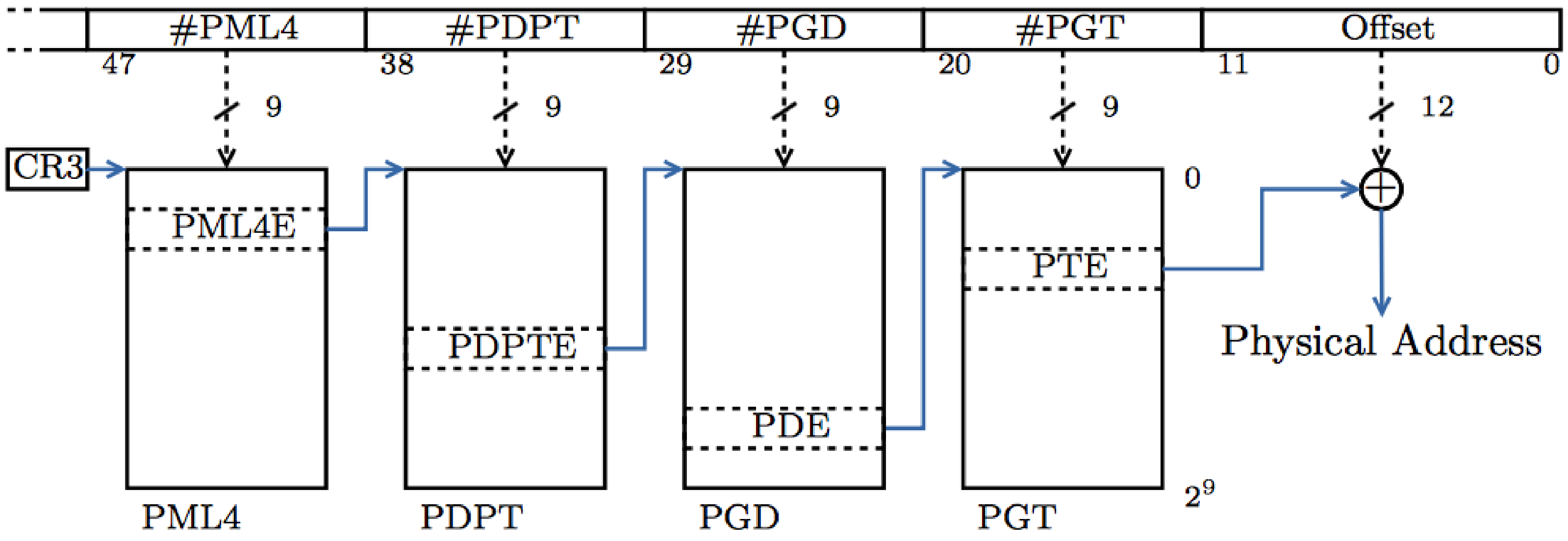
Process Address Space



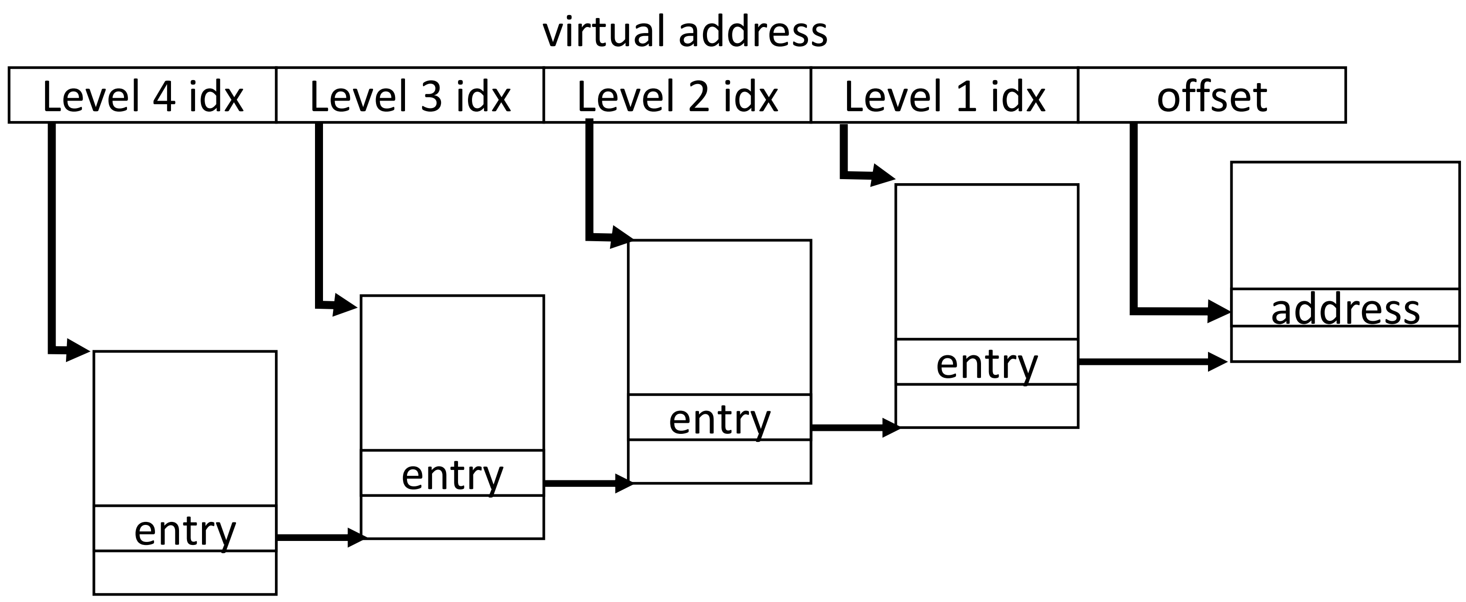
Q: How to manage mapping (VA/LA/PA)?

Q: How to enforce access restrictions?

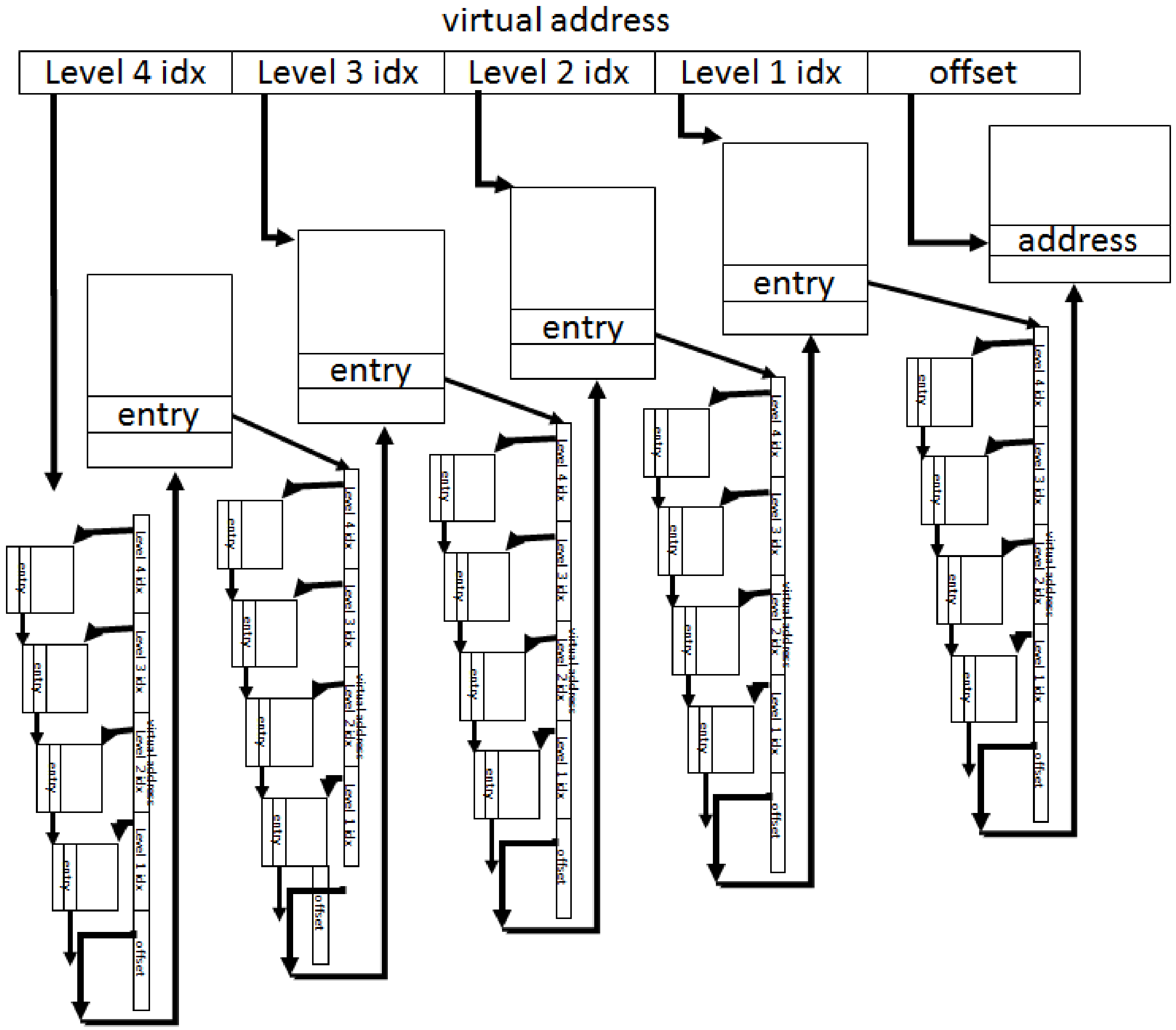
Linear → Physical Translation



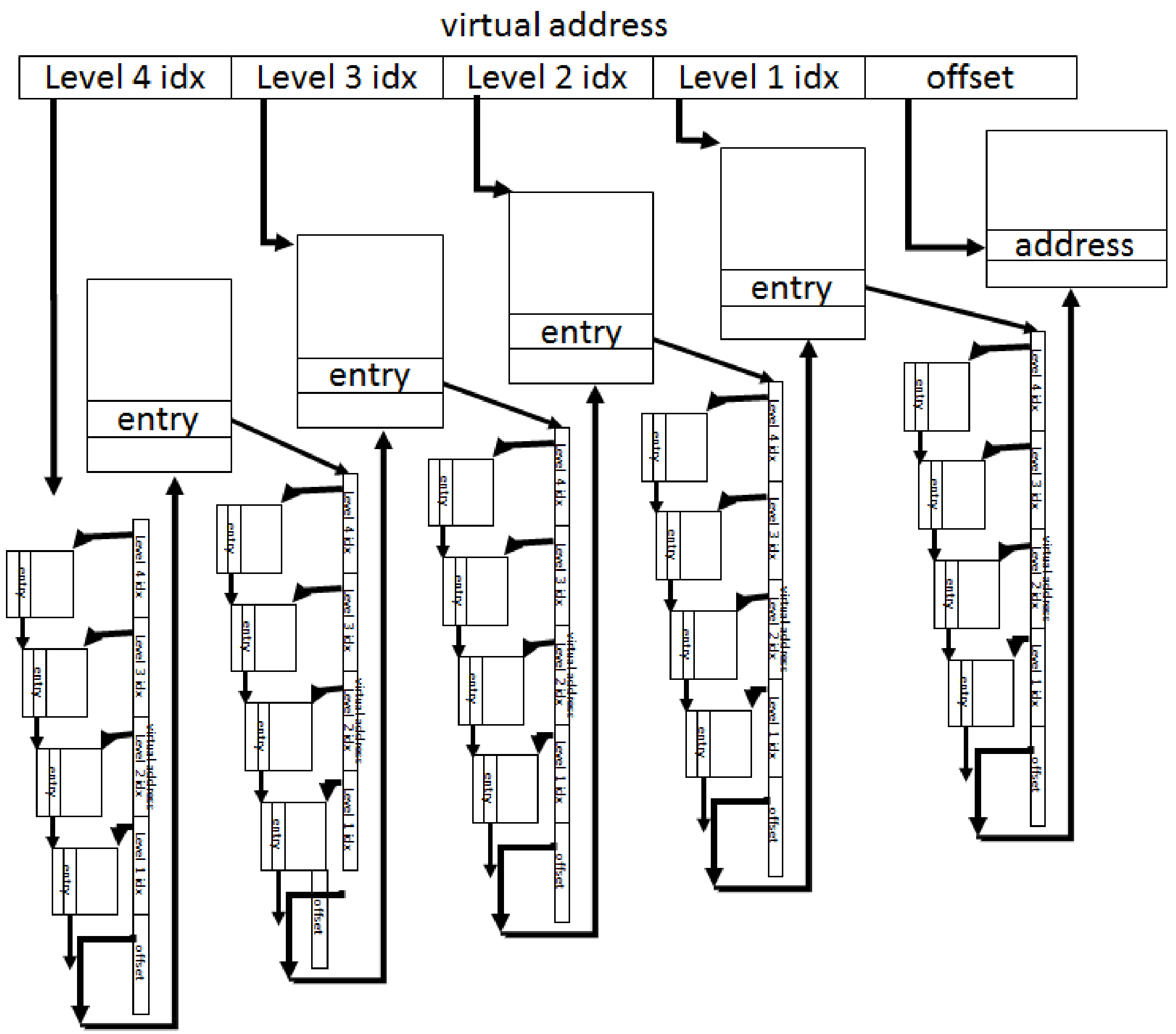
Linear → Physical Translation



Linear → Physical Translation

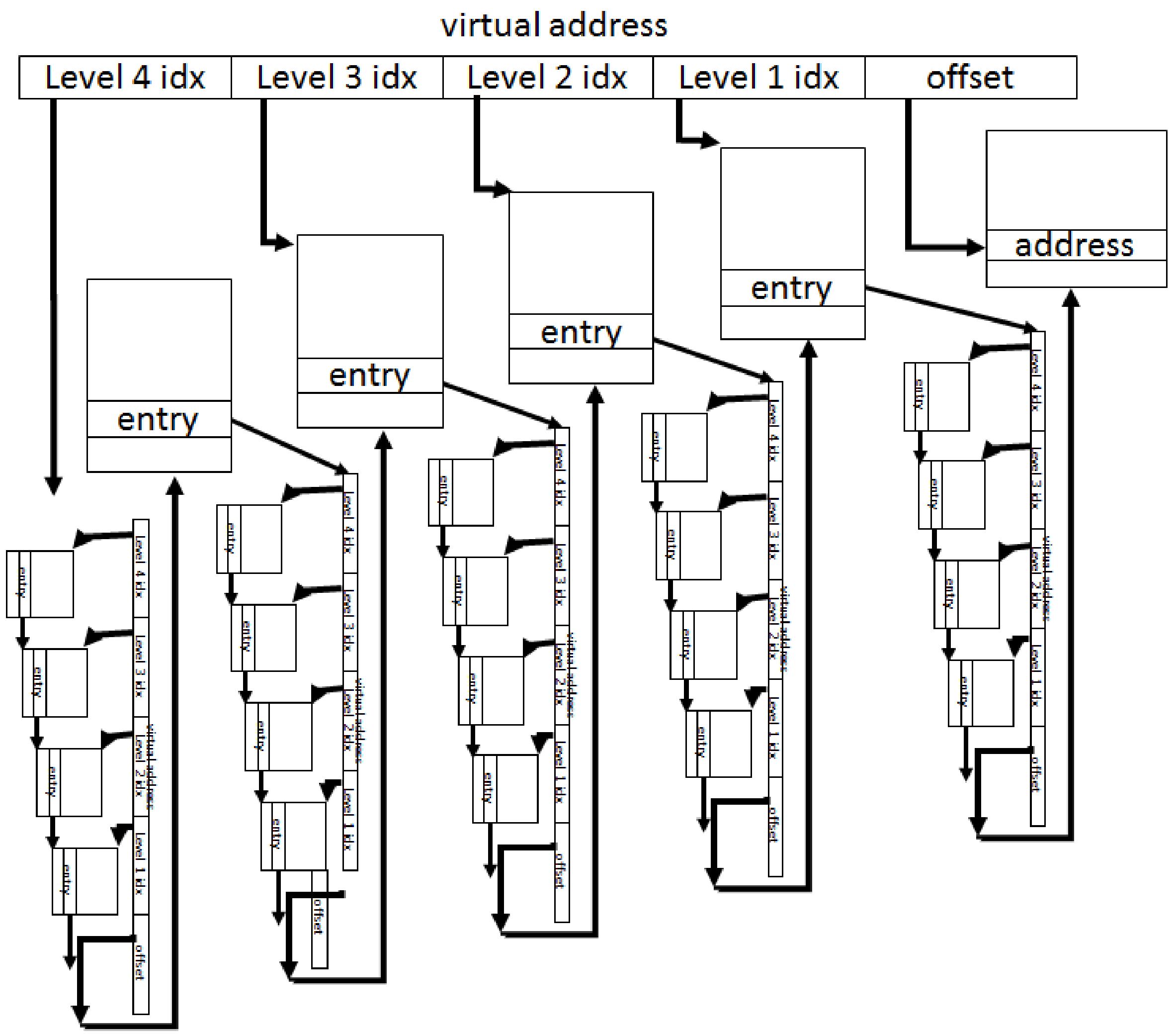


Linear → Physical Translation



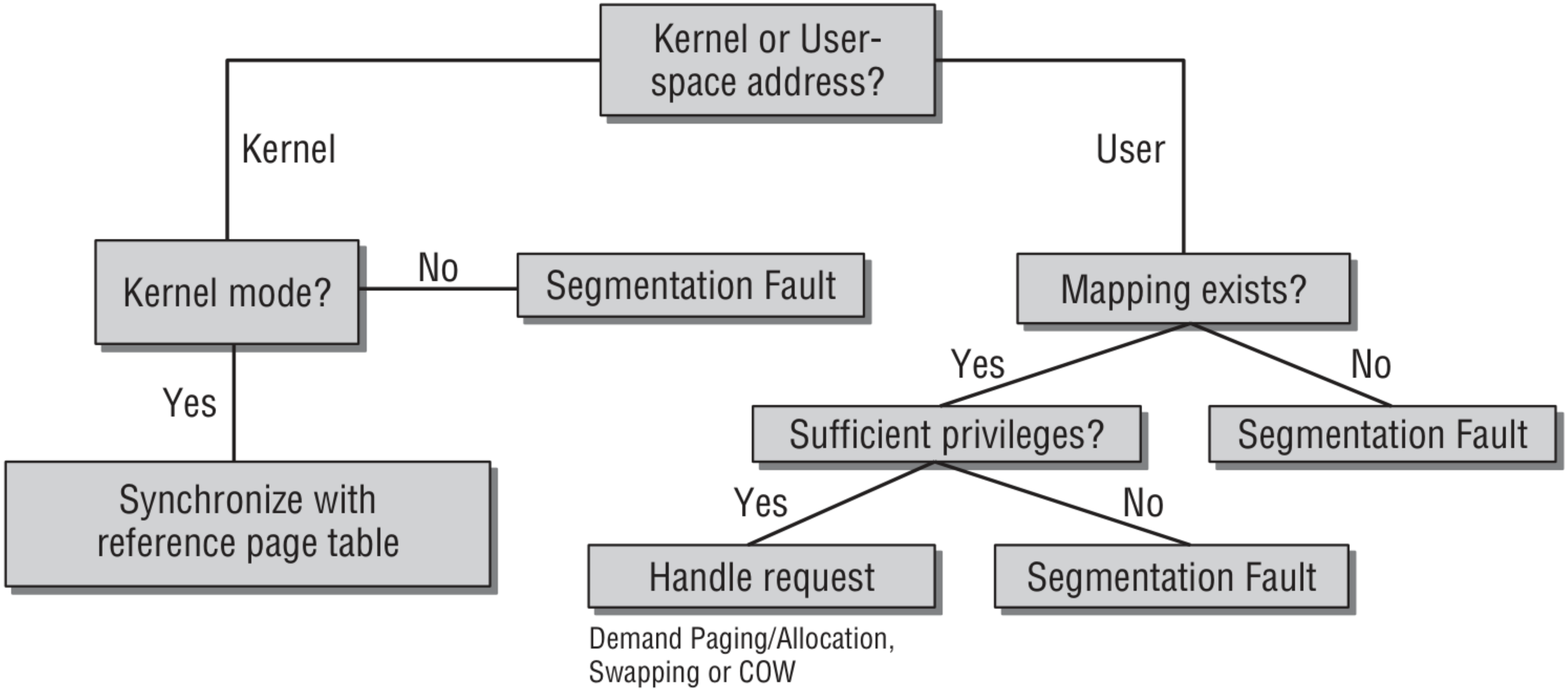
- Q: how many ops to translate?

Linear → Physical Translation

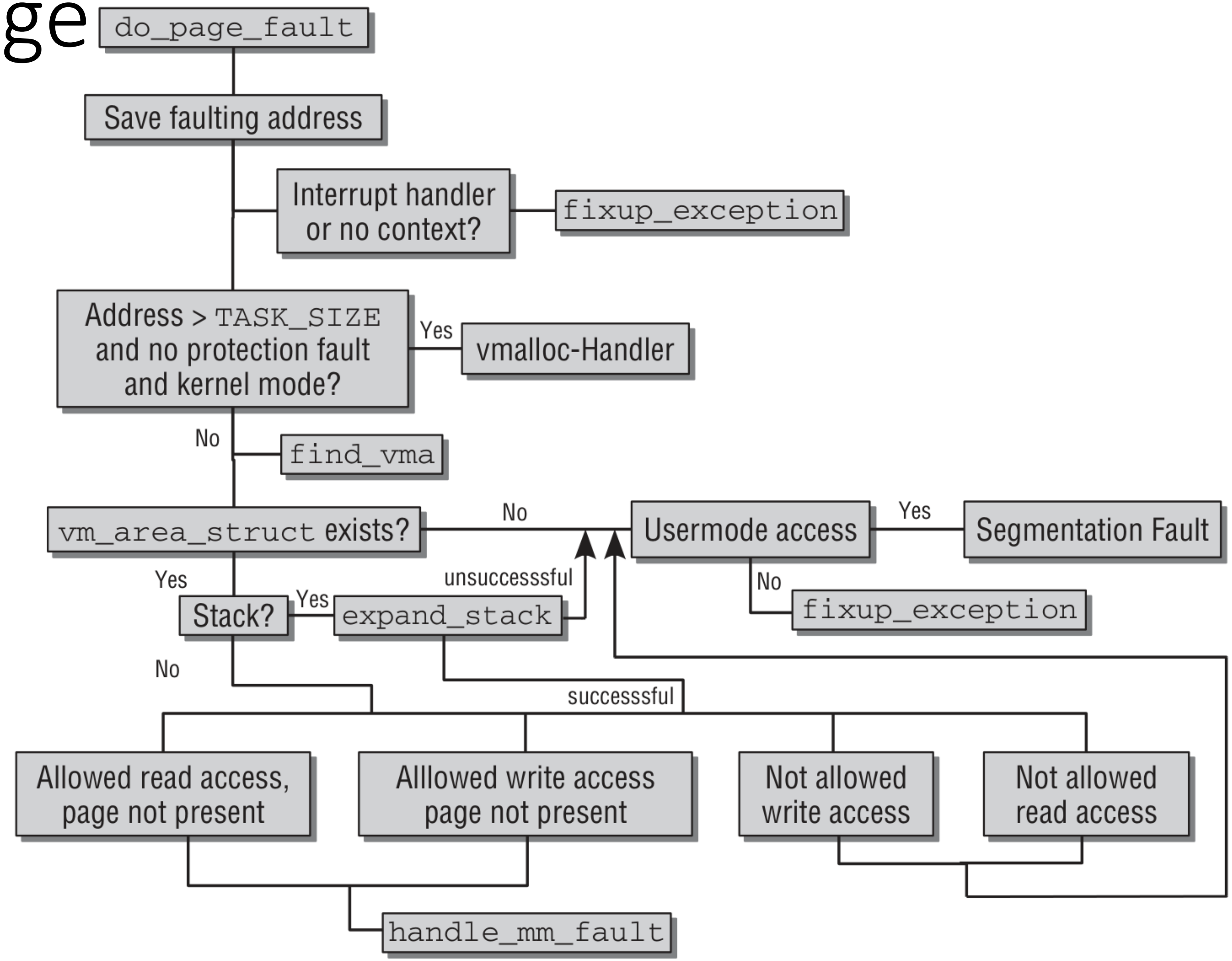


- Q: how many ops to translate?
- Q: why does this perform *at all*?

Handle a page fault

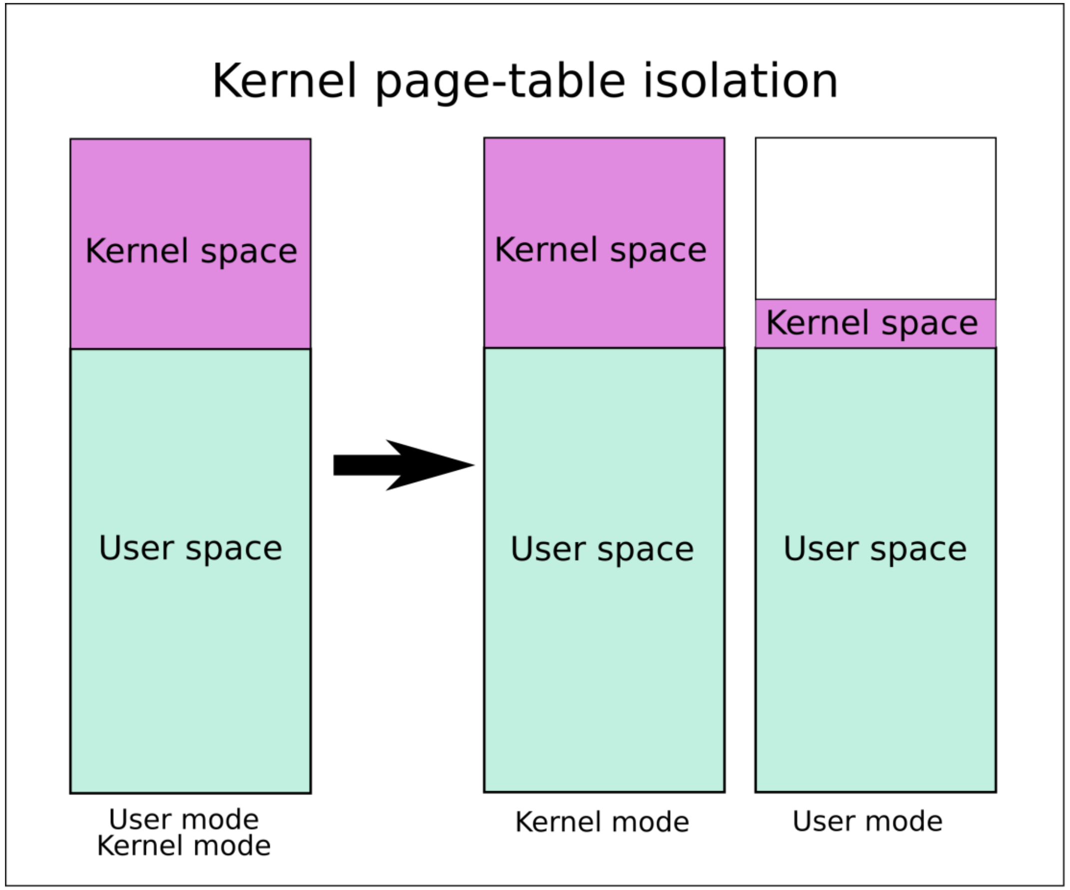


Handle a page fault, more detail



Sidebar: KASLR, KPTI

- KASLR → Kernel address space layout randomization
- KPTI/KAISER → Kernel page-table isolation separates most kernel memory from user
- In user-space, map only information needed for interrupts, sys enter/exit



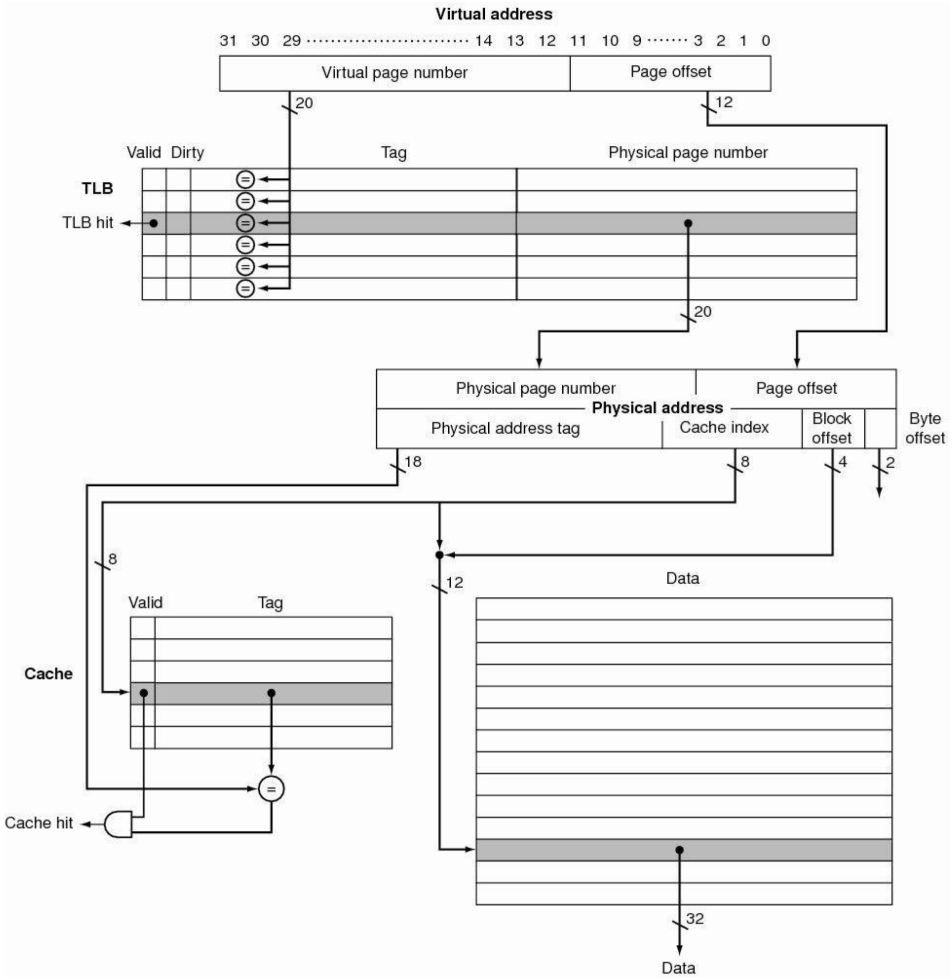
TLB organization

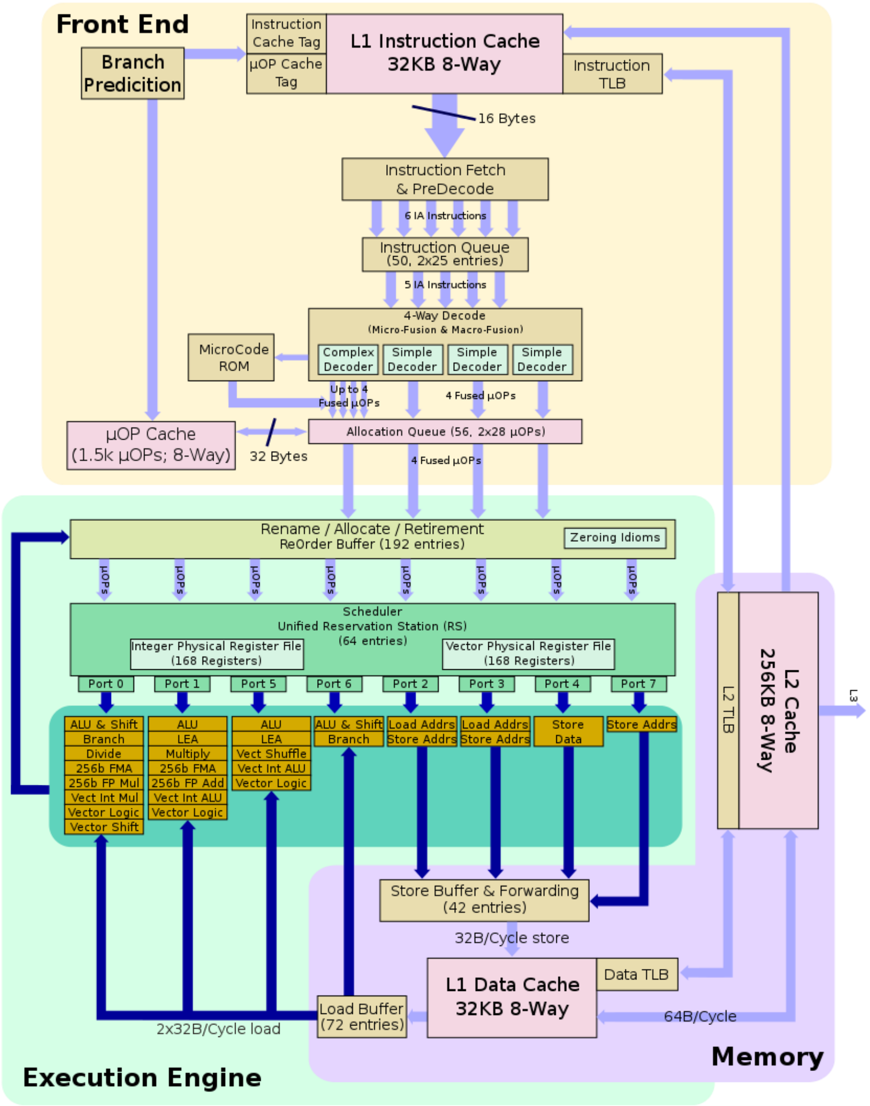
- How big does TLB actually have to be?
  - Usually small: 128-512 entries
  - Not very big, can support higher associativity
- TLB usually organized as fully-associative cache
  - Lookup is by Virtual Address
  - Returns Physical Address + other info
  - Recent architectures: set associativity in multi-level TLBs

Virtual Address	Physical Address	Dirty	Ref	Valid	Access	ASID
0xFA00	0x0003	Y	N	Y	R/W	34
0x0040	0x0010	N	Y	Y	R	0
0x0041	0x0011	N	Y	Y	R	0

Example: MIPS R3000

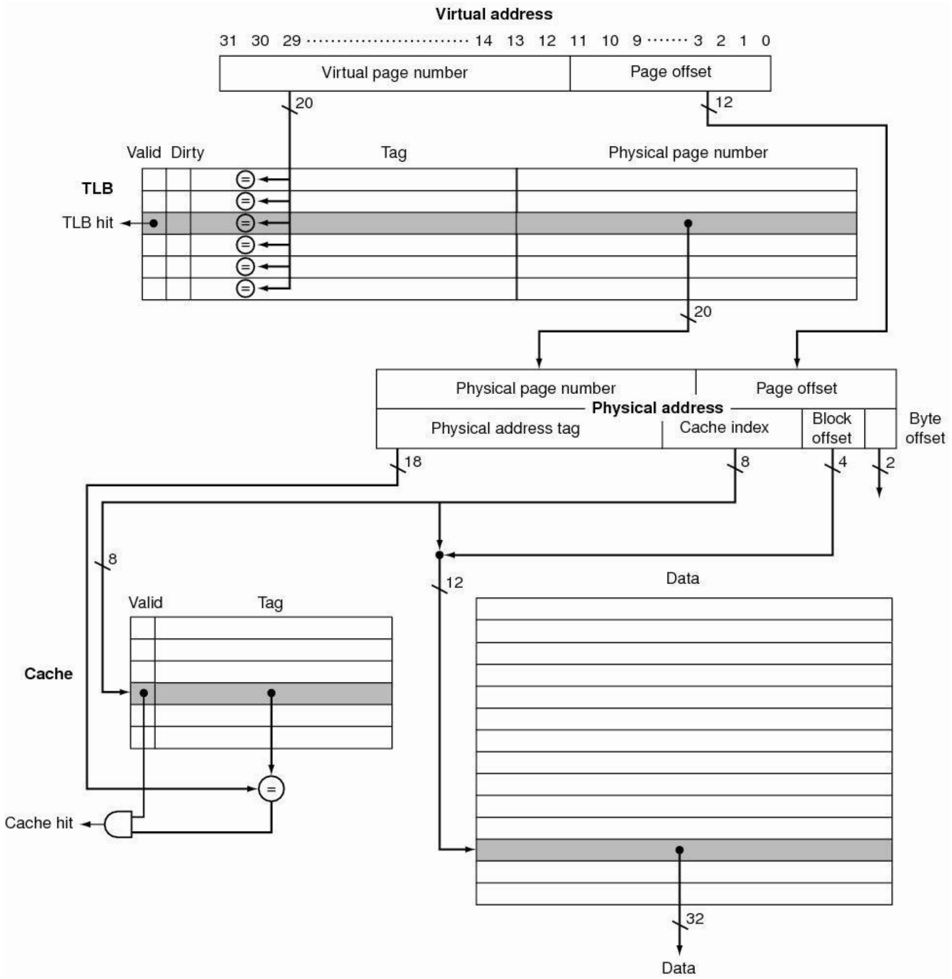
It's all about the TLB!

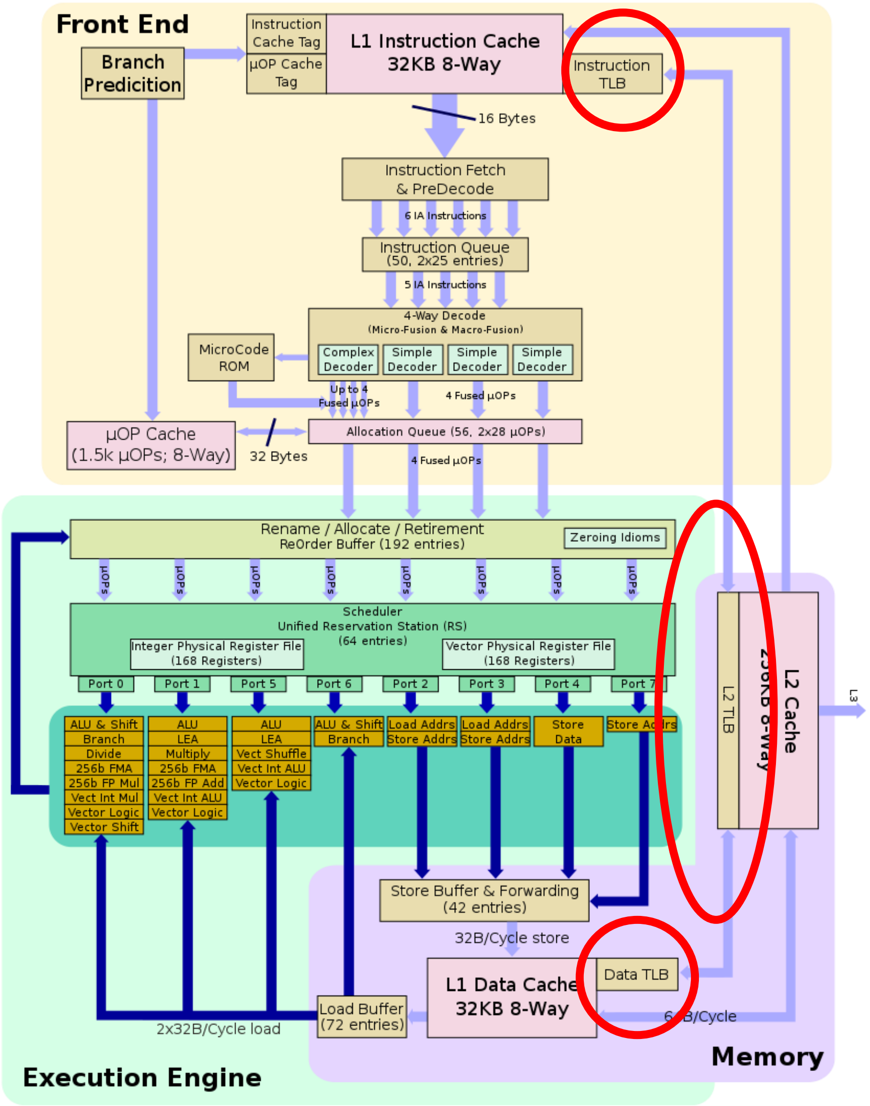




Broadwell

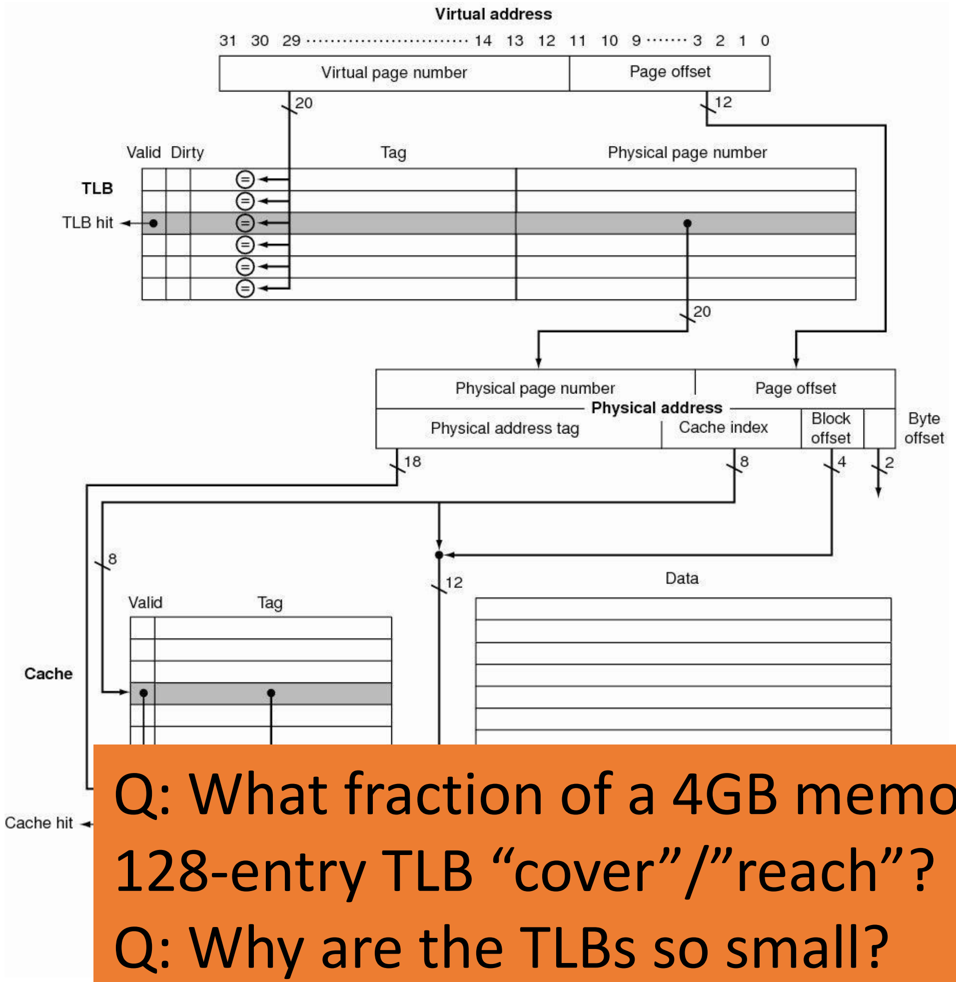
It’s all about the TLB!

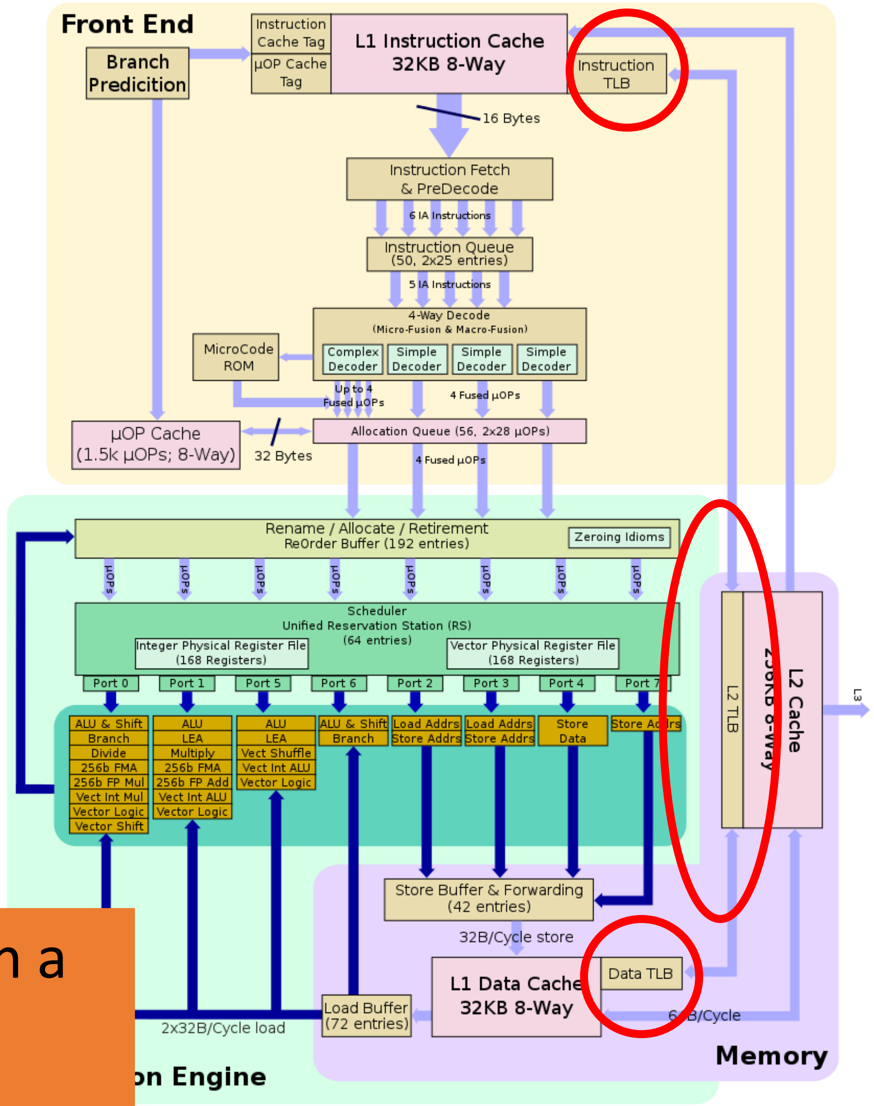




Broadwell

It’s all about the TLB!





Q: What fraction of a 4GB memory can a 128-entry TLB “cover”/”reach”?

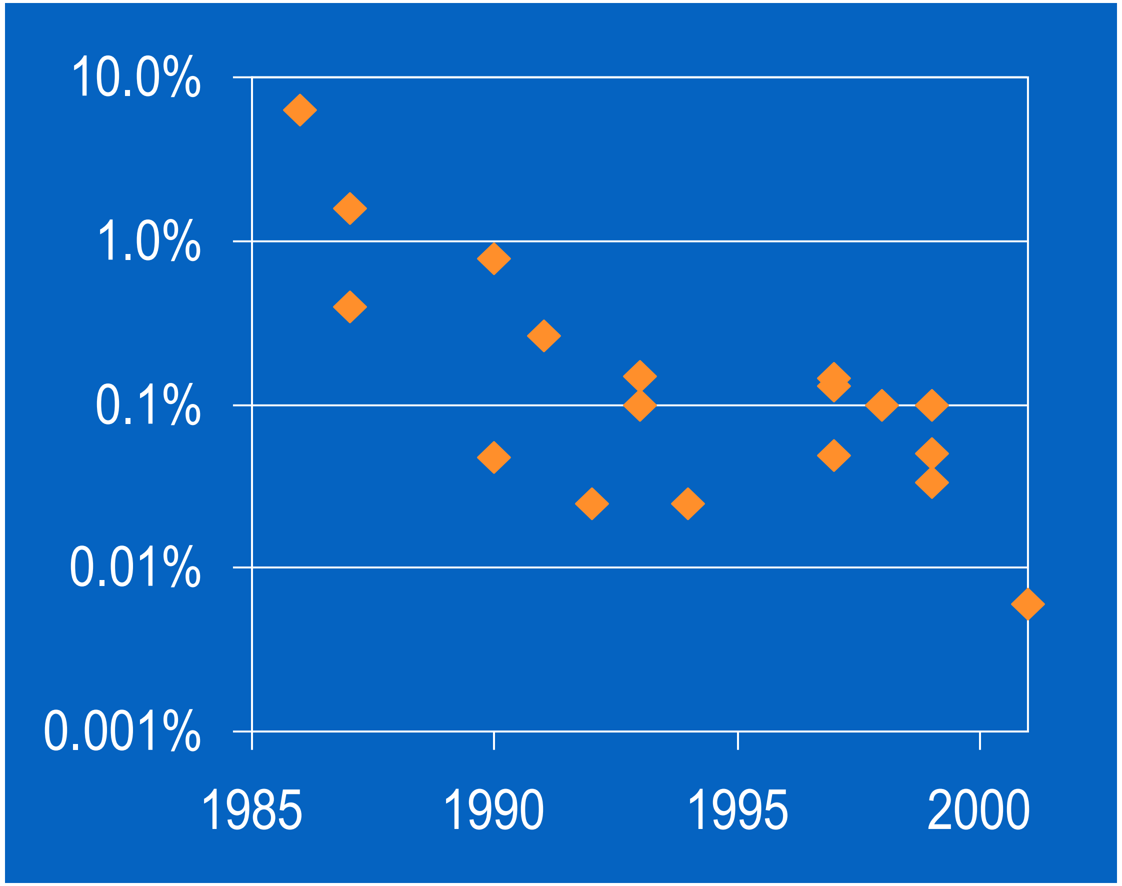
Q: Why are the TLBs so small?

Broadwell

TLB coverage trend

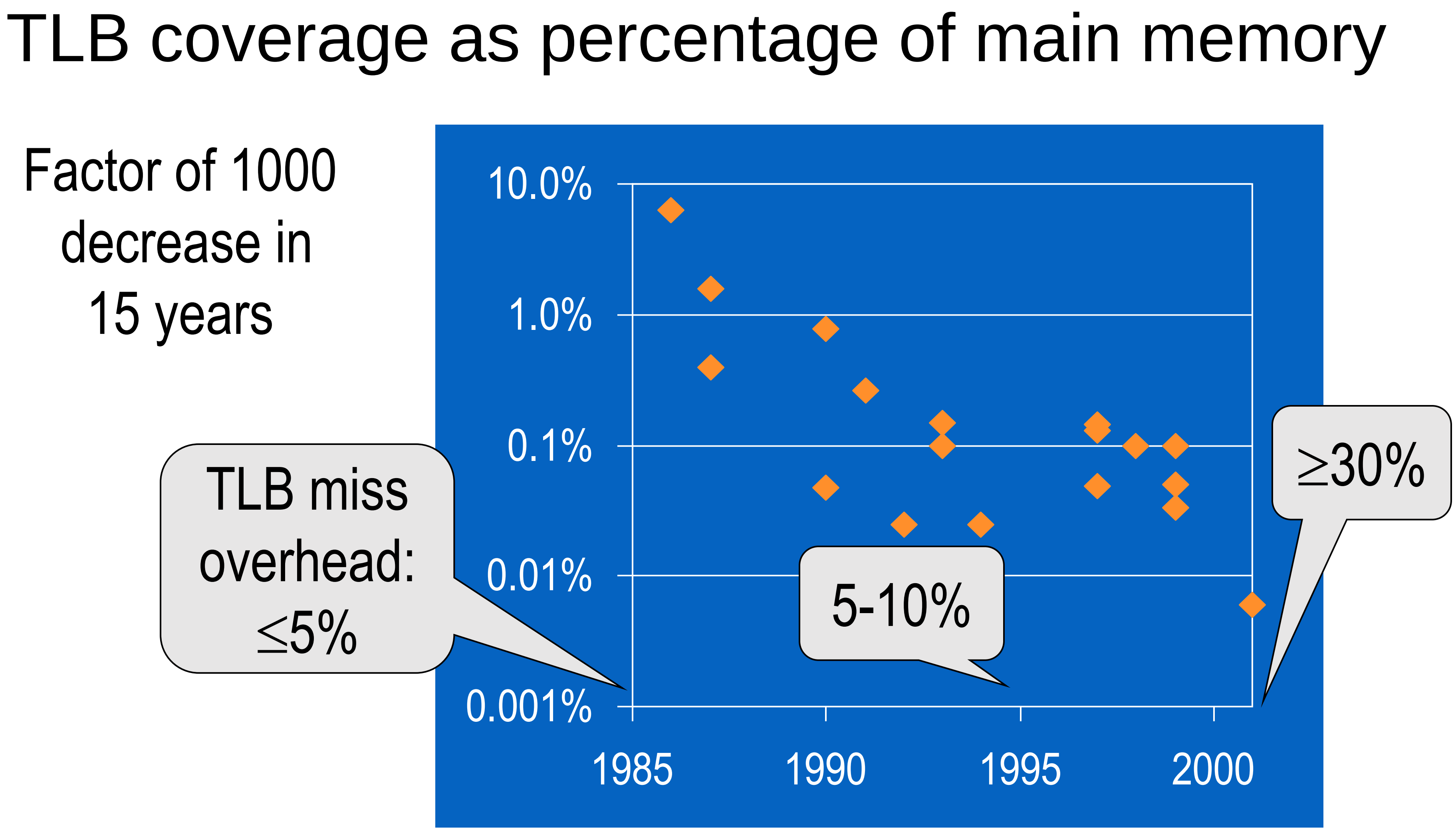
TLB coverage as percentage of main memory

Factor of 1000 decrease in 15 years



TLB coverage trend

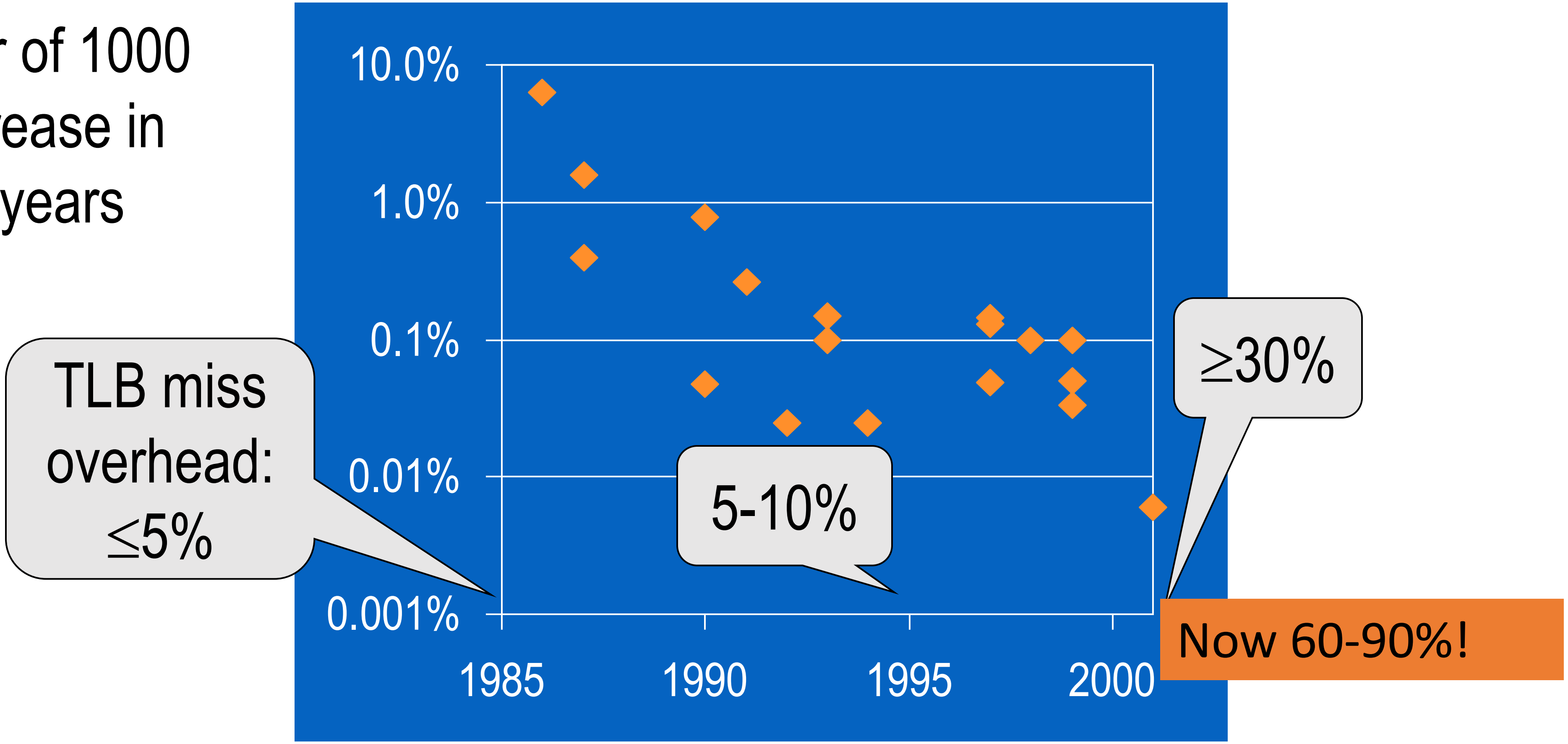
Factor of 1000 decrease in 15 years



TLB coverage trend

TLB coverage as percentage of main memory

Factor of 1000 decrease in 15 years



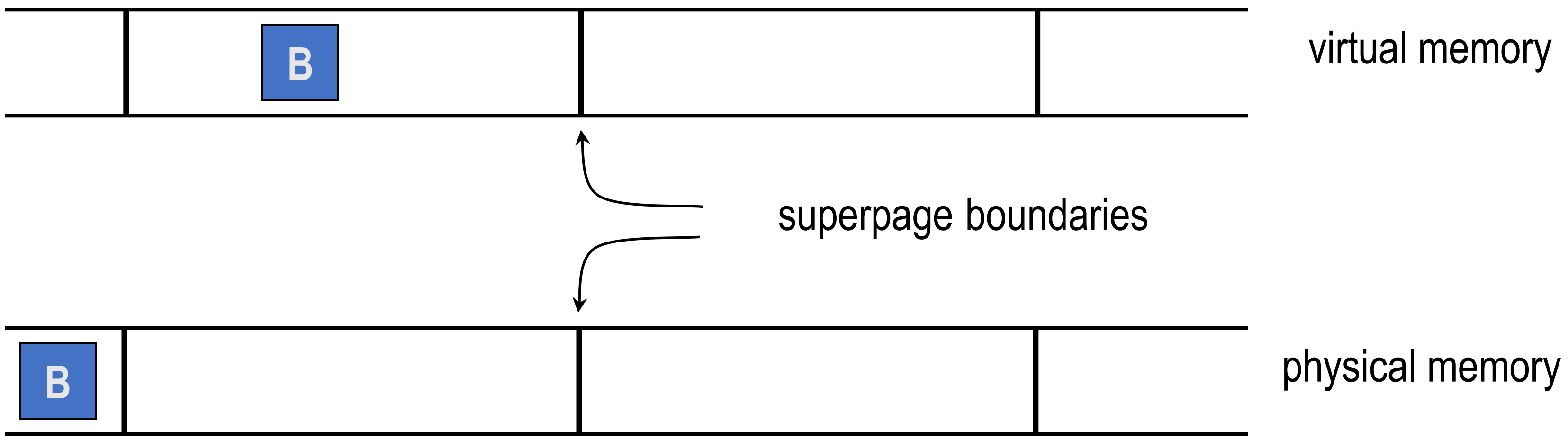
How to increase TLB coverage

- Typical TLB coverage ≈ 1 MB (before ~2015)
- Use superpages!
  - large and small pages
- Increase TLB coverage
  - no increase in TLB size
  - no internal fragmentation
- Paper covers the challenges

Superpage Concepts

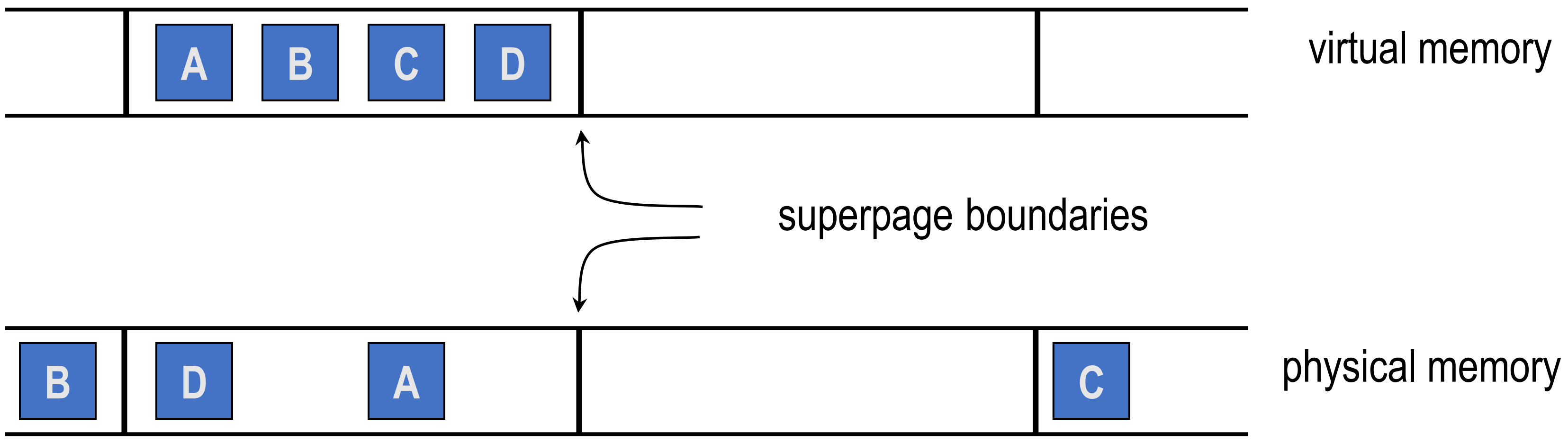
- Memory pages of larger sizes
  - supported by most modern CPUs via MMU
- Otherwise, same as normal pages
  - power of 2 size
  - use only one TLB entry
  - contiguous
  - aligned (physically and virtually)
  - uniform protection attributes
  - one reference bit, one dirty bit

Issue 1: superpage allocation



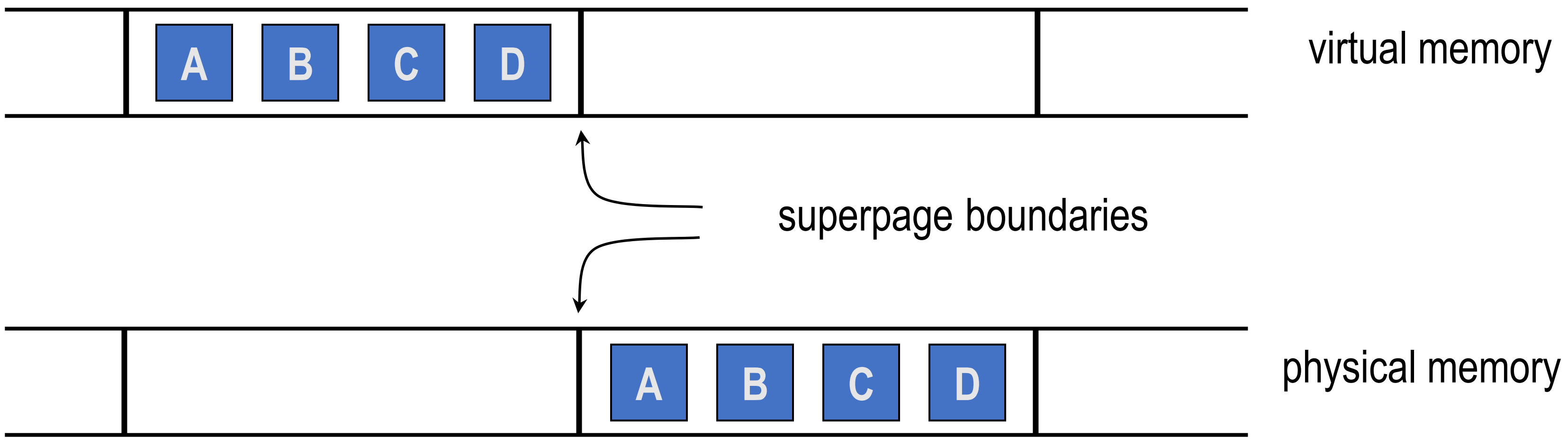
- How / when / what size to allocate?

Issue 1: superpage allocation



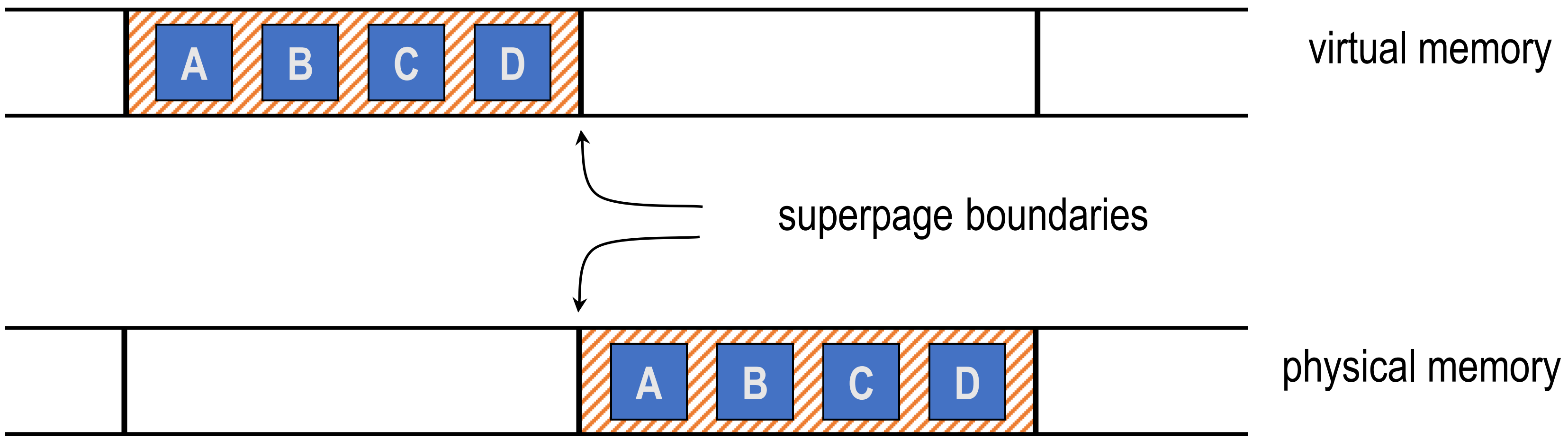
- How / when / what size to allocate?

Issue 1: superpage allocation



- How / when / what size to allocate?

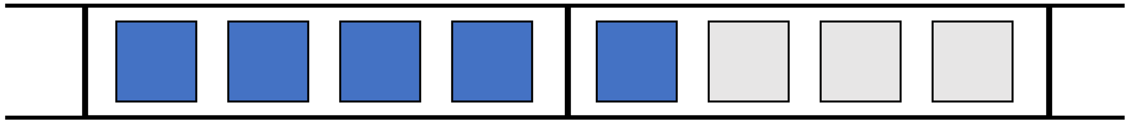
Issue 1: superpage allocation



- How / when / what size to allocate?

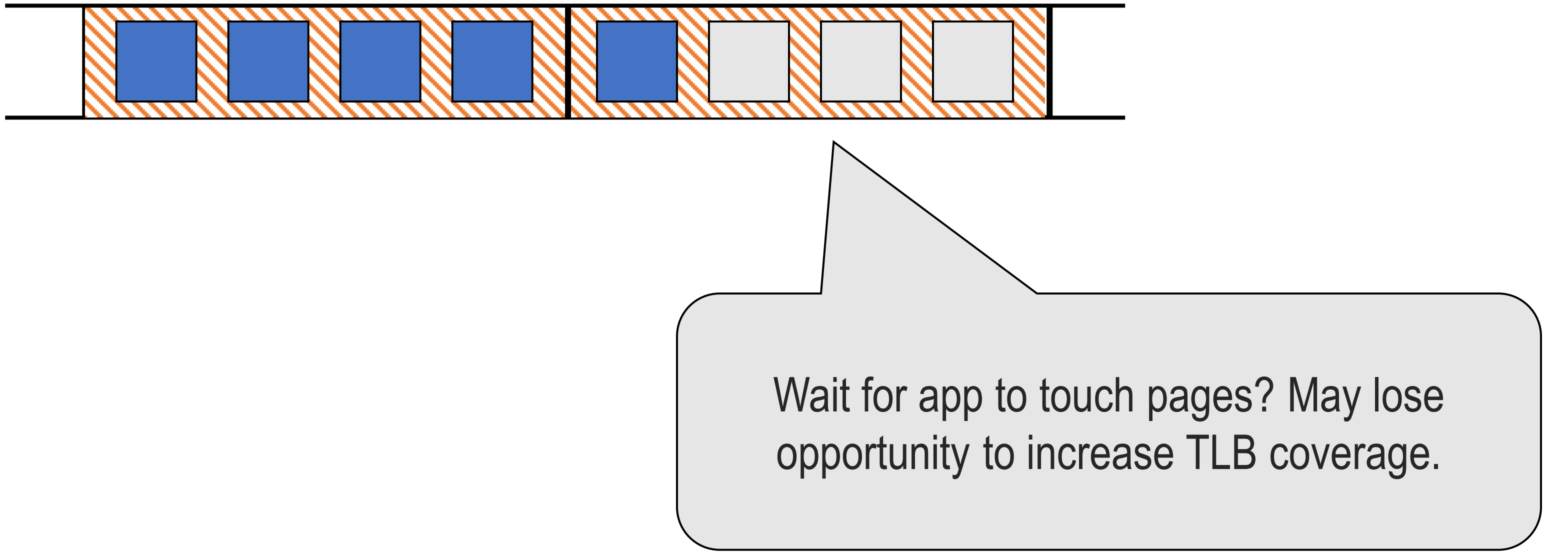
Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?



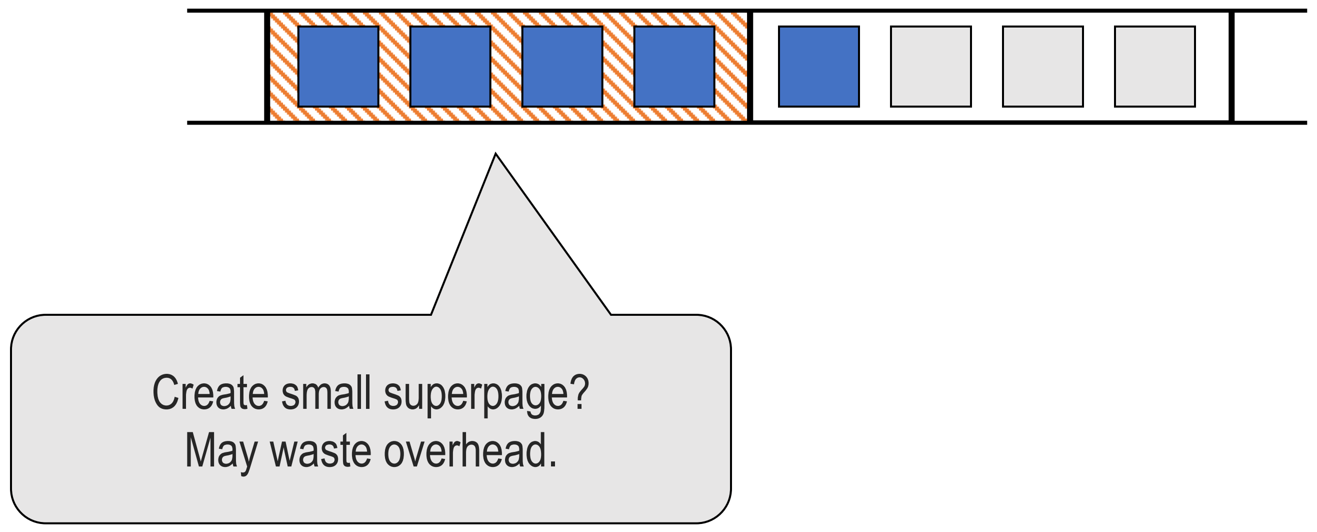
Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?



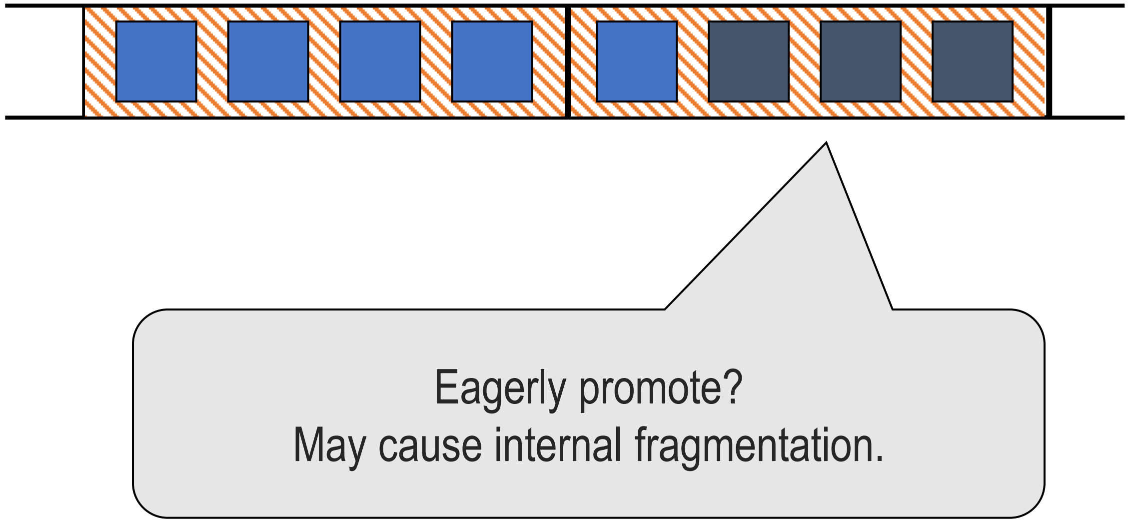
Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?



Issue 2: promotion

- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?



Issue 3: demotion

Demotion: “splinter” superpage into smaller pages

When?

- page attributes of base pages in superpage become non-uniform
- during partial pageouts

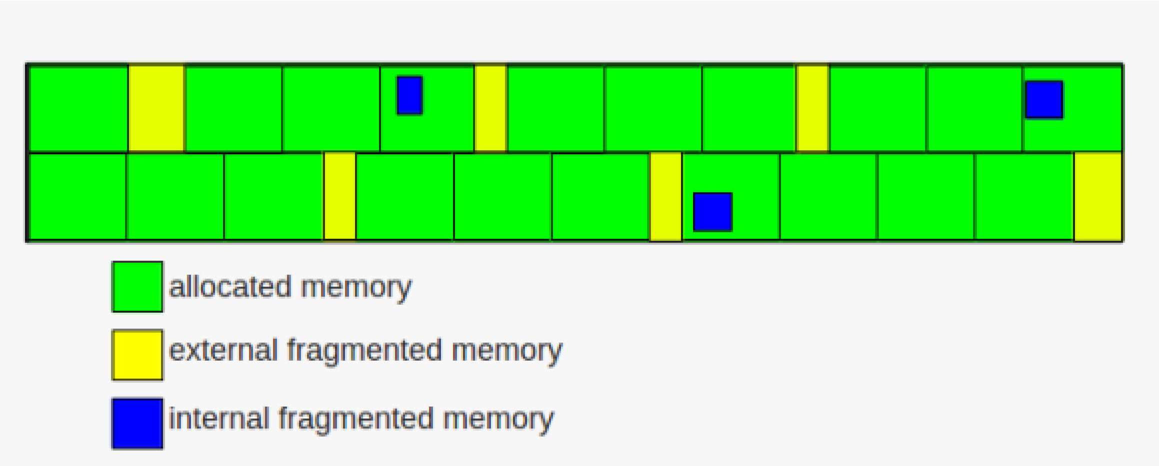
Group activity

- Summarize the paper’s policy for
  - Allocation
  - Promotion
  - Demotion
- Give a specific example of where each would happen

Issue 4: fragmentation

- Memory becomes fragmented due to
  - use of multiple page sizes
  - persistence of file cache pages
  - scattered *wired* (non-pageable) pages
- Contiguity: contended resource
- OS must
  - use contiguity restoration techniques
  - trade off impact of contiguity restoration against superpage benefits

Issue 4: fragmentation

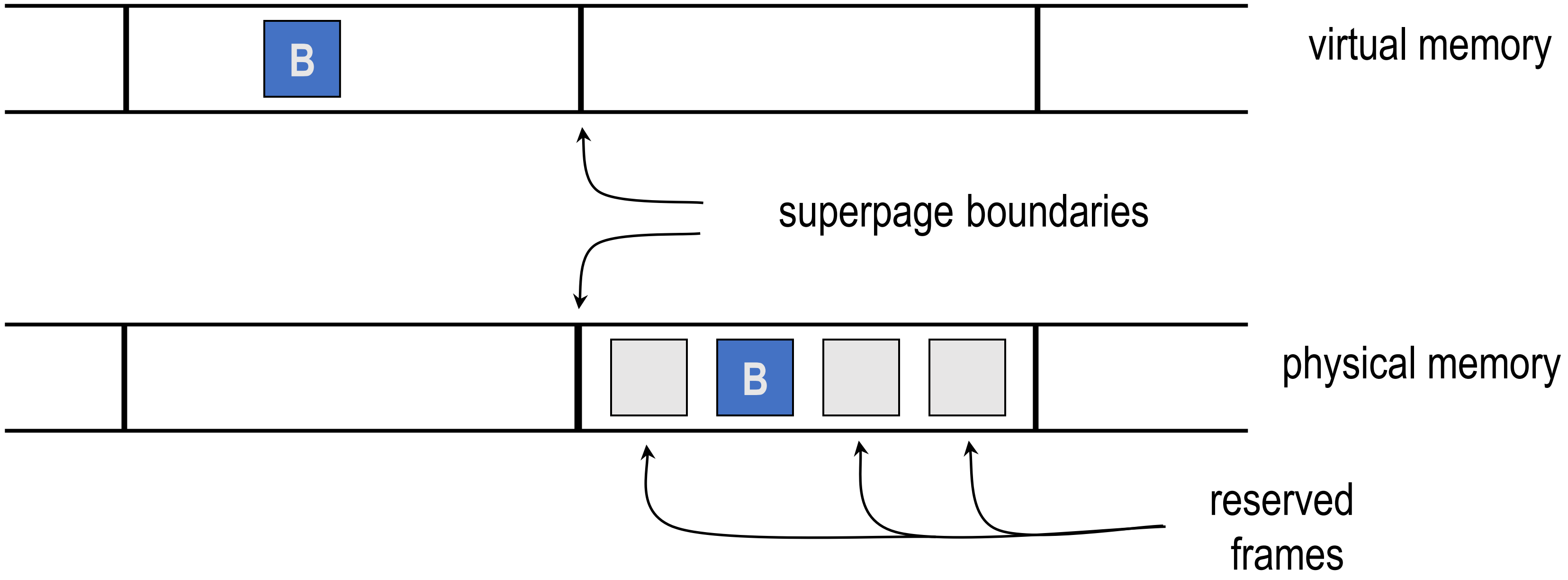


- Memory becomes fragmented due to
  - use of multiple page sizes
  - persistence of file cache pages
  - scattered *wired* (non-pageable) pages
- Contiguity: contended resource
- OS must
  - use contiguity restoration techniques
  - trade off impact of contiguity restoration against superpage benefits

FreeBSD Design

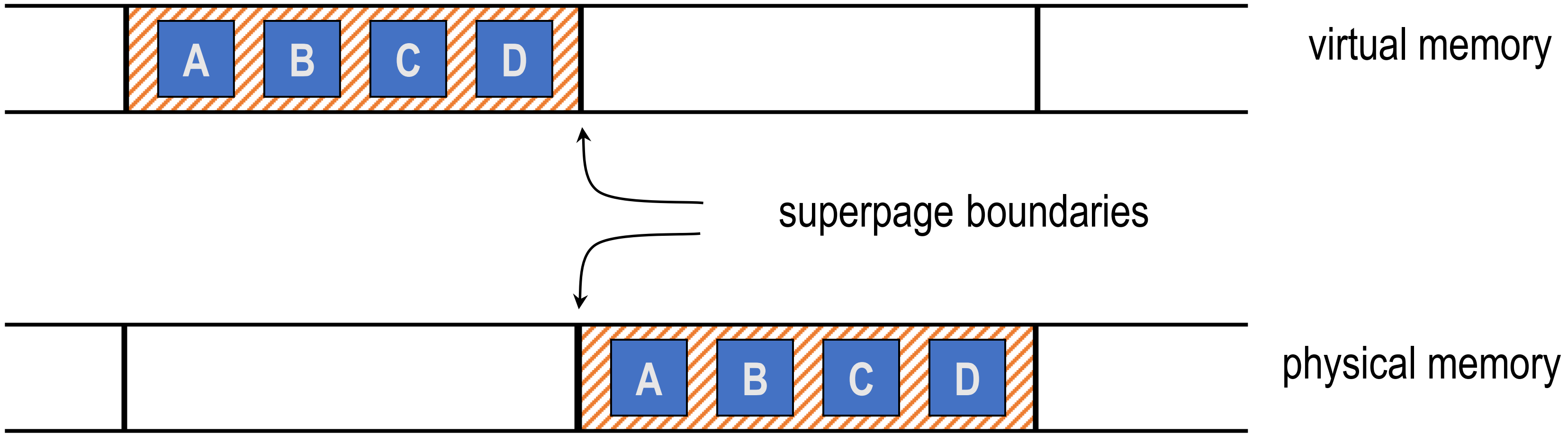
Superpage allocation

Preemptible reservations



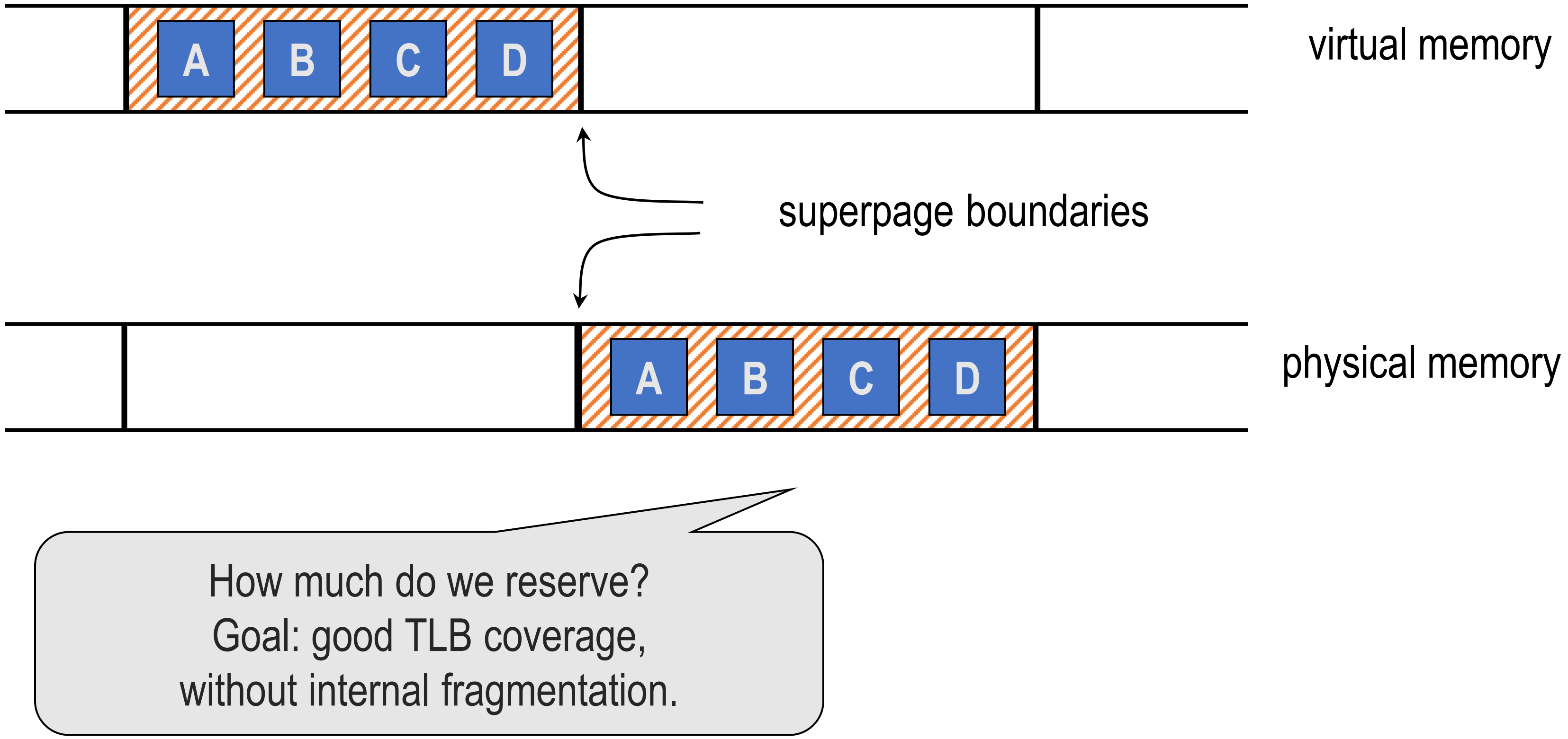
Superpage allocation

Preemptible reservations



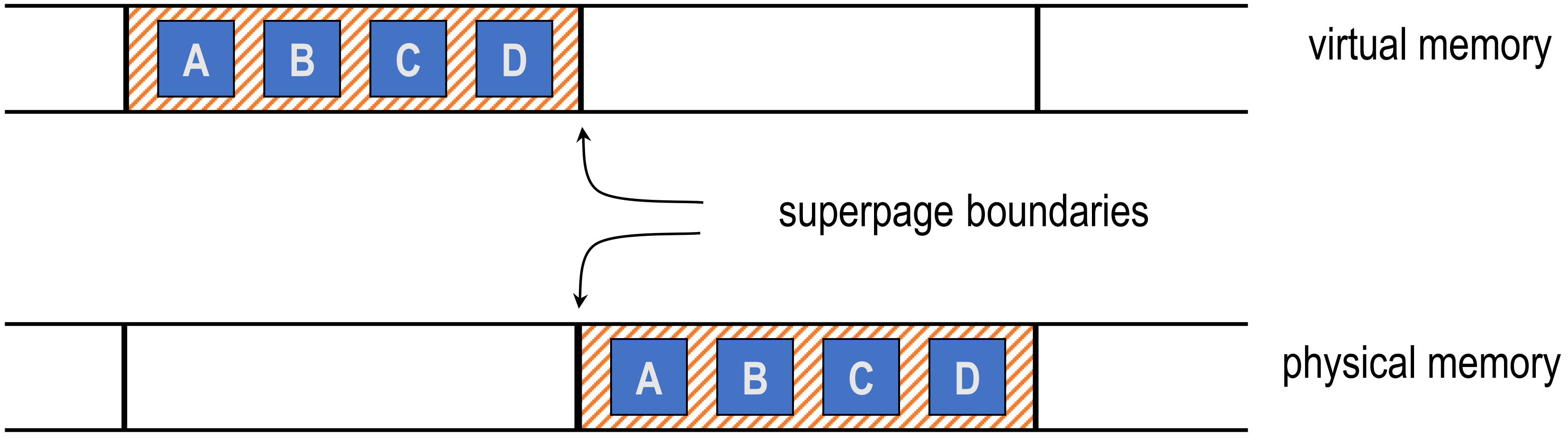
Superpage allocation

Preemptible reservations



Superpage allocation

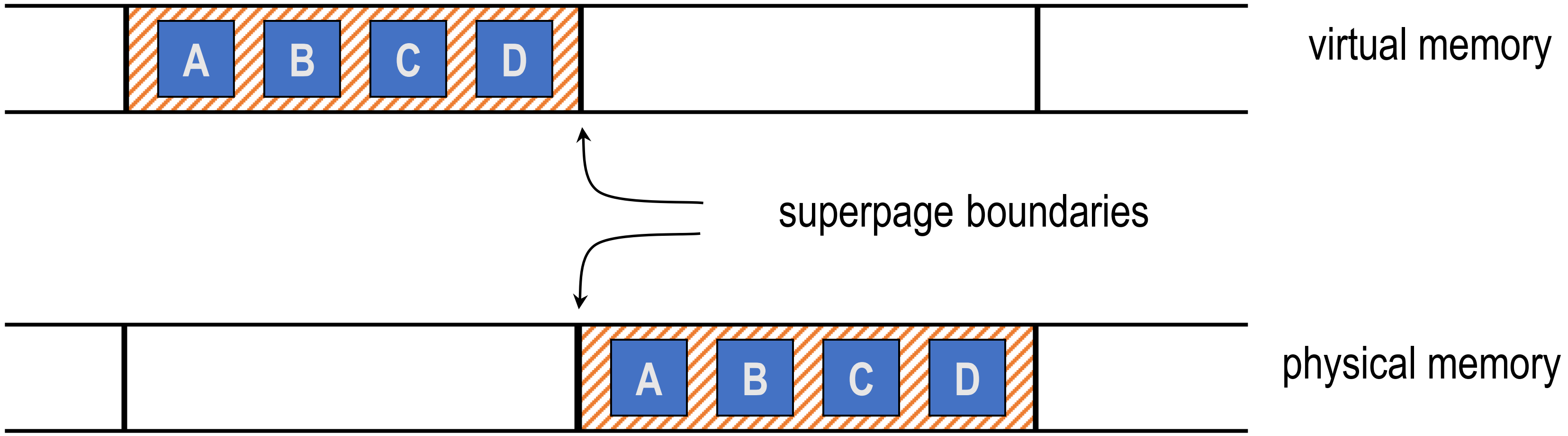
Preemptible reservations



How much do we reserve?
Goal: good TLB coverage,
without internal fragmentation.

Superpage allocation

Preemptible reservations

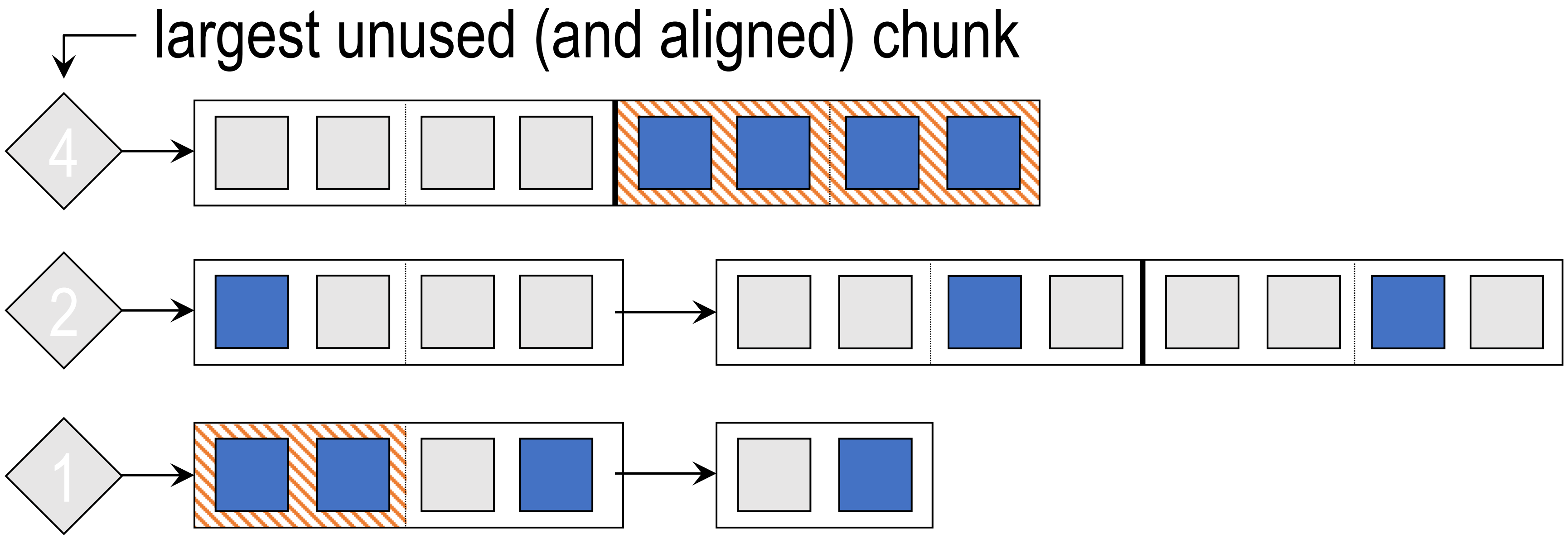


How much do we reserve?
Goal: good TLB coverage,
without internal fragmentation.

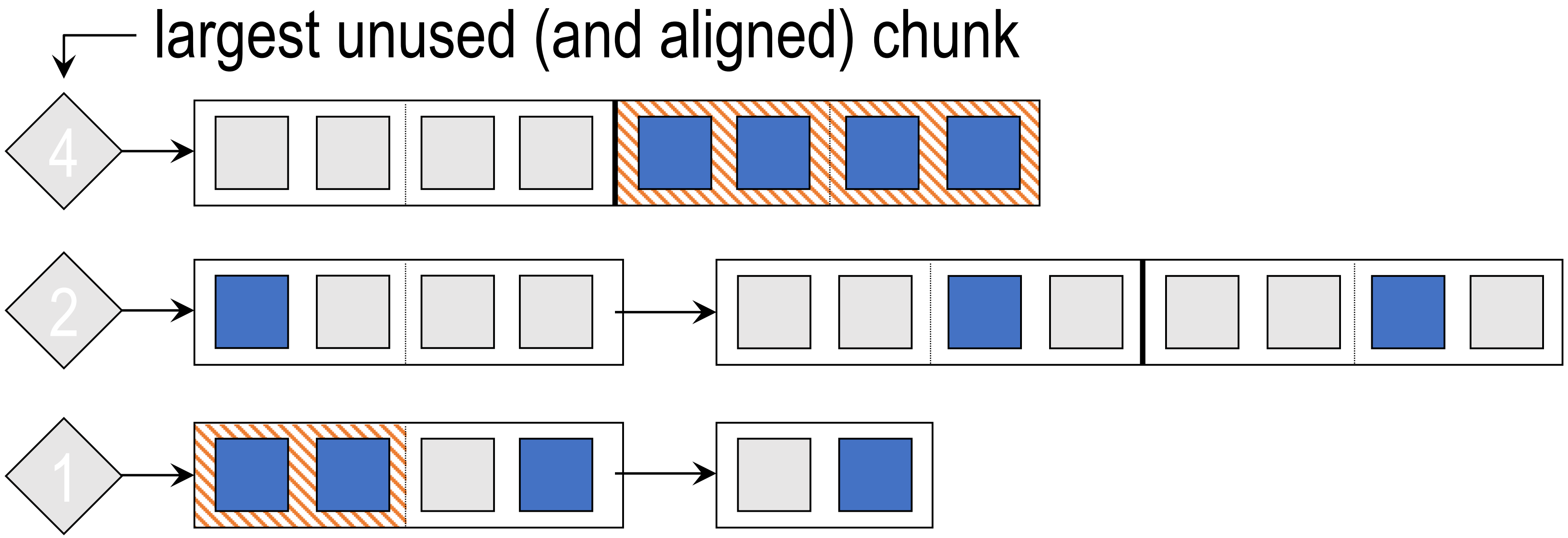
Allocation: reservation size

- Opportunistic policy
- Choose biggest size no larger than object (e.g., file)
- Size not available → preempt or resign to a smaller size

Allocation: managing reservations



Allocation: managing reservations

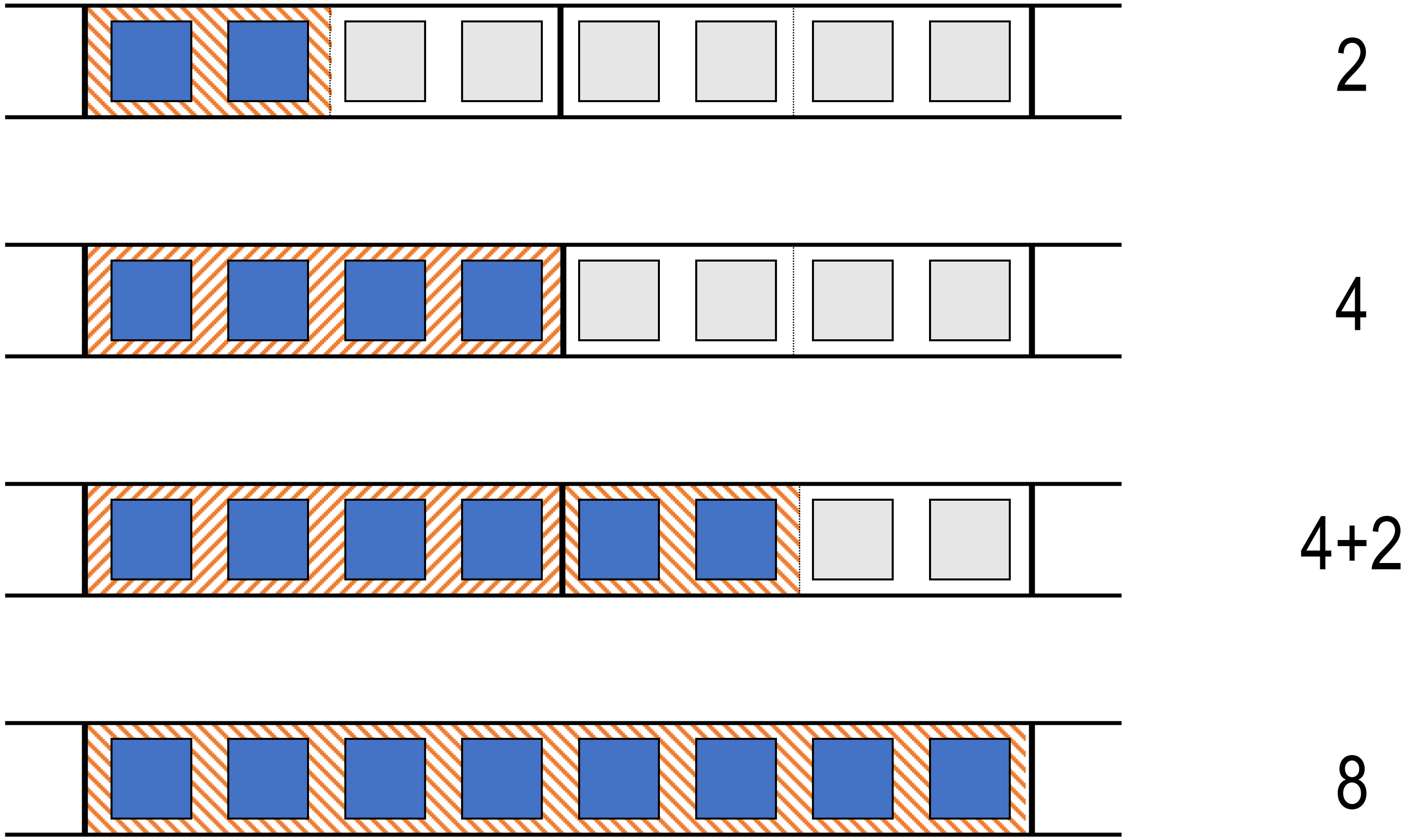


best candidate for preemption at front:

- reservation whose most recently populated frame was populated the least recently

Incremental promotions

Promotion policy: opportunistic

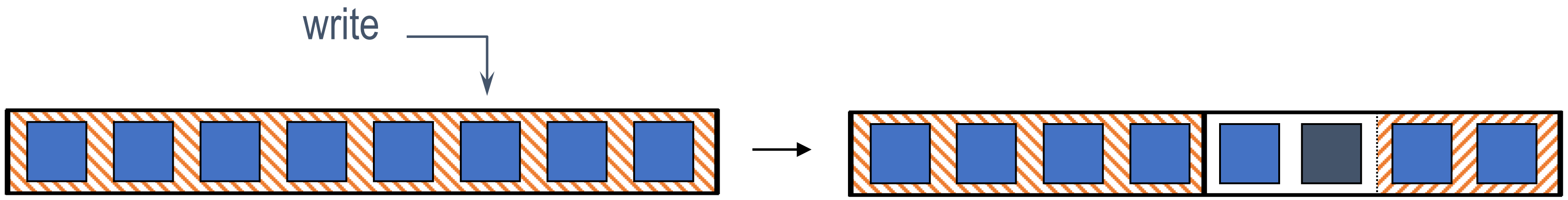


Speculative demotions

- One reference bit per superpage
  - How do we detect portions of a superpage not referenced anymore?
- On memory pressure, demote superpages when resetting ref bit
- Re-promote (incrementally) as pages are referenced

Demotions: dirty superpages

- One dirty bit per superpage
  - what's dirty and what's not?
  - page out entire superpage
- Demote on first write to clean superpage



- Re-promote (incrementally) as other pages are dirtied

Fragmentation control

- Low contiguity: modified page daemon
  - restore contiguity
    - move clean, inactive pages to the free list
  - minimize impact
    - prefer pages that contribute the most to contiguity
    - keep contents for as long as possible
      (even when part of a reservation:
      if reactivated, break reservation)
- Cluster wired pages

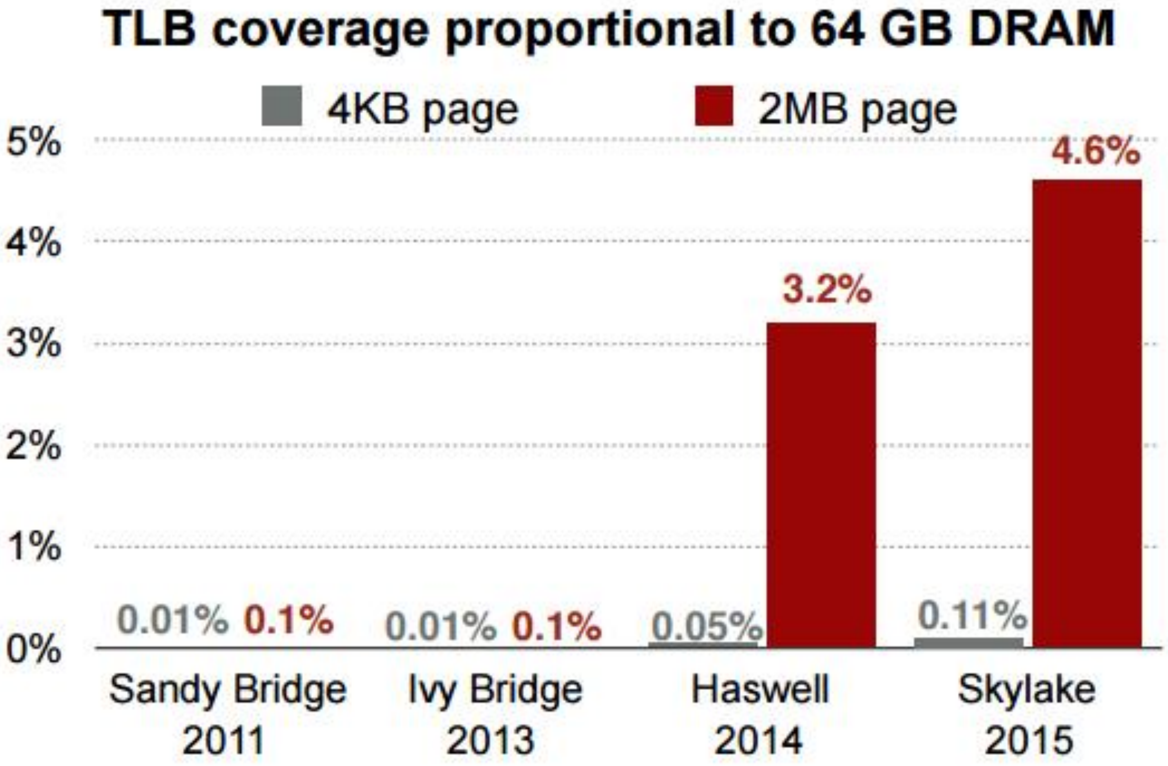
Intel’s Sunny Cove 2019

- 5-Level Paging
  - Large virtual address (57 bits, up from 48 bits)
  - Large virtual address space (128 PiB, up from 256 TiB)
- DTLB Load (DTLB split for loads and stores)
  - 4KB; 64 entries; 4-way set associative
  - 2MB; 32 entries; 4-way set associative
  - 1GB; 8 entries; 8-way set associative
- DTLB Stores
  - 4KB, 2MB, 1GB; 16 entries; 16-way set associative
- STLB, 2048 entries
  - 2,048 entries; 16-way set associative

https://en.wikichip.org/wiki/intel/microarchitectures/sunny_cove

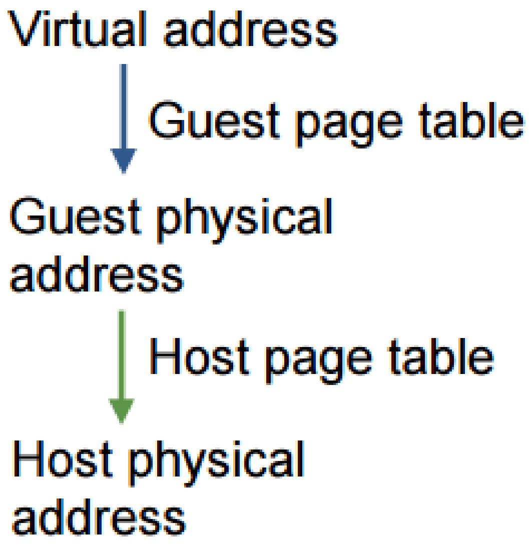
Huge pages improve TLB coverage

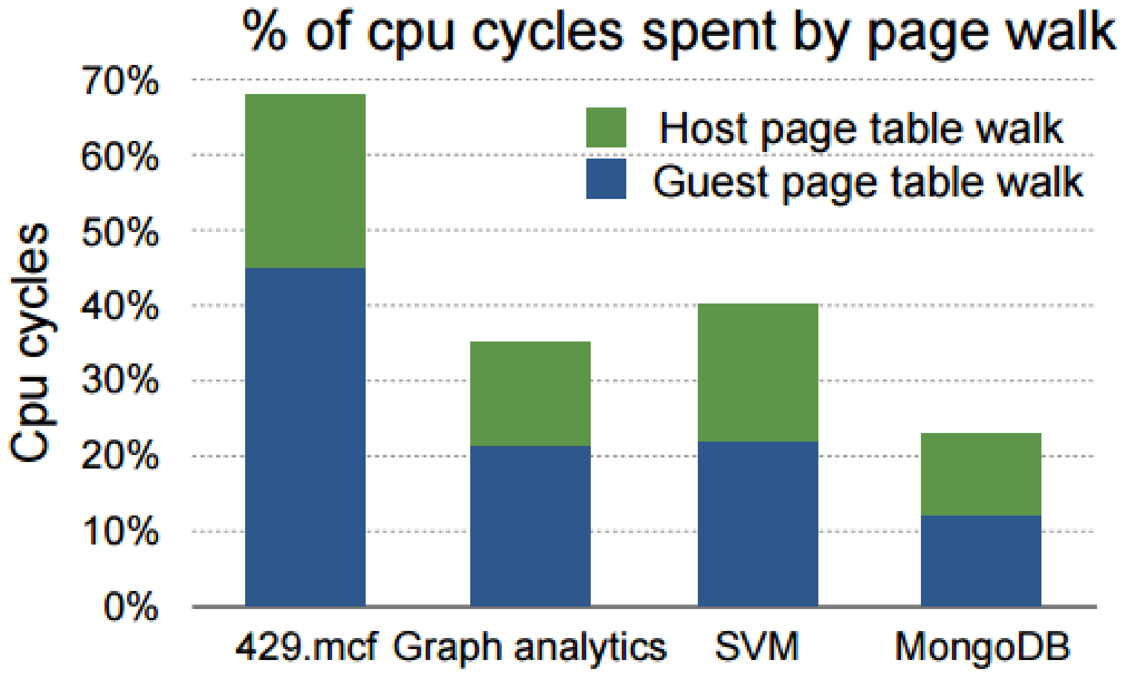
- Architecture supports larger page size (e.g., 2MB page)
  - Intel: 0 to 1,536 entries in 2 years (2013 ~ 2015)
- Operating system has the burden of better huge page support



High address translation cost

- Virtualization requires additional address translation





Guest superpages Or Host superpages?

- Need both!

Workloads	h_B g_H	h_H g_B	h_H g_H
429.mcf	1.18	1.13	1.43
Canneal	1.11	1.10	1.32
SVM	1.14	1.17	1.53
Tunkrank	1.11	1.11	1.30
Nutch	1.01	1.07	1.12
MovieRecmd	1.03	1.02	1.11
Olio	1.43	1.08	1.46
Redis	1.12	1.04	1.20
MongoDB	1.08	1.22	1.37

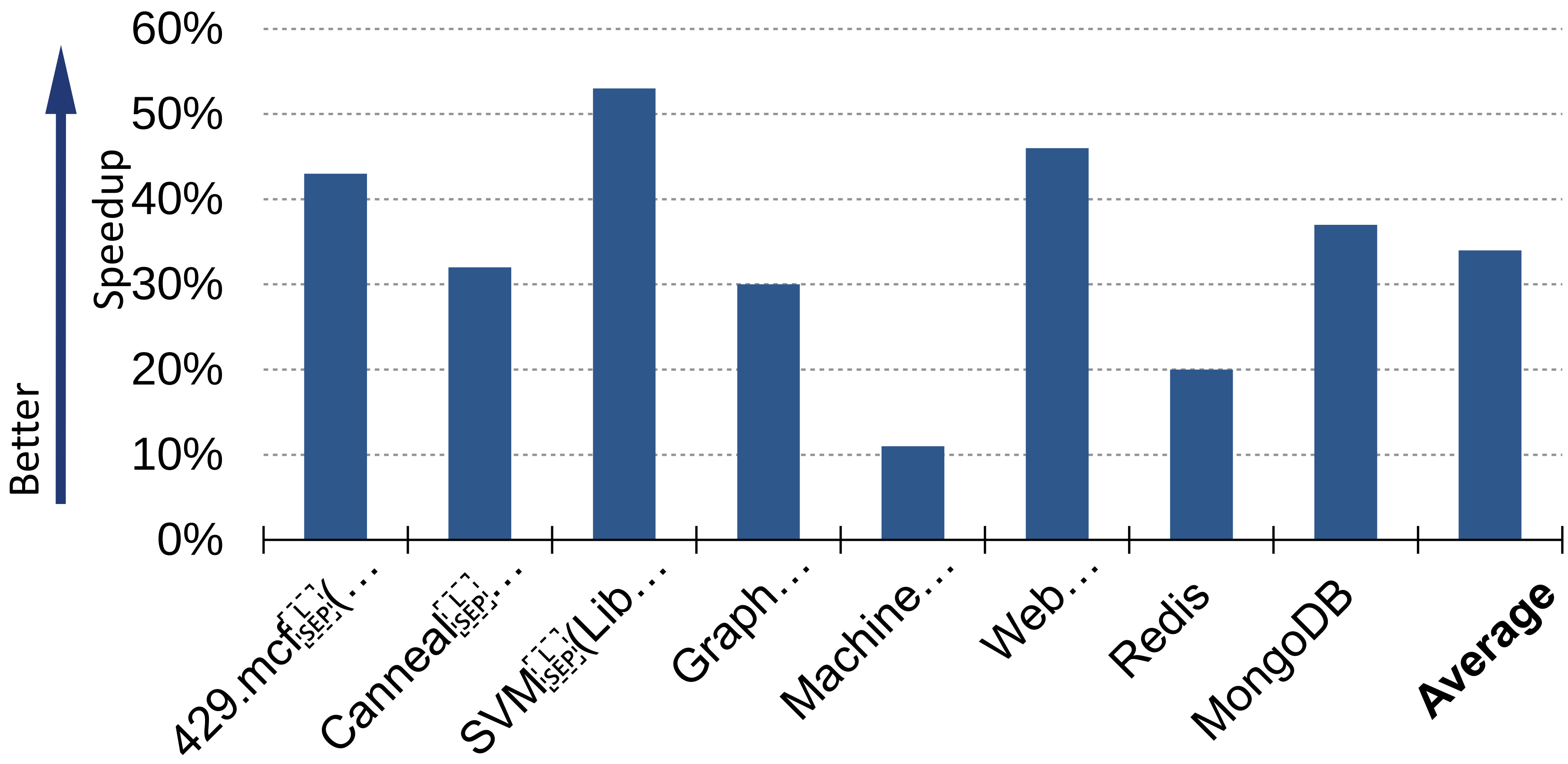
Table 3: Application speed up for huge page (2 MB) support relative to host (h) and guest (g) using base (4 KB) pages. For example, h_B means the host uses base pages and h_H means the host uses both base and huge pages.

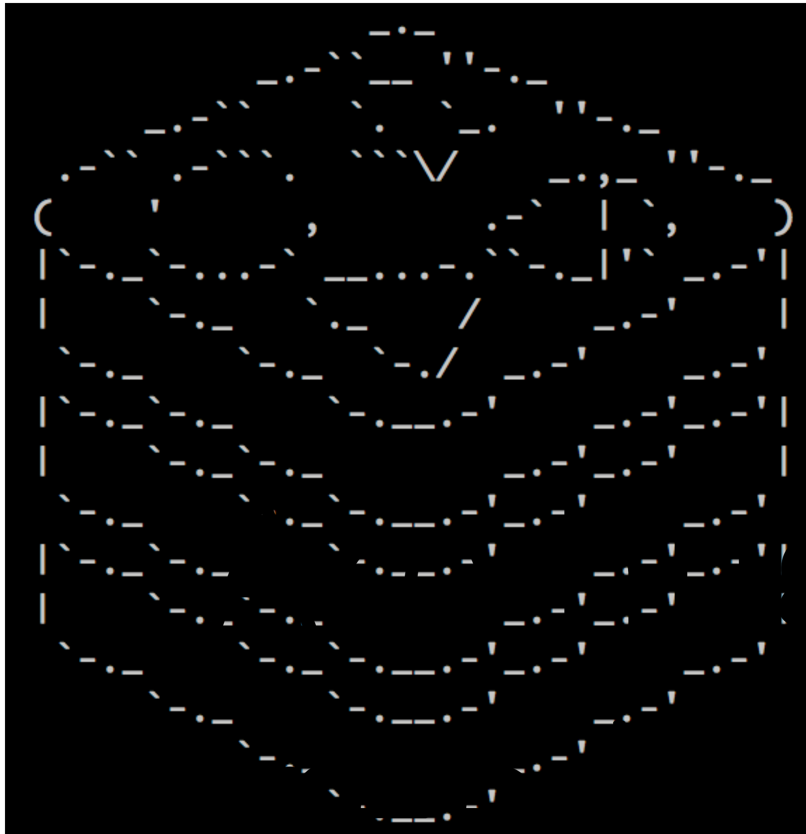
Operating system support for huge pages

- User-controlled huge page management
  - Admin reserves huge page in advance
  - New APIs for memory allocation/deallocation
  - It could fail to reserve huge pages when memory is fragmented
- Transparent huge page management
  - Developers do not know about huge page
  - OS Transparently allocates/deallocates huge pages
  - OS manages memory fragmentation

Huge pages improve performance

- Application speed up over using base pages only



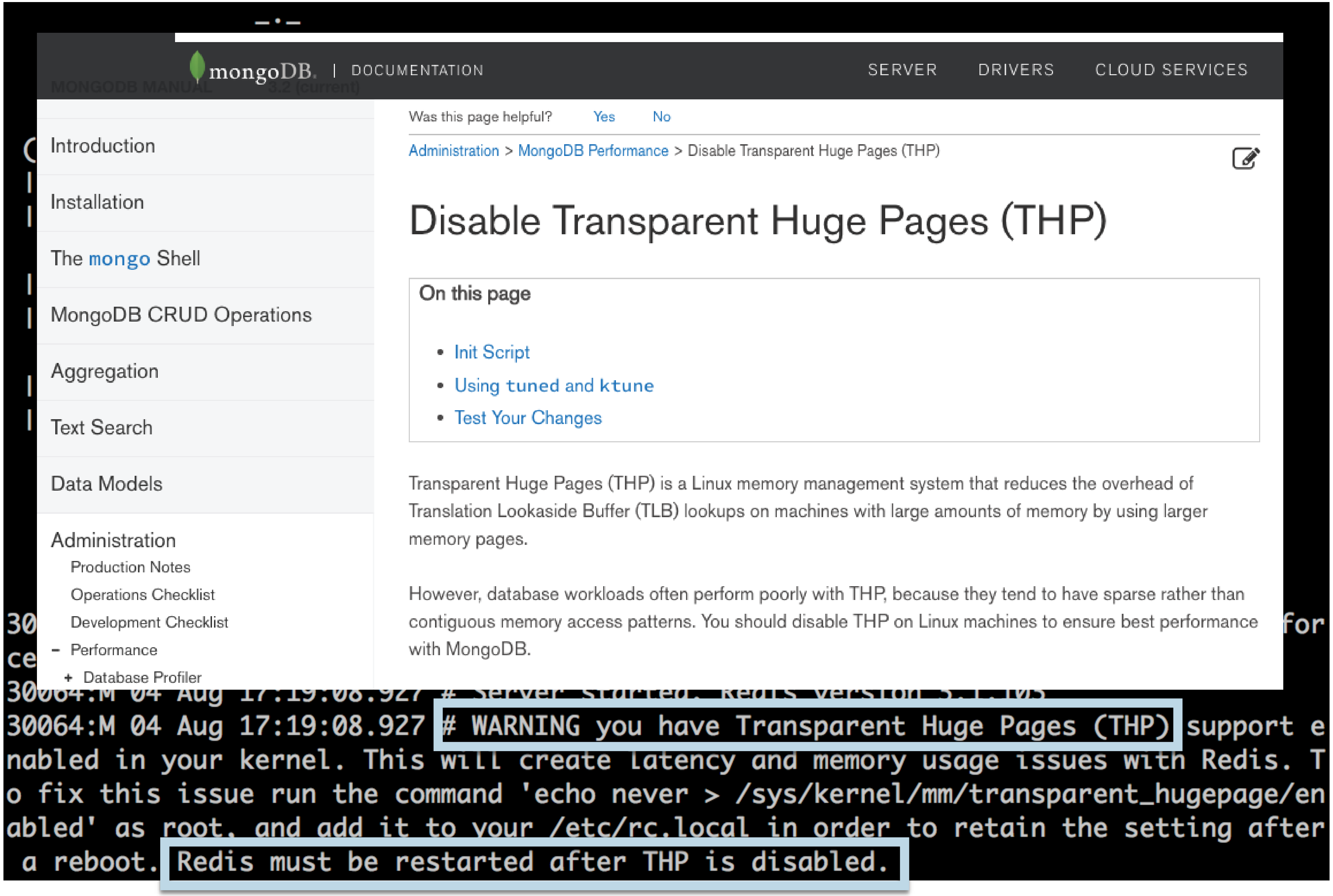


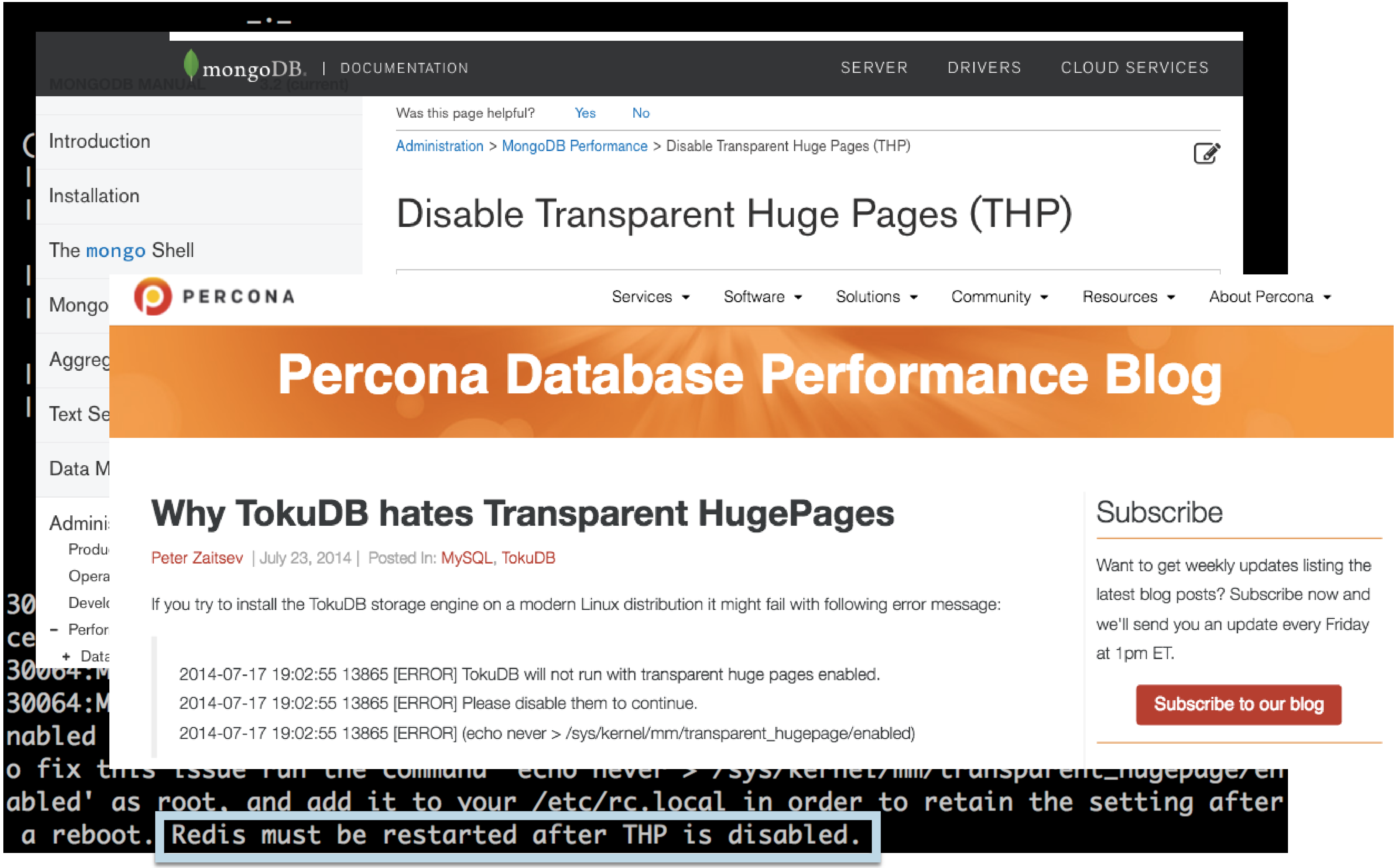
```
Redis 3.1.103 (3bba4842/1) 64 bit

Running in standalone mode
Port: 6379
PID: 30064

http://redis.io

30064:M 04 Aug 17:19:08.927 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
30064:M 04 Aug 17:19:08.927 # Server started, Redis version 3.1.103
30064:M 04 Aug 17:19:08.927 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
```


mongoDB. | DOCUMENTATION
SERVER
DRIVERS
CLOUD SERVICES
Introduction
Installation
The mongo Shell
MongoDB CRUD Operations
Aggregation
Text Search
Data Models
Administration
Production Notes
Operations Checklist
Development Checklist
- Performance
+ Database Profiler
Was this page helpful? Yes No
Administration > MongoDB Performance > Disable Transparent Huge Pages (THP)
Disable Transparent Huge Pages (THP)
On this page
Init Script
Using tuned and ktune
Test Your Changes
Transparent Huge Pages (THP) is a Linux memory management system that reduces the overhead of Translation Lookaside Buffer (TLB) lookups on machines with large amounts of memory by using larger memory pages.
However, database workloads often perform poorly with THP, because they tend to have sparse rather than contiguous memory access patterns. You should disable THP on Linux machines to ensure best performance with MongoDB.
30064:M 04 Aug 17:19:08.927 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.


mongoDB. | DOCUMENTATION
SERVER
DRIVERS
CLOUD SERVICES
Was this page helpful? Yes No
Administration > MongoDB Performance > Disable Transparent Huge Pages (THP)
Disable Transparent Huge Pages (THP)
Introduction
Installation
The mongo Shell
PERCONA
Services
Software
Solutions
Community
Resources
About Percona
Percona Database Performance Blog
Why TokuDB hates Transparent HugePages
Peter Zaitsev | July 23, 2014 | Posted In: MySQL, TokuDB
If you try to install the TokuDB storage engine on a modern Linux distribution it might fail with following error message:
2014-07-17 19:02:55 13865 [ERROR] TokuDB will not run with transparent huge pages enabled.
2014-07-17 19:02:55 13865 [ERROR] Please disable them to continue.
2014-07-17 19:02:55 13865 [ERROR] (echo never > /sys/kernel/mm/transparent_hugepage/enabled)
Subscribe
Want to get weekly updates listing the latest blog posts? Subscribe now and we'll send you an update every Friday at 1pm ET.
Subscribe to our blog
abled' as root, and add it to your /etc/rc.local in order to retain the setting after
a reboot. Redis must be restarted after THP is disabled.

Disable Transparent Huge Pages (THP)

2.3.2. Disable Transparent Huge Pages

S
Tran
m map
Volt
c
Ther
t
$ e
$ e
a

IBM Support

splunk> docs PRODUCTS SOLUTIONS CUSTOMERS COMMUNITY SPLEXICON Support & Services My Account Search D

cloudera Why Cloudera Products Services & Support Solutions Get Started

okta PRODUCT DOCS DISCUSSION SUPPORT GET STARTE

Transparent Huge Pages: Thanks for your help…please don't help

By the next morning **CPU contention was worse.**

The alarmingly high system CPU usage that we'd seen in the previous 3 months was always due to MySQL using kernel mutex. But since we'd fixed that problem, *what the heck was this?*

We discussed turning off TCMalloc, but that would've been a mistake. Implementing TCMalloc was a critical link in the chain of problems and solutions that ultimately strengthened our platform.

We discovered very quickly that the culprit this time was a *khugepaged* enabled by a Linux kernel flag called **Transparent Huge Pages** (THP; turned on by default in most Linux distributions). Huge pages are designed to improve performance by helping the operating system manage large amounts of memory. They effectively increase the page size from the standard 4kb to 2MB or 1Gb (depending on how it is configured).

THP makes huge pages easier to use by, among other things, arranging your memory into larger chunks. It works great for app servers that are not performing memory-intensive operations.

a
a

- High Availability
- Backup and Disaster Recovery
- Cloudera Manager Administration
- Cloudera Navigator Data Management Component Administration

Disabling Transparent Hugepage Compaction

Most Linux platforms supported by CDH 5 include a feature called **transparent hugepage compaction** which interacts poorly with Hadoop workloads and can seriously degrade performance.

Huge page pathologies in Linux

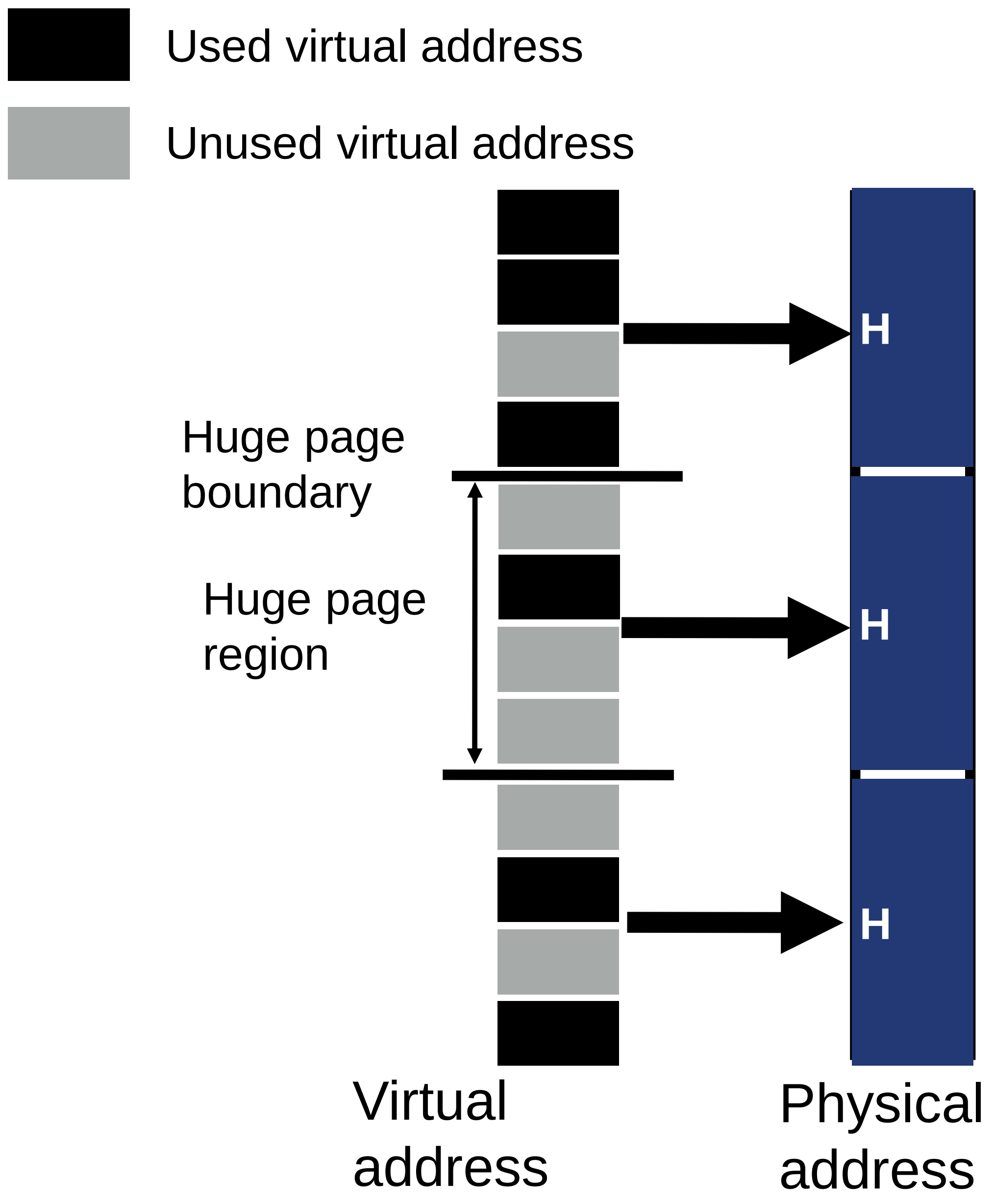
- High page fault latency
  - Due to synchronous allocation in fault handler
- Memory bloating
  - Huge pages greedily allocated
- Unfair huge page allocation
  - E.g., one VM gets huge pages, maintains improved performance

Page fault latency

- Fault handler gets huge page from allocator and zeroes it (terrible for application tail latency)
  - 4KB page : 3.6 us
  - 2MB page : 378.0 us (mostly from page zeroing)
- Fault handler can trigger memory compaction
  - 2 minutes to fragment 24 GB
  - All memory sizes eventually fragment

Memory bloating

- Greedy allocation in Linux
  - Allocate a huge page on first fault to huge page region
  - The huge page region may not be fully used
- Greedy allocation causes severe internal fragmentation
  - Memory use often sparse
- What kind of fragmentation is this?



Memory bloating

- Greedy allocation in Linux
  - Allocate a huge page on first fault to huge page region
  - The huge page region may not be fully used
- Greedy allocation causes severe internal fragmentation
  - Memory use often sparse
- What this?