TLB & TLB Coverage

Kartik Gopalan

TLBs – Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- TLB is a small cache that speeds up the translation of virtual addresses to physical addresses.
- TLB is part of the MMU hardware (comes with CPU)
- It is not a Data Cache or Instruction Cache. Those are separate.
- TLB simply caches translations from virtual page number to physical page number so that the MMU don't have to access page-table in memory too often.
- On older x86 processors, TLB had to be "flushed" upon every context switch because there is no field in TLB to identify the process context.
    - Tagged TLB can reduce this overhead

Cold Start Penalty

- Cost of repopulating the TLB (and other caches) upon a context switch.

- Immediately after a context switch, all (or many) of TLB entries are invalidated.
  - On some x86 processors, TLB has to be "flushed" upon every context switch because there is no field in TLB to identify the process context.
- Every memory access by the newly scheduled process may results in a TLB miss.

- MMU must then walk the page-table in main memory to repopulate the missing TLB entry, which takes longer than a cache hit.

TLB Coverage

- Max amount of memory mapped by TLB
  - Max mount of memory that can be accessed without TLB misses

- TLB Coverage = N x P bytes
  - N = Number of entries in TLB
  - P = Page size in bytes
  - N is fixed by hardware constraints
  - So, to increase TLB Coverage, we must increase P.

- Consider these extreme examples
  - Suppose P = 1 byte
    - TLB Coverage = N bytes only
  - Suppose P = 2^64 bytes (on a 64-bit ISA)
    - TLB Coverage = N x2^64bytes
    - TLB can perform translations for N processes without any TLB misses!

- Of course, both examples above are impractical and meant to illustrate the tradeoffs.

- But what if P is something reasonable, but greater than than the standard 4KB?

- This brings us next to superpages.

Tagged TLB

- A“tag” in each TLB entry identifies the process/ thread context to which the TLB entry belongs
- Thus TLB entries for more than one execution context can be stored simultaneously in the TLB.
  - TLB lookup hardware matches the tag in addition to the virtual page number.
- With tags, context switch no longer requires a complete TLB flush.
  - Reduces cold-start penalty.

Two types of memory translation architectures

- Architected Page Tables
  - Page table interface defined by ISA and understood by memory translation hardware
  - E.g. x86 architecture
  - MMU handles TLB miss (in hardware)
  - OS handles page faults (in software)
  - ISA specifies page table format

- Architected TLBs
  - TLB interface defined by ISA and understood by MMU
  - E.g. alpha architecture
  - TLB miss handled by OS (in software)
  - ISA does not specify page table format

Superpages/Hugepages/Largepages

Kartik Gopalan

Ref:

- “Practical, transparent operating system support for superpages”, Juan Navarro, Sitaram Iyer, Peter Druschel, Alan Cox, OSDI 2002
- https://dl.acm.org/citation.cfm?id=844138

Overview

- Increasing cost in TLB miss overhead
  - growing working sets
  - TLB size does not grow at same pace
- Processors now provide superpages
  - one TLB entry can map a large region
- OSs have been slow to harness them
  - no transparent superpage support for apps
- This talk: a practical and transparent solution to support superpages

How to increase TLB coverage

- Typical TLB coverage ≈ 1 MB
- Use superpages!
  - large and small pages
  - Increase TLB coverage
  - no increase in TLB size

Superpages

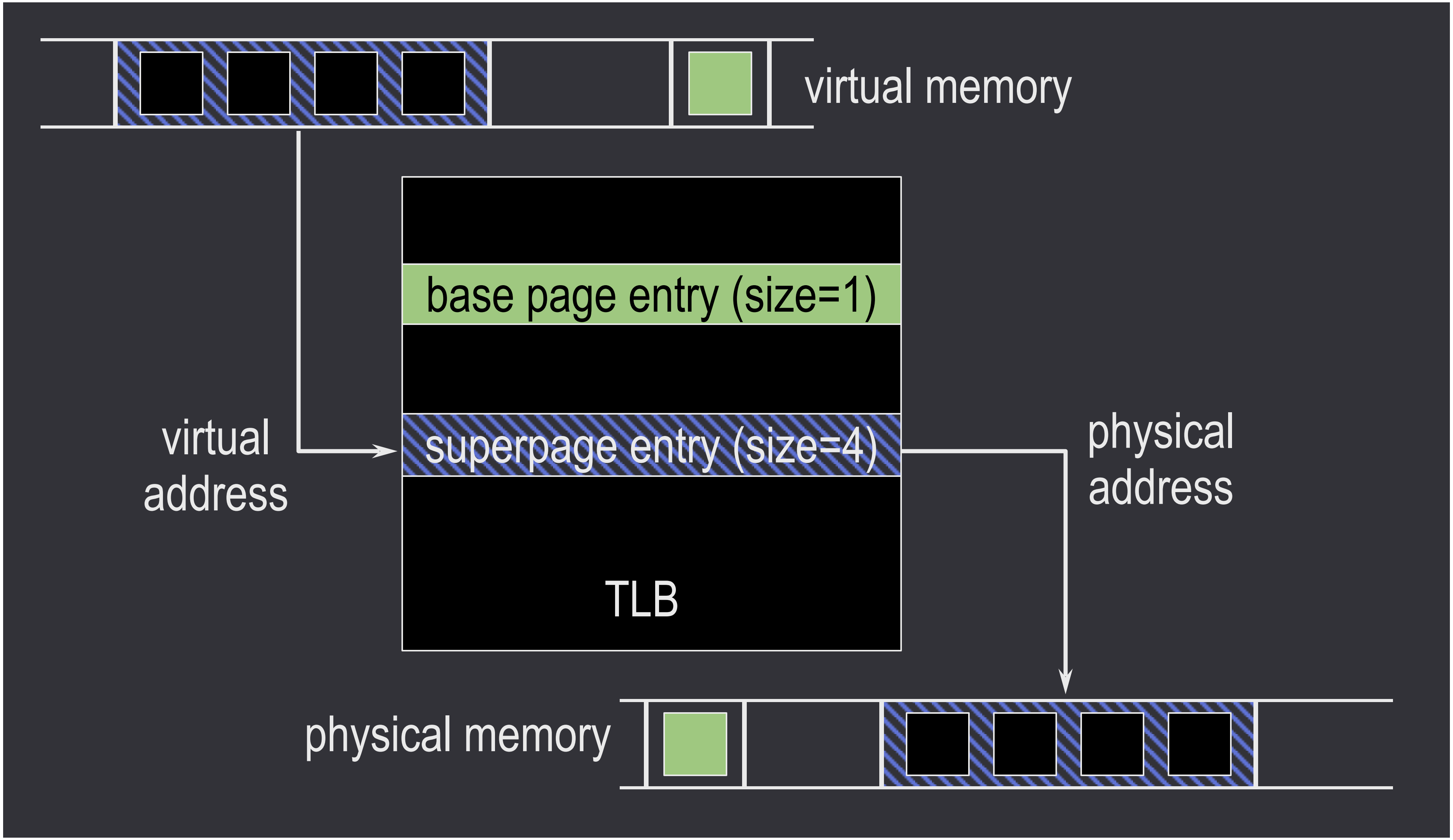
- Memory pages of larger sizes than standard pages
  - supported by most modern CPUs

- Superpage size = power of 2 x the base page size

- Only one TLB entry per superpage
  - But multiple (identical) page-table entries, one per base page

- Constraints:
  - contiguous (physically and virtually)
  - aligned (physically and virtually)
  - uniform protection attributes
  - one reference bit, one dirty bit

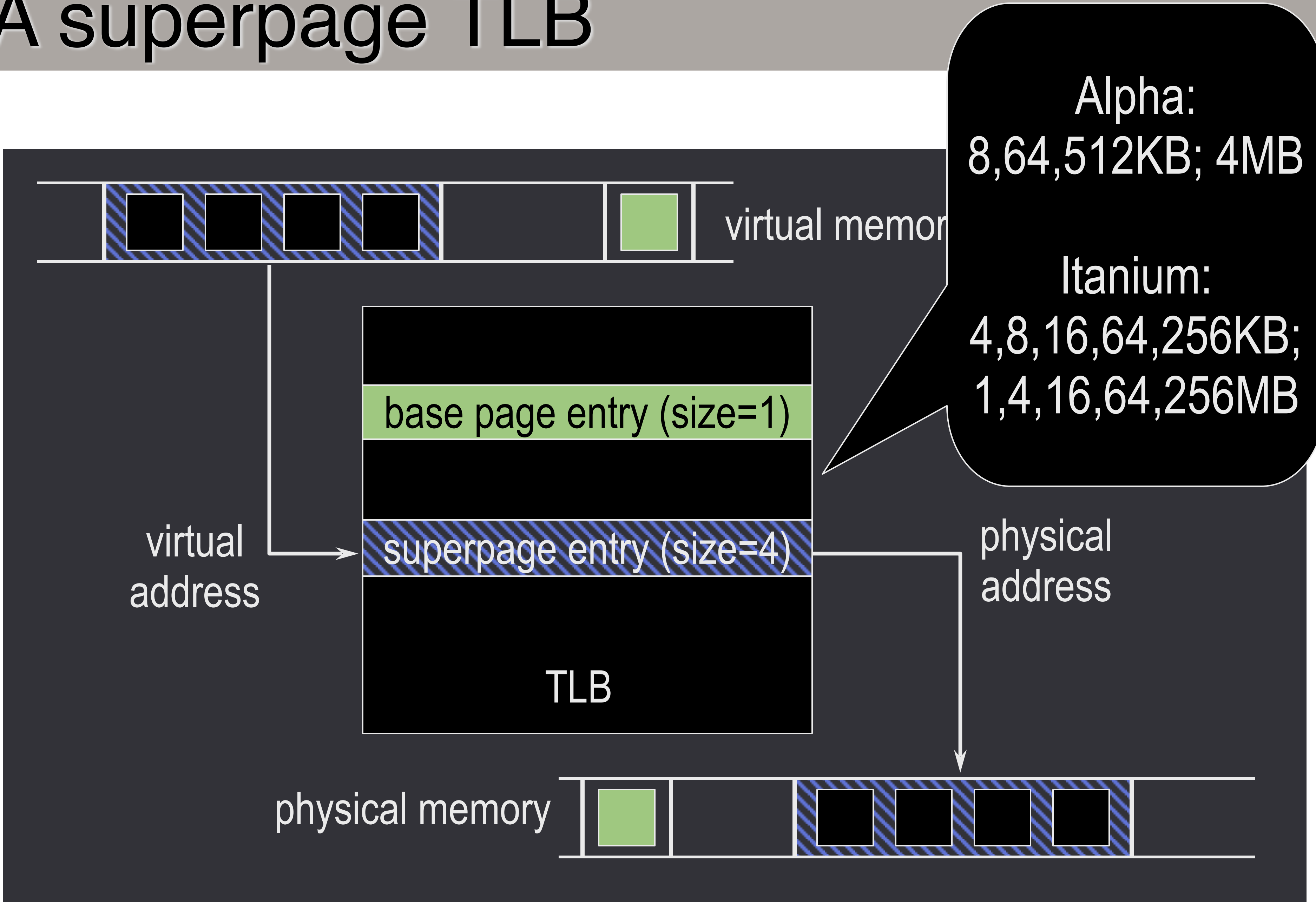
A superpage TLB



Examples of restrictions on superpage

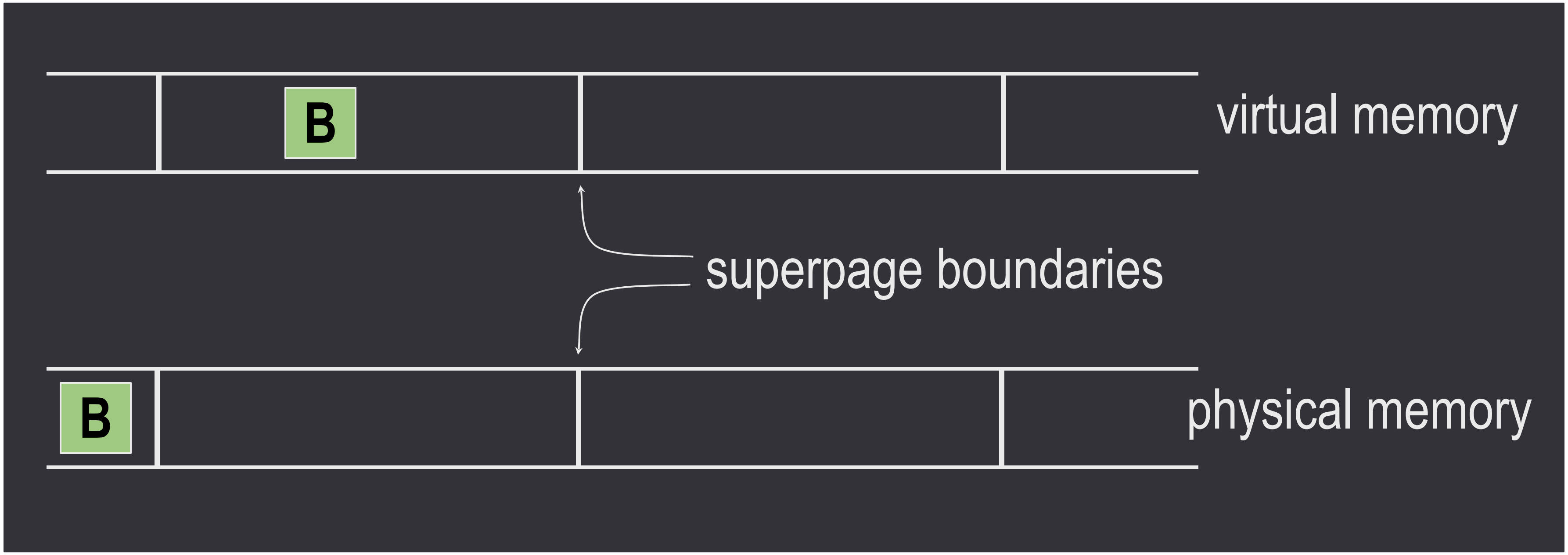
- **Size restrictions**
  - Possible
    - 8KiB = 2^1 x 4KiB
    - 16KiB = 2^2 x 4KiB
    - 32KiB = 2^3 x 4KiB etc
- **Contiguity restrictions for superpage of size 4**
  - Possible:
    - Base pages 8,9,10,11 in one superpage
  - Not possible:
    - Base pages 8, 10, 20, 22 in one superpage
- **Alignment restrictions for superpage of size 4**
  - Possible:
    - Base pages 4,5,6,7 in one superpage
    - Base pages 8,9,10,11 in one superpage
    - Base pages 12,13,14,15 in one superpage
  - Not possible:
    - Base pages 6,7,8,9 in one superpage
    - Base pages 13,14,15,16 in one superpage

A superpage TLB



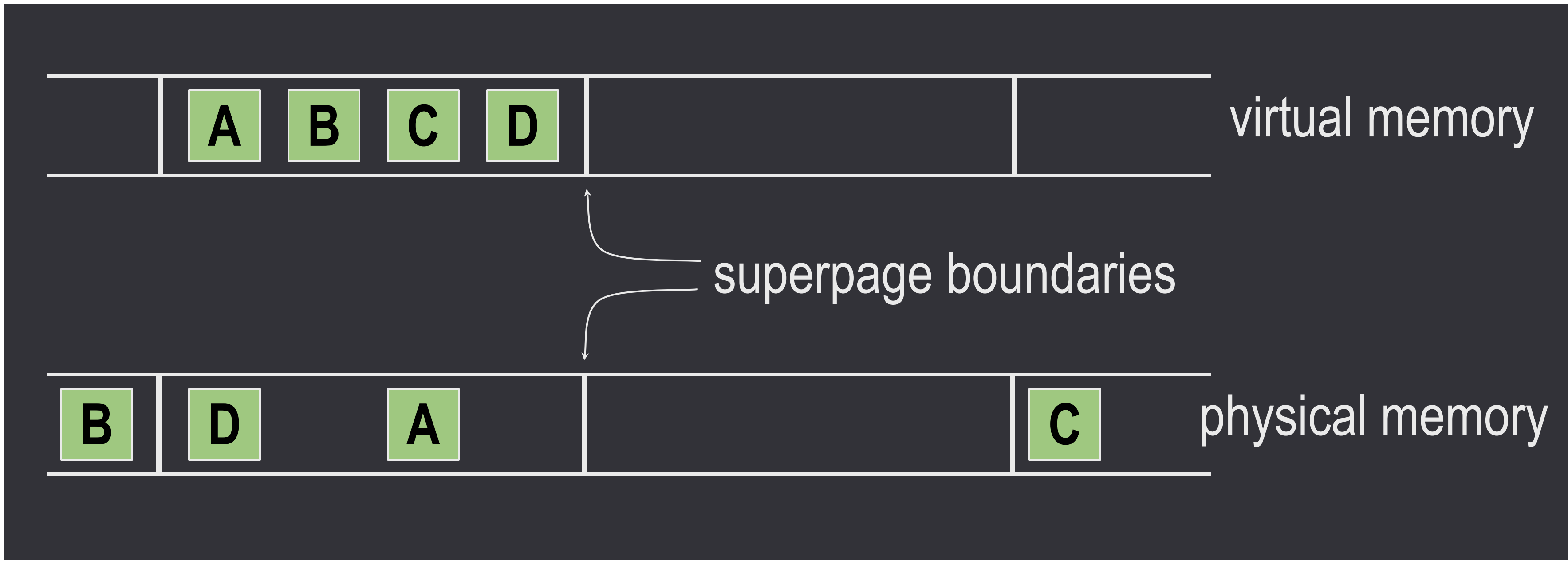
II
The superpage problem

Issue 1: superpage allocation



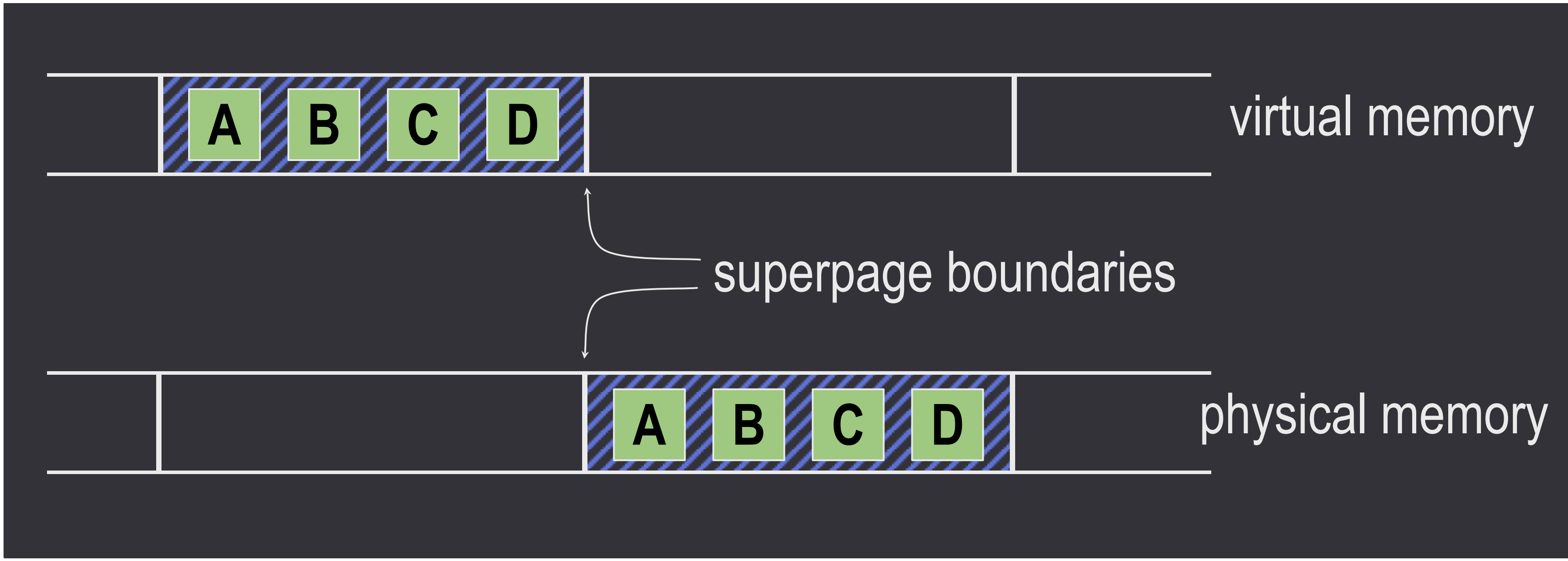
- How / when / what size to allocate?

Issue 1: superpage allocation



- How / when / what size to allocate?

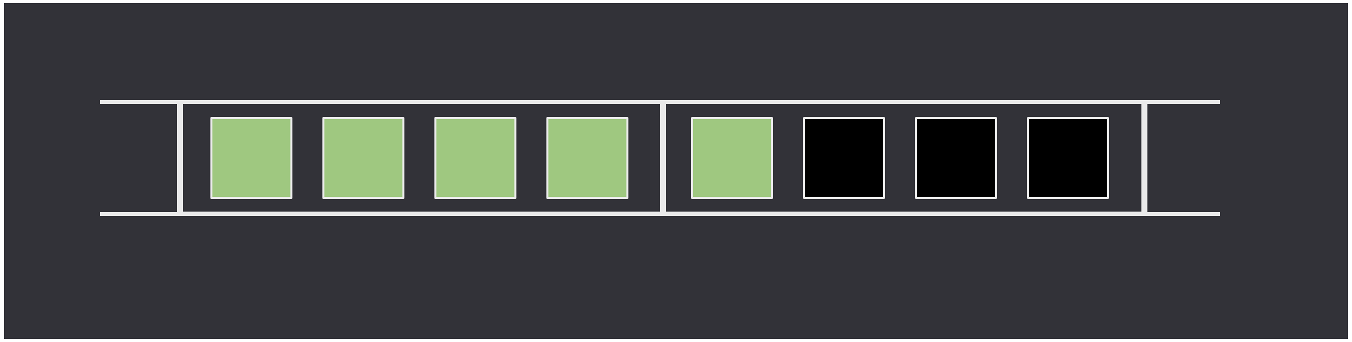
Issue 1: superpage allocation



- How / when / what size to allocate?

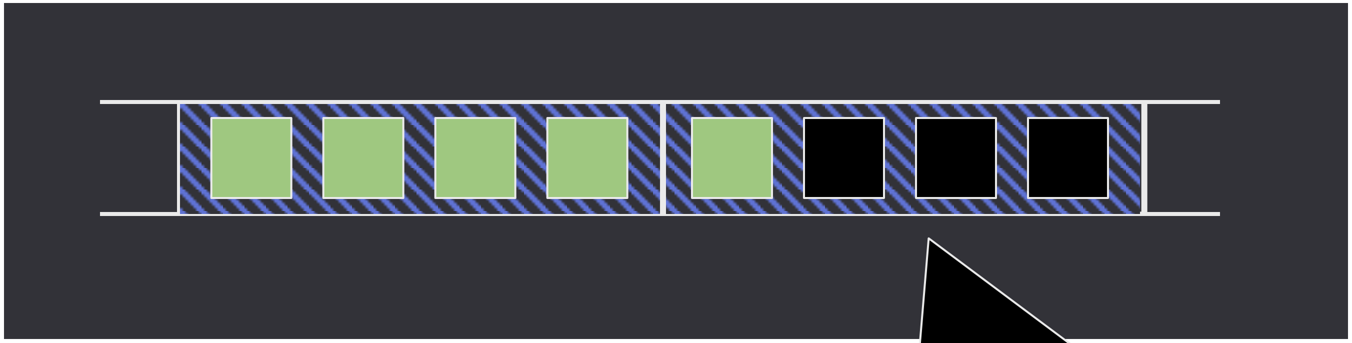
Issue 2: promotion

- 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

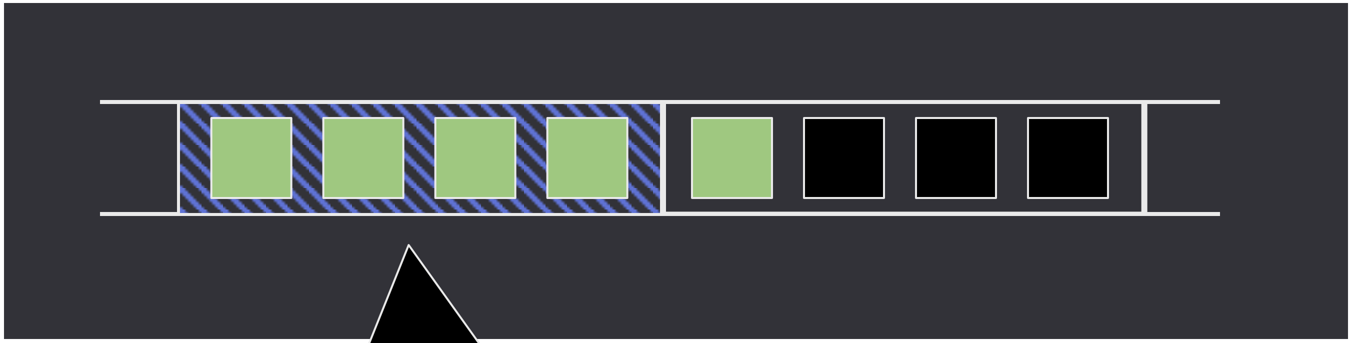
- Promotion: create a superpage out of a set of smaller pages
  - mark page table entry of each base page
- When to promote?



Wait for app to touch pages?
May lose opportunity to increase
TLB coverage.

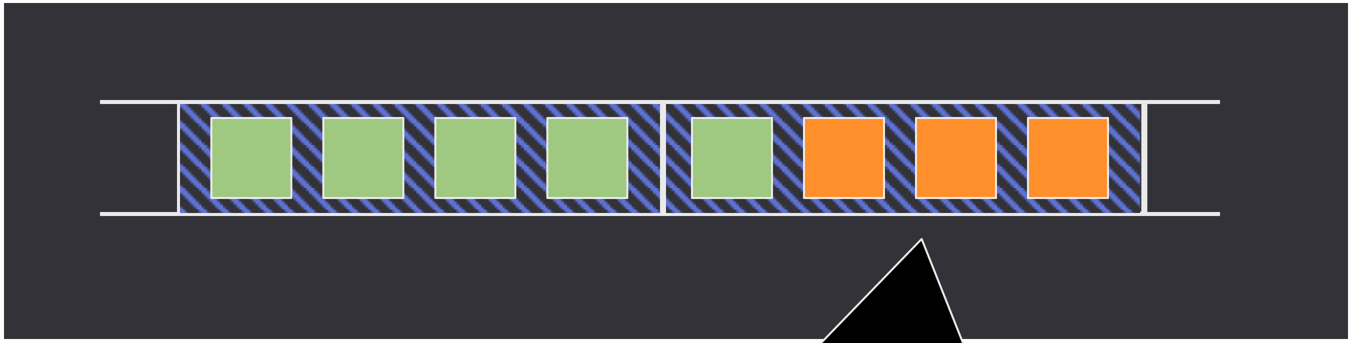
Issue 2: promotion

- 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?



Forcibly populate pages?
May incur I/O cost or increase internal fragmentation.

Issue 3: demotion

Demotion: convert a superpage into smaller pages

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

Issue 4: fragmentation

- Memory becomes externally fragmented due to
  - use of multiple page sizes
  - Scattered wired pages
    - Wired pages = pages that can’t be paged out to swap device
    - break contiguity of free base pages since they cannot be relocated.

- External fragmentation occurs at superpage sizes.
    - No external fragmentation at base page granularity

- Contiguity of free pages is a contended resource
  - Contiguous pages = pages that are next to each other
  - Allocating a superpage requires that sufficient number of contiguous base pages must be free.

- OS must
  - use contiguity restoration techniques
  - trade off impact of contiguity restoration against superpage benefits

Previous approaches

- Reservations
  - one superpage size only
- Relocation
  - move pages at promotion time
  - must recover copying costs
- Eager superpage creation (IRIX, HP-UX)
  - size specified by user: non-transparent
- Hardware support
  - Contiguous virtual superpage mapped to discontiguous physical base pages
- Demotion issues not addressed
  - large pages partially dirty/referenced

III
Design

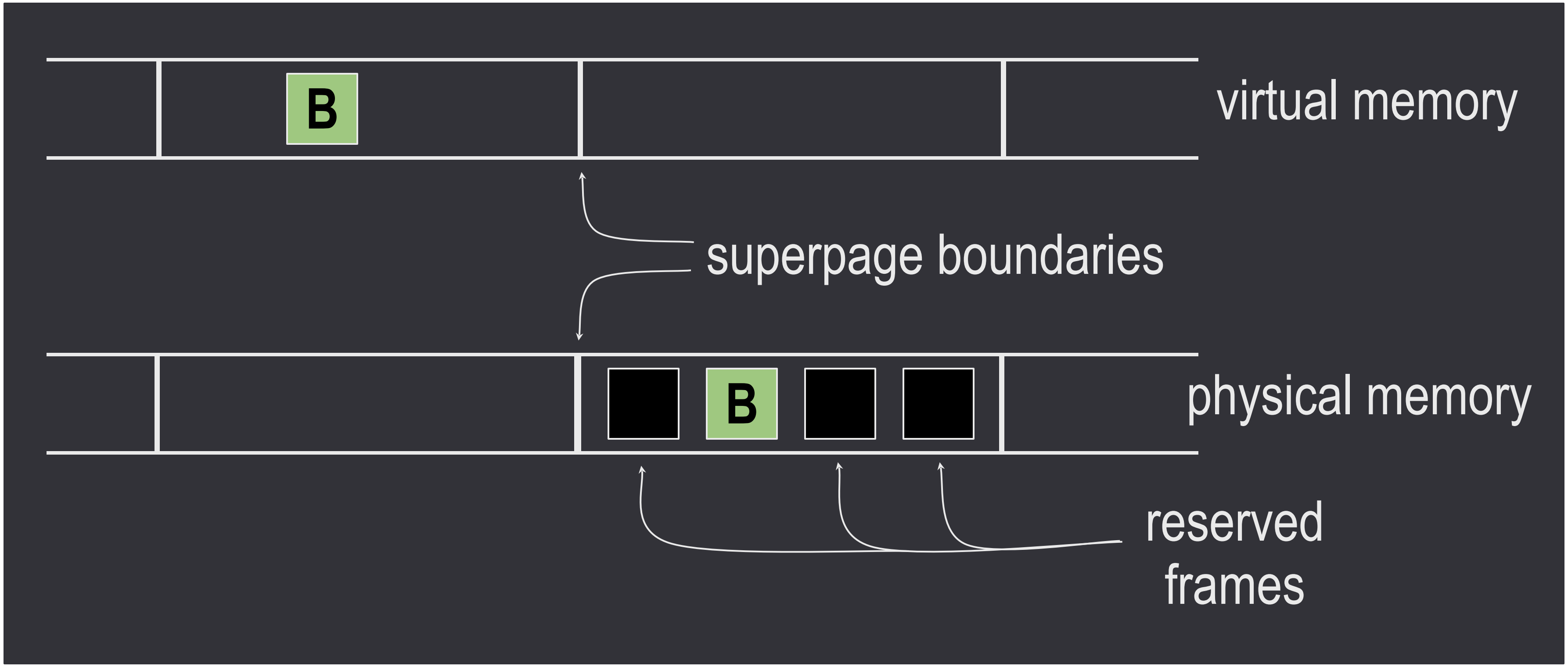
Key observation

Once an application touches the first page of a memory object then it is likely that it will quickly touch every page of that object

- Example: array initialization
- Opportunistic policies
  - superpages as large and as soon as possible
  - as long as no penalty if wrong decision

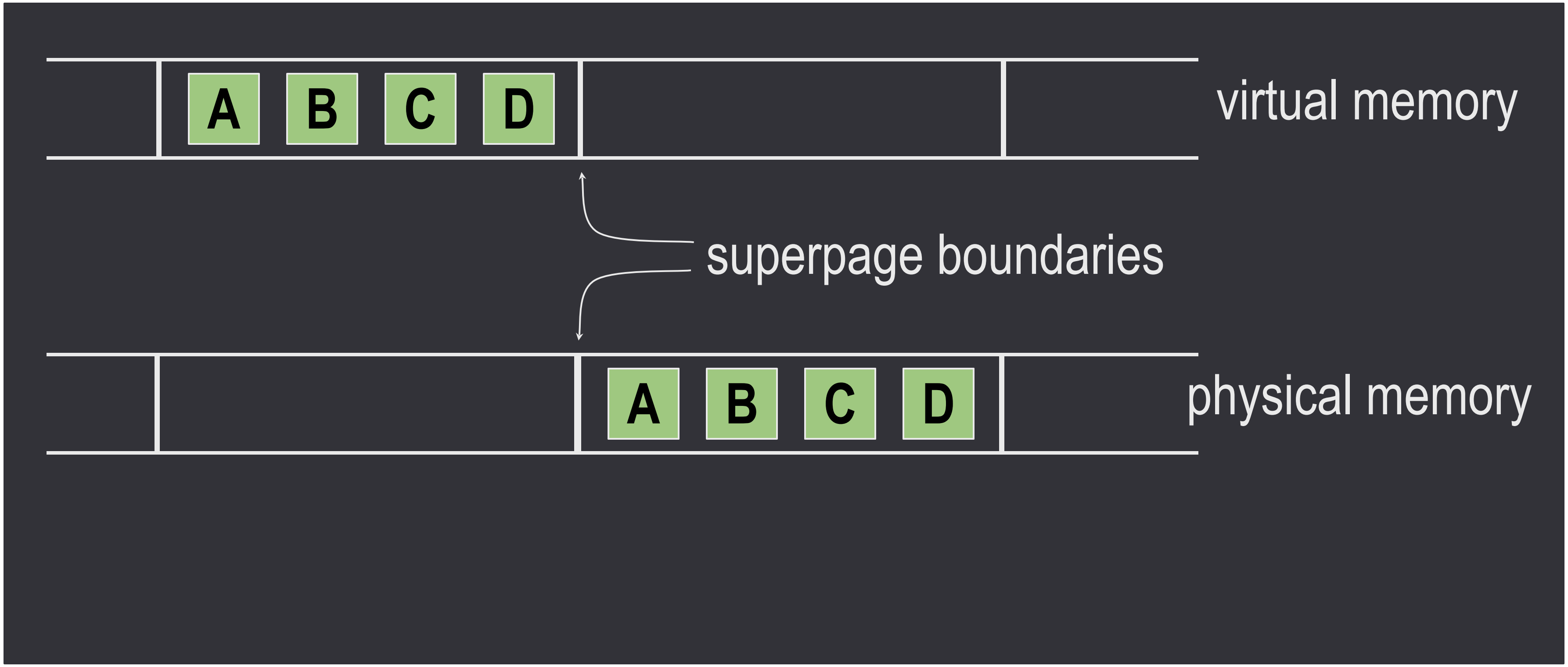
Superpage allocation

Preemptible reservations



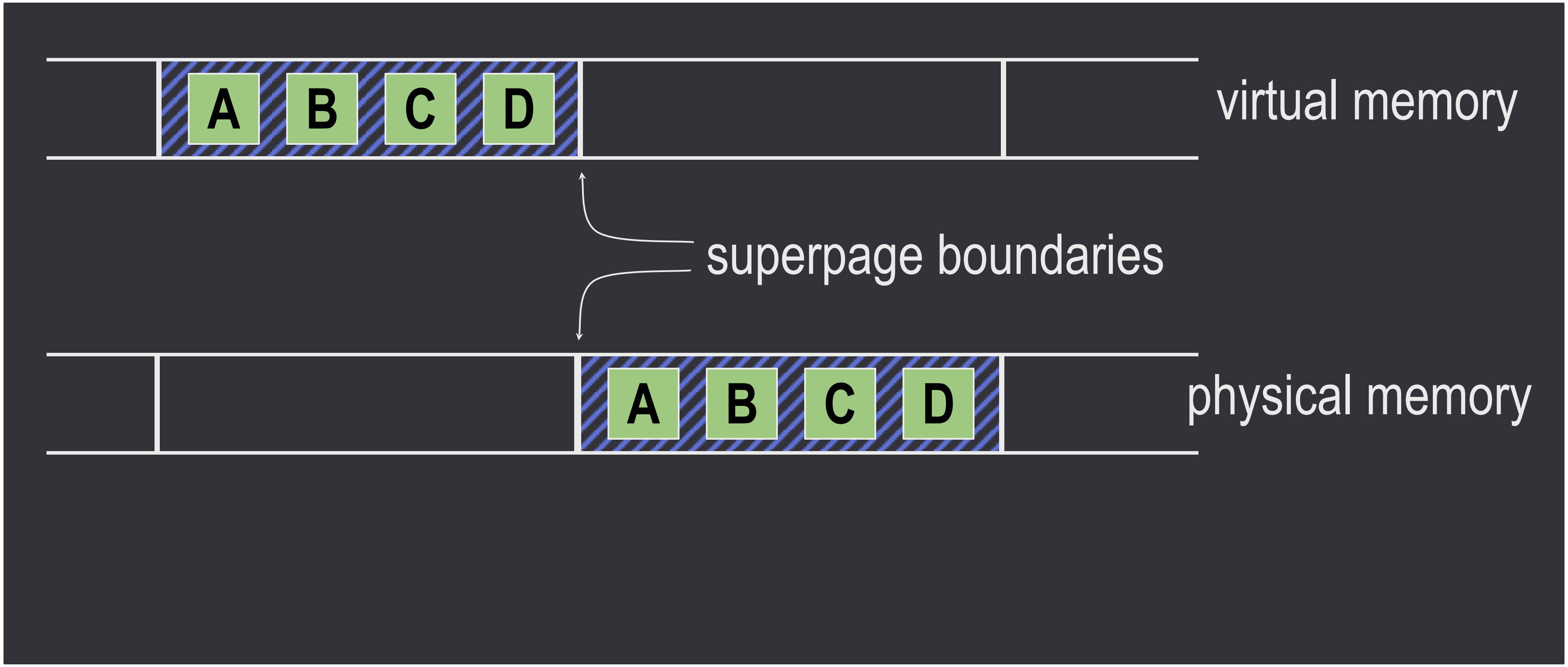
Superpage allocation

Preemptible reservations



Superpage allocation

Preemptible reservations

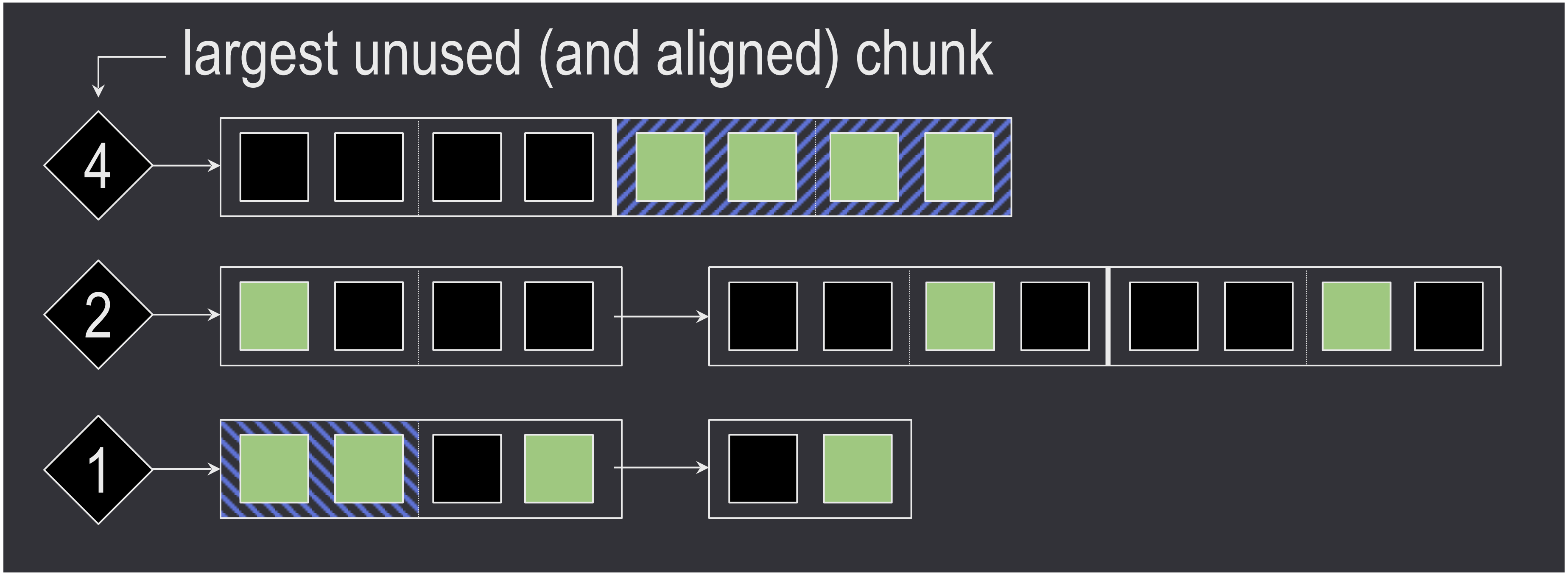


Allocation: reservation size

Opportunistic policy

- Go for biggest size that is no larger than the memory object (e.g., file)
- If required size not available, try preemption before resigning to a smaller size
  - preempted reservation had its chance

Allocation: managing reservations



best candidate for preemption at front:

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

Incremental promotions

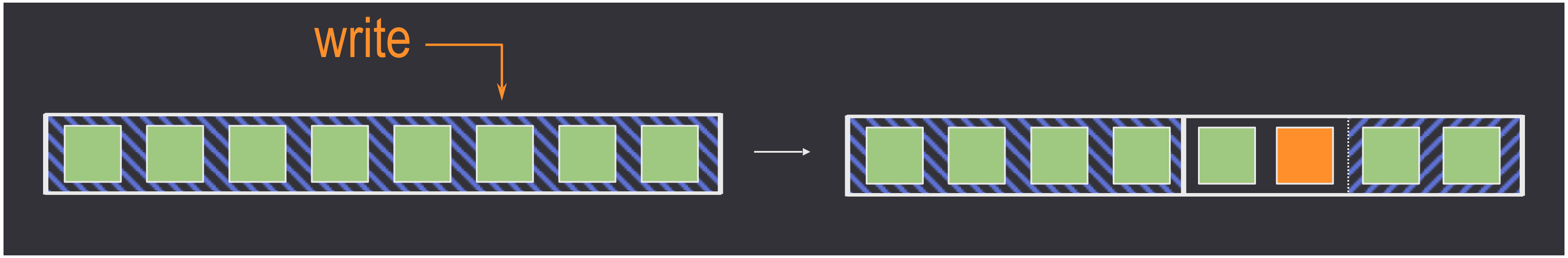
Promotion policy: opportunistic

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
- Demote also when the page daemon selects a base page as a victim page.

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 for victim selection
  - restore contiguity
    - move clean, inactive pages to the free list
  - minimize impact
    - prefer victim pages that contribute the most to contiguity

- Cluster wired pages
  - Assign a dedicated region of physical memory for wired pages
  - So that they break contiguity for superpage allocations from rest of the memory.

IV
Experimental evaluation

Experimental setup

- FreeBSD 4.3
- Alpha 21264, 500 MHz, 512 MB RAM
- 8 KB, 64 KB, 512 KB, 4 MB pages
- 128-entry DTLB, 128-entry ITLB
- Unmodified applications

Best-case benefits

- TLB miss reduction usually above 95%
- SPEC CPU2000 integer
  - 11.2% improvement (0 to 38%)
- SPEC CPU2000 floating point
  - 11.0% improvement (-1.5% to 83%)
- Other benchmarks
  - FFT (200^3 matrix): 55%
  - 1000x1000 matrix transpose: 655%
- 30%+ in 8 out of 35 benchmarks

Why multiple superpage sizes

	64KB	512KB	4MB	All
FFT	1%	0%	55%	55%
galgel	28%	28%	1%	29%
mcf	24%	31%	22%	68%

Improvements with only one superpage size vs. all sizes

Conclusions

- Superpages
  - OS can provide transparent support for a mix of superpages by applications.
- Contiguity restoration is necessary
  - sustains benefits; low impact
- Multiple page sizes are important
  - scales to very large superpages

More references:

- Multiple page sizes in different processors
  - https://en.wikipedia.org/wiki/Page_(computer_memory)#Multiple_page_sizes
- Linux Transparent Hugepages
  - https://lwn.net/Articles/423584/