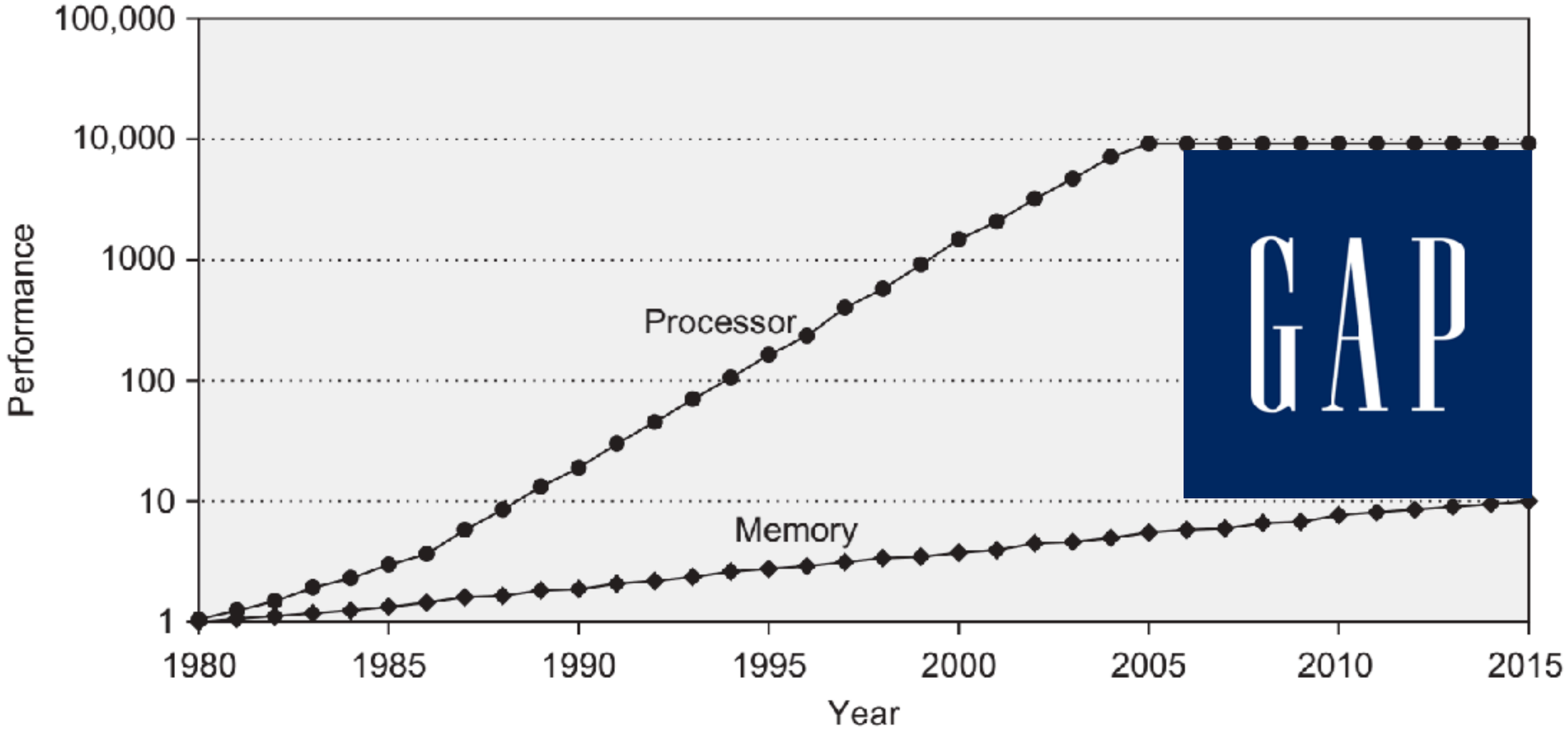
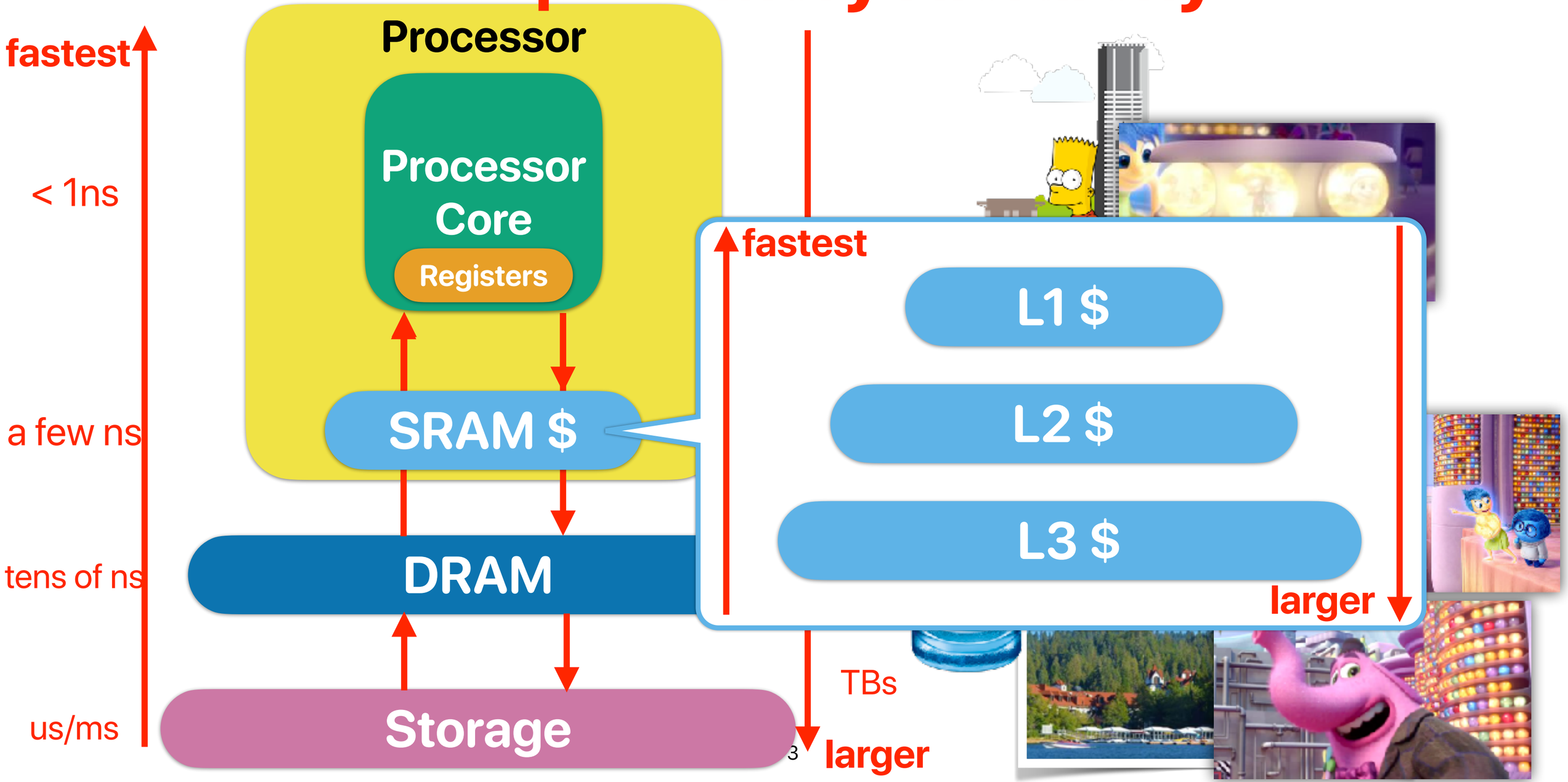
Virtual Memory (cont.)

Hung-Wei Tseng

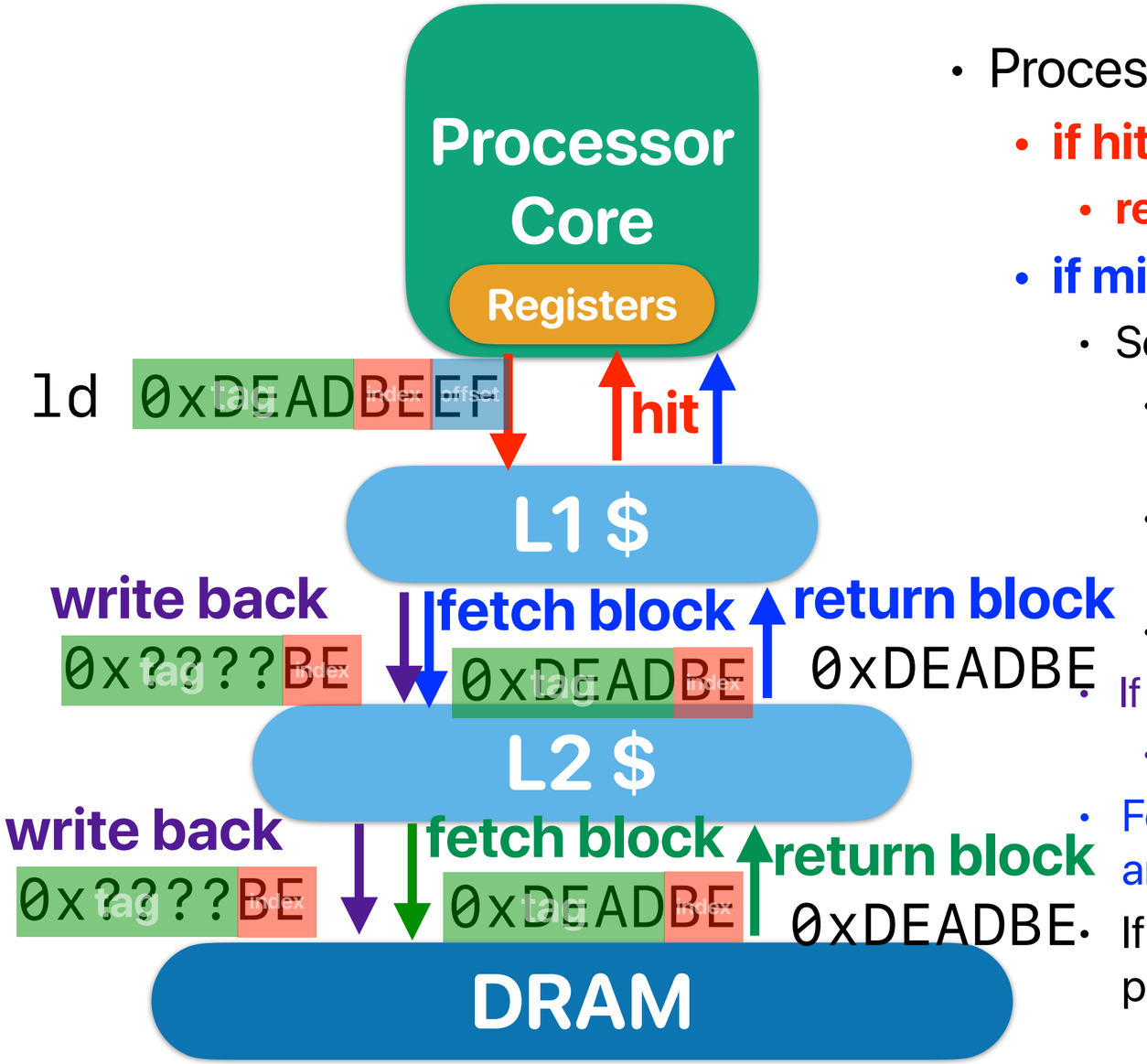
Performance gap between Processor/Memory



Recap: Memory Hierarchy

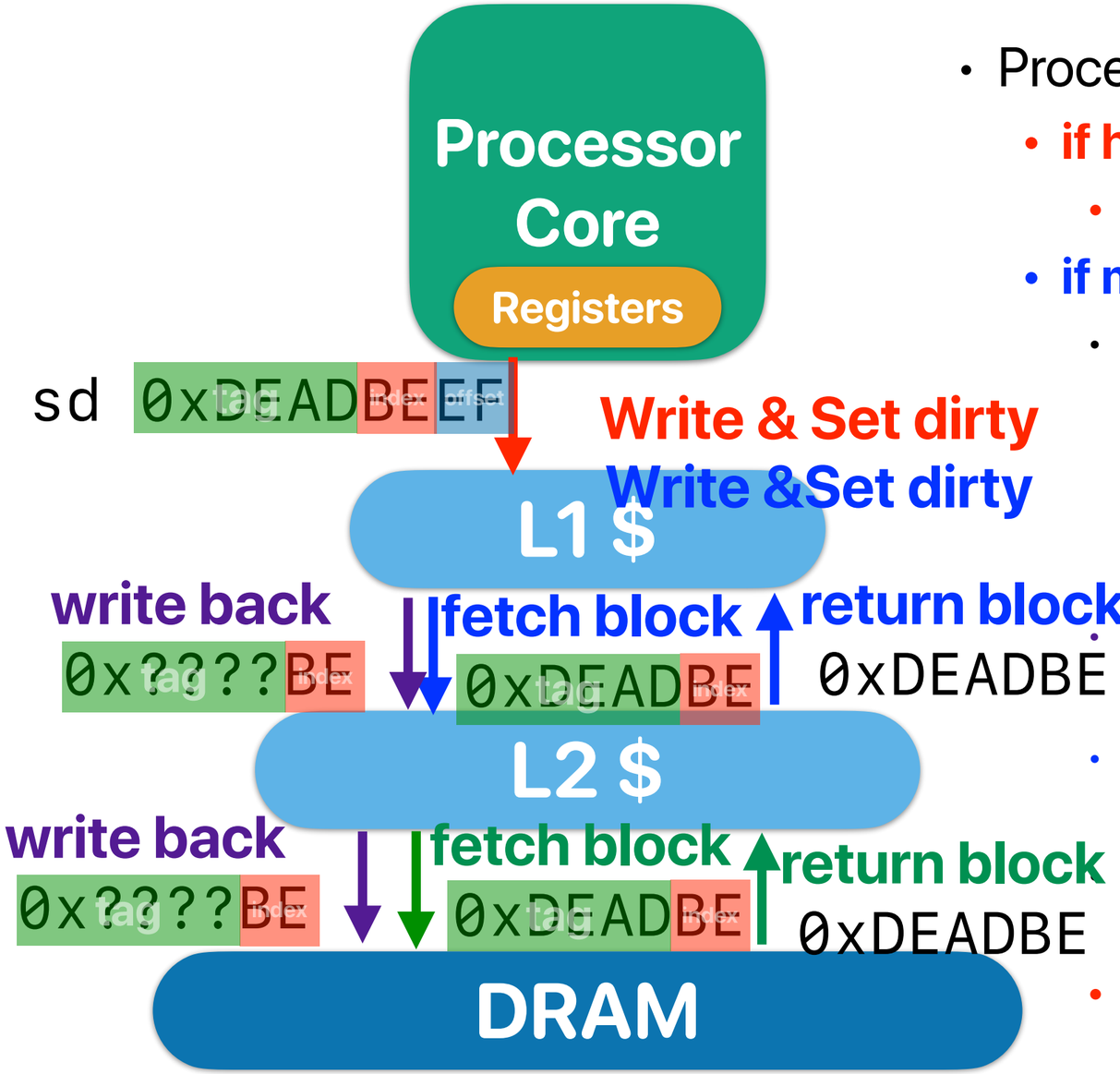


What happens when we read data



- Processor sends load request to L1-$
  - **if hit**
    - **return data**
  - **if miss**
    - Select a victim block
      - If the target "set" is not full — select an empty/invalidated block as the victim block
      - If the target "set is full — select a victim block using some policy
      - LRU is preferred — to exploit temporal locality!
    - If the victim block is "dirty" & "valid"
      - **Write back** the block to lower-level memory hierarchy
    - Fetch the requesting block from lower-level memory hierarchy and place in the victim block
    - If write-back or fetching causes any miss, repeat the same process

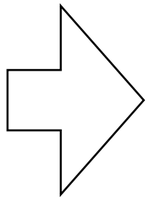
What happens when we write data



- Processor sends load request to L1-$
  - **if hit**
    - **return data — set DIRTY**
  - **if miss**
    - Select a victim block
      - If the target "set" is not full — select an empty/invalidated block as the victim block
      - If the target "set is full — select a victim block using some policy
      - LRU is preferred — to exploit temporal locality!
    - If the victim block is "dirty" & "valid"
      - **Write back** the block to lower-level memory hierarchy
    - Fetch the requesting block from lower-level memory hierarchy and place in the victim block
    - If write-back or fetching causes any miss, repeat the same process
  - **Present the write "ONLY" in L1 and set DIRTY**

Recap: Matrix Multiplication

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```



```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}

// blocking algorithm
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

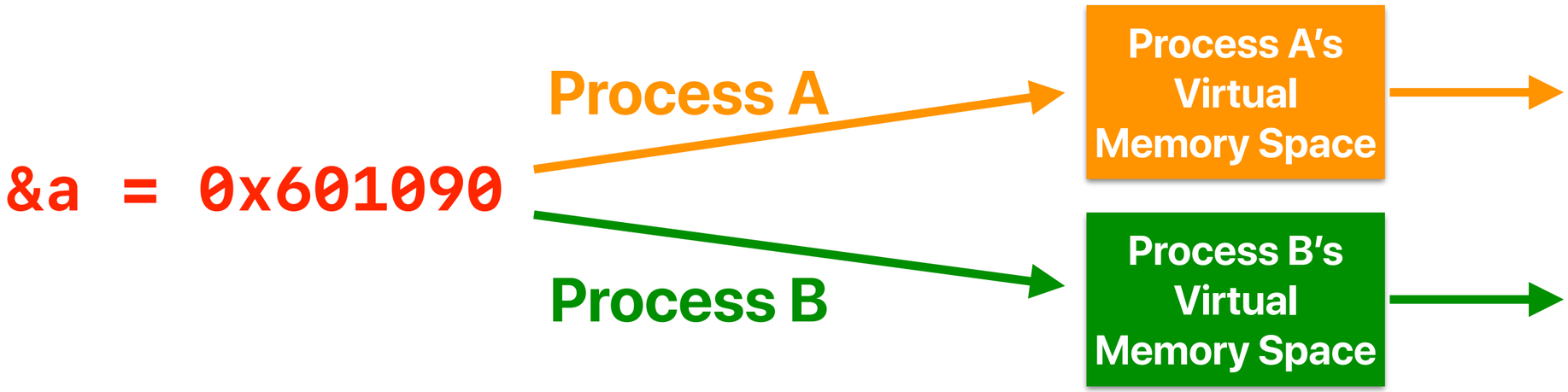
Recap: Summary of Optimizations

- Software
  - Data layout — capacity miss, conflict miss, compulsory miss
  - Blocking — capacity miss/conflict miss
  - Transpose — conflict miss
  - Loop fission — conflict miss — when $ has limited way associativity
  - Loop fusion — capacity miss — when $ has enough way associativity
  - Loop interchange — conflict/capacity miss
- Hardware
  - Prefetch — compulsory miss
  - Write buffer — miss penalty
  - Bank/pipeline — miss penalty
  - Critical word first and early restart — miss panelty

Recap: Virtual memory

- An **abstraction** of memory space available for programs/software/programmer
- Programs execute using virtual memory address
- The operating system and hardware work together to handle the mapping between virtual memory addresses and real/physical memory addresses
- Virtual memory organizes memory locations into "**pages**"

Recap: Demo revisited




```c
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sched.h>
#include <sys/syscall.h>
#include <time.h>

double a;

int main(int argc, char *argv[])
{
    int i, number_of_total_processes=4;
    number_of_total_processes = atoi(argv[1]);
    for(i = 0; i< number_of_total_processes-1 && fork(); i++);
    srand((int)time(NULL)+(int)getpid());
    fprintf(stderr, "\nProcess %d. Value of a is %lf and address of a is %p\n",getpid(), a, &a);
    sleep(10);
    fprintf(stderr, "\nProcess %d. Value of a is %lf and address of a is %p\n",getpid(), a, &a);
    return 0;
}
```

Recap: Why Virtual memory?

- Allowing multiple applications to share physical main memory
  - Memory protection/isolation among programs/processes is automatically achieved
- Allowing applications to work even the installed physical memory or available physical memory is smaller than the working set of the application
  - Programmer does not need to worry about the physical memory capacity of different machines — make compiled program compatible
  - Multiple programs can work concurrently even through their total memory demand is larger than the installed physical memory

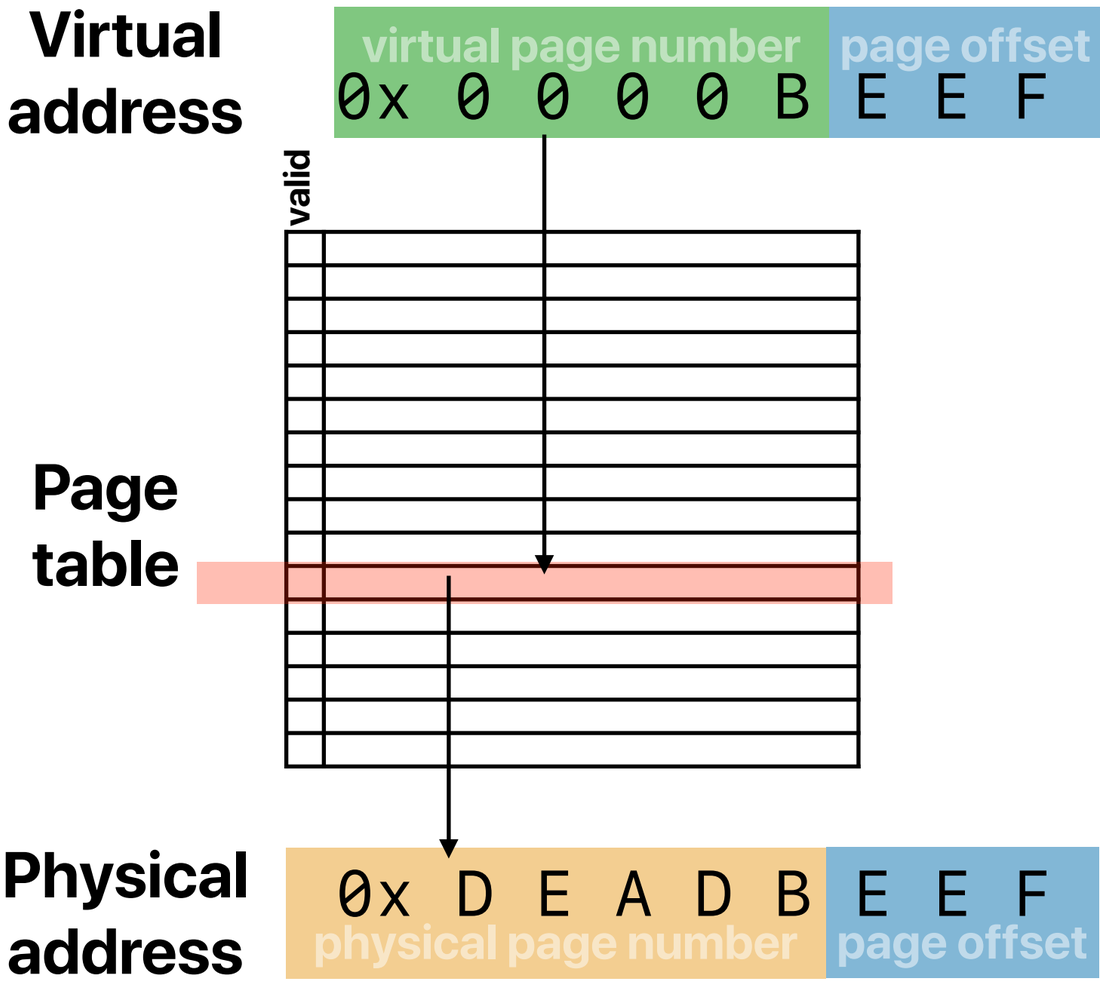
Outline

- Virtual memory
- Architectural support for virtual memory
- Advanced hardware support for virtual memory

Virtual Memory (cont.)

Address translation

- Processor receives virtual addresses from the running code, main memory uses physical memory addresses
- Virtual address space is organized into “pages”
- The system references the **page table** to translate addresses
  - Each process has its own page table
  - The page table content is maintained by OS



Size of page table

- Assume that we have **64-bit** virtual address space, each page is 4KB, each page table entry is 8 Bytes, what magnitude in size is the page table for a process?
  A. MB — 2^{20} Bytes
  B. GB — 2^{30} Bytes
  C. TB — 2^{40} Bytes
  D. PB — 2^{50} Bytes
  E. EB — 2^{60} Bytes

Size of page table

- Assume that we have **64-bit** virtual address space, each page is 4KB, each page table entry is 8 Bytes, what magnitude in size is the page table for a process?
  A. MB — 2^{20} Bytes
  B. GB — 2^{30} Bytes
  C. TB — 2^{40} Bytes
  D. PB — 2^{50} Bytes
  E. EB — 2^{60} Bytes

$$\frac{2^{64}\ Bytes}{4\ KB} \times 8\ Bytes = 2^{55}\ Bytes = 32\ PB$$

If you still don't know why — you need to take CS202

Conventional page table

0x0 0xFFFFFFFFFFFFFFFF

Virtual Address Space

— must be consecutive in the physical memory

— need a big segment! — difficult to find a spot

— simply too big to fit in memory if address space is large!

$\frac{2^{64}\ B}{2^{12}\ B}$ page table entries/leaf nodes

Do we really need a large table?

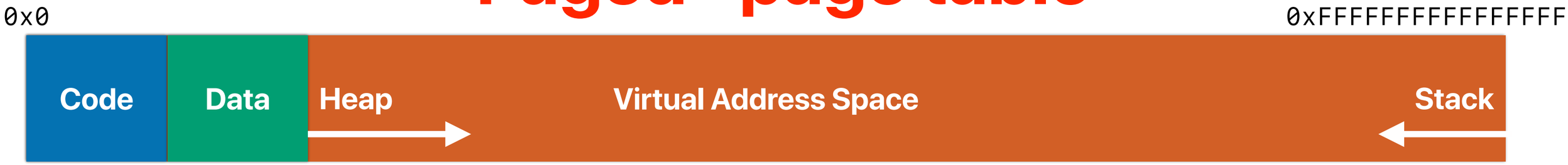
1TB — only 0.000006% of 2^{64} Bytes

Your program probably never uses this huge area!

$$\frac{2^{64}\ B}{2^{12}\ B}$$ page table entries/leaf nodes

Why bother presenting these nodes?

"Paged" page table

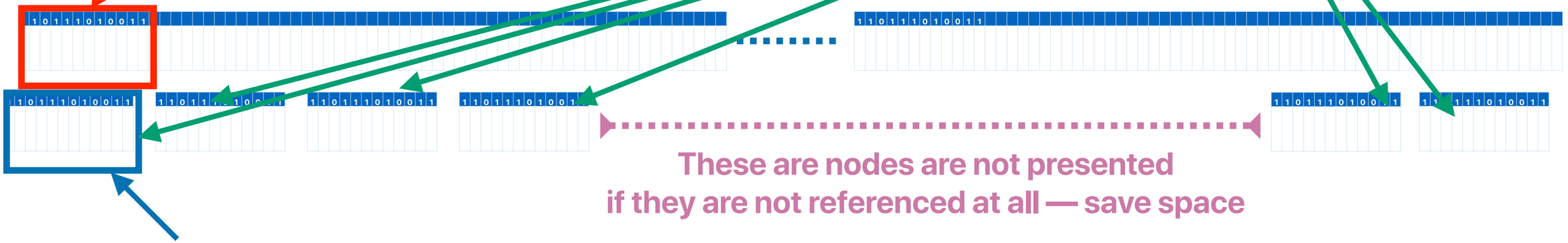


Break up entries into pages!
Each of these occupies exactly a page

$$\frac{2^{12}\ B}{2^{3}\ B} = 2^{9}$$ **PTEs per node**

Question:
These nodes are spread out, how to locate them in the memory?

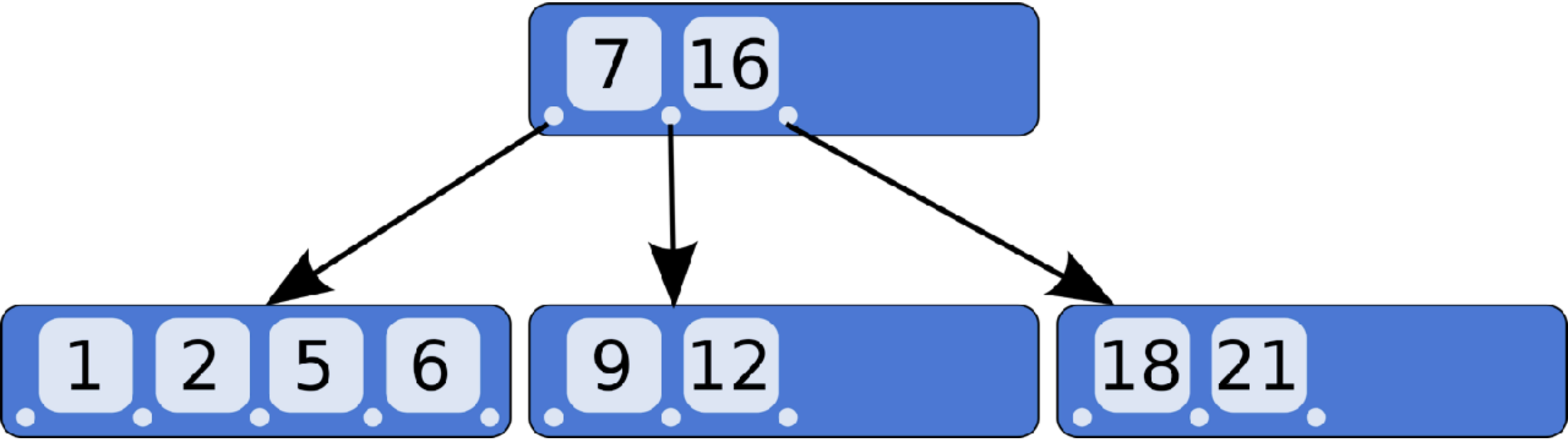
Otherwise, you always need to find more than one consecutive pages — difficult!



These are nodes are not presented if they are not referenced at all — save space

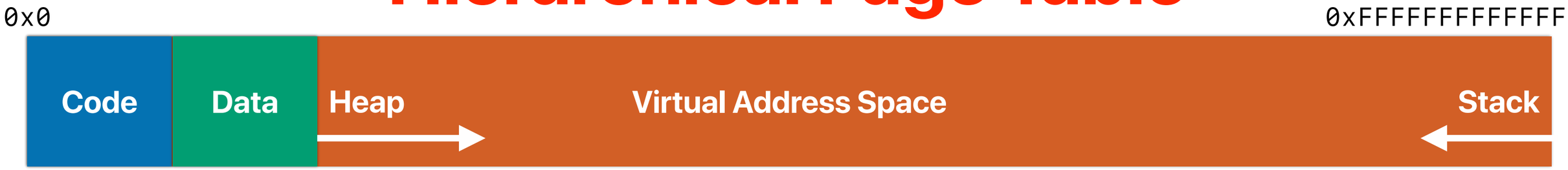
Allocate page table entry nodes "on demand"

B-tree



https://en.wikipedia.org/wiki/B-tree#/media/File:B-tree.svg

Hierarchical Page Table

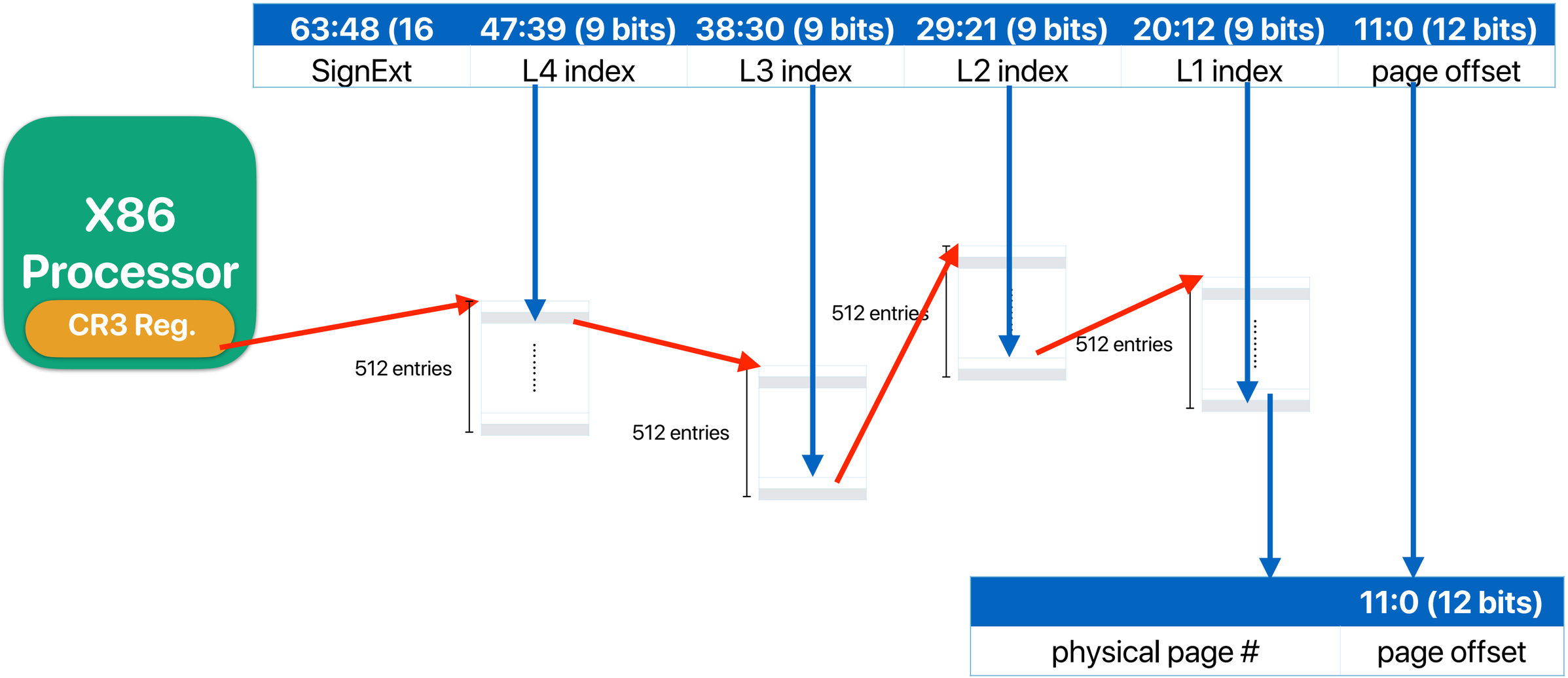


$$\lceil log_{2^9} \frac{2^{64}\ B}{2^{12}\ B} \rceil = \lceil log_{2^9} 2^{52} \rceil = 6 \text{ levels}$$

These are nodes are not presented as they are not referenced at all.

$\frac{2^{64}\ B}{2^{12}\ B}$ page table entries/leaf nodes (worst case)

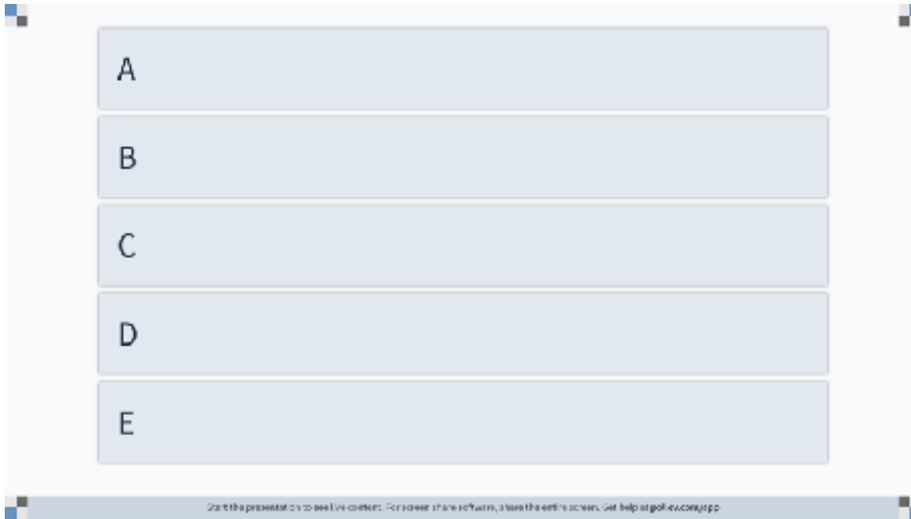
Address translation in x86-64



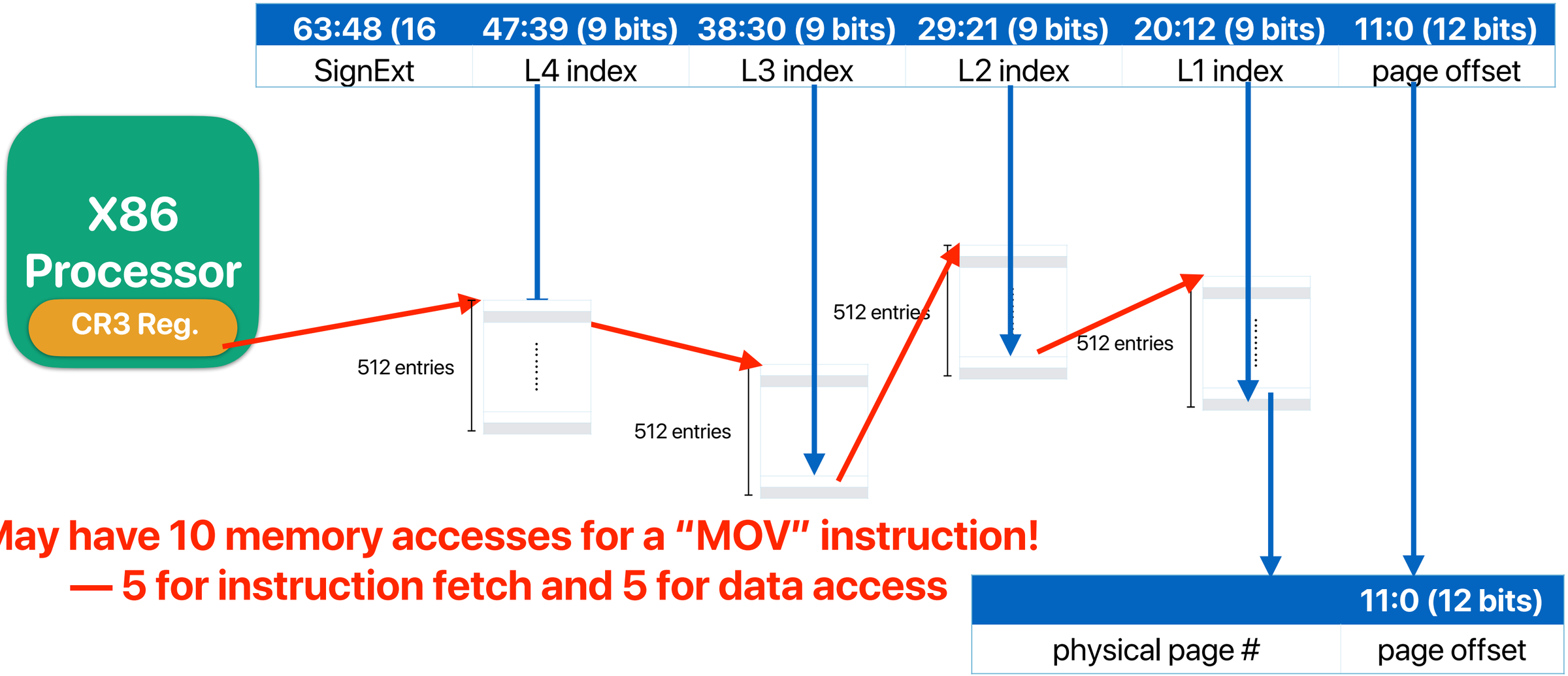
Translation Caching: Skip, Don't Walk (the Page Table). Thomas W. Barr, Alan L. Cox, Scott Rixner

When we have virtual memory…

- If an x86 processor supports virtual memory through the basic format of the page table as shown in the previous slide, how many memory accesses can a **`mov`** instruction that access data memory once incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. 10



Address translation in x86-64



May have 10 memory accesses for a "MOV" instruction!

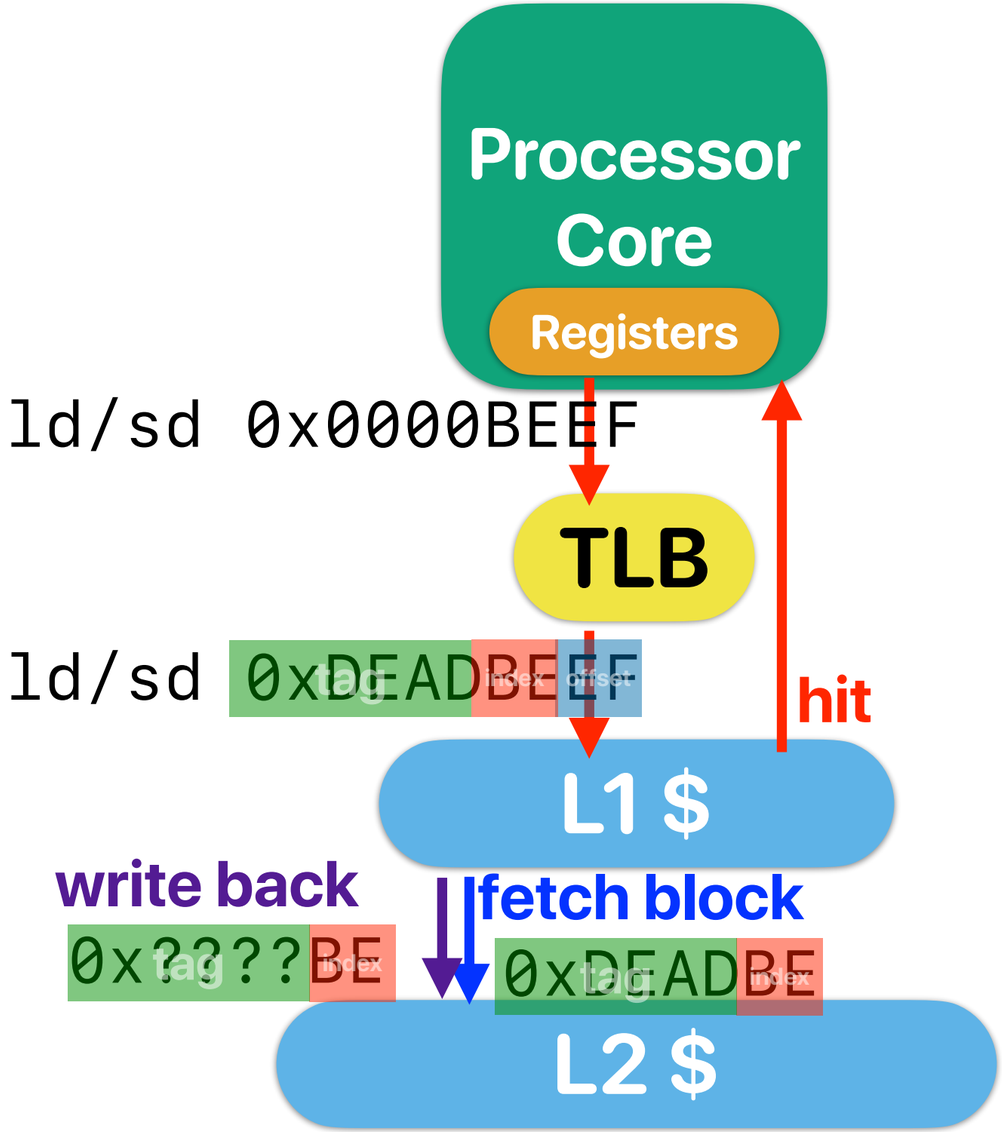
— 5 for instruction fetch and 5 for data access

When we have virtual memory…

- If an x86 processor supports virtual memory through the basic format of the page table as shown in the previous slide, how many memory accesses can a **`mov`** instruction that access data memory once incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. 10

Avoiding the address translation overhead

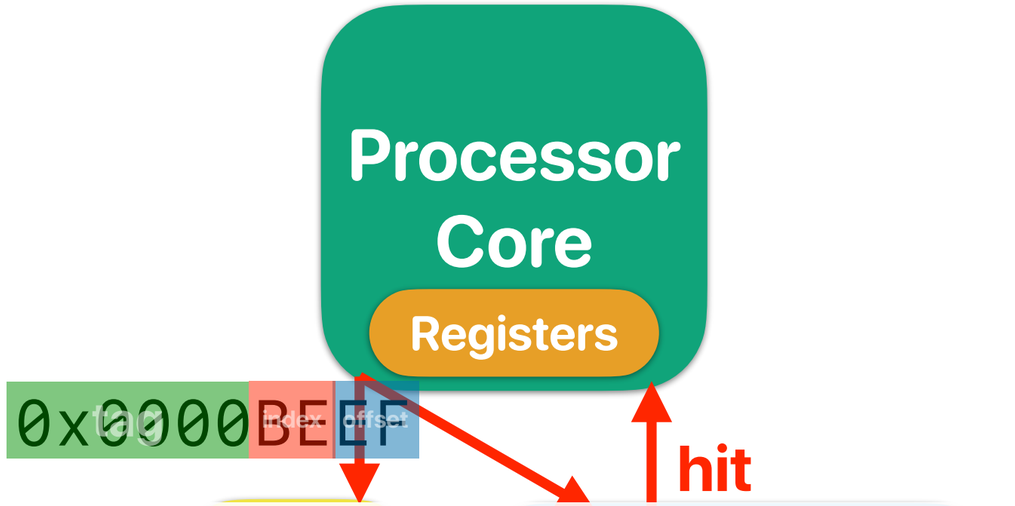
TLB: Translation Look-aside Buffer



- TLB — a small SRAM stores frequently used page table entries
- Good — A lot faster than having everything going to the DRAM
- Bad — Still on the critical path

TLB + Virtual cache

- L1 $ accepts virtual address — you don't need to translate
- Good — you can access both TLB and L1-$ at the same time and physical address is only needed if L1-$ misses
- Bad — it doesn't work in practice
  - Many applications have the same virtual address but should be pointing different **physical addresses**
  - An application can have "aliasing virtual addresses" pointing to the same **physical address**



You really need "physical address" to judge if that's what you want

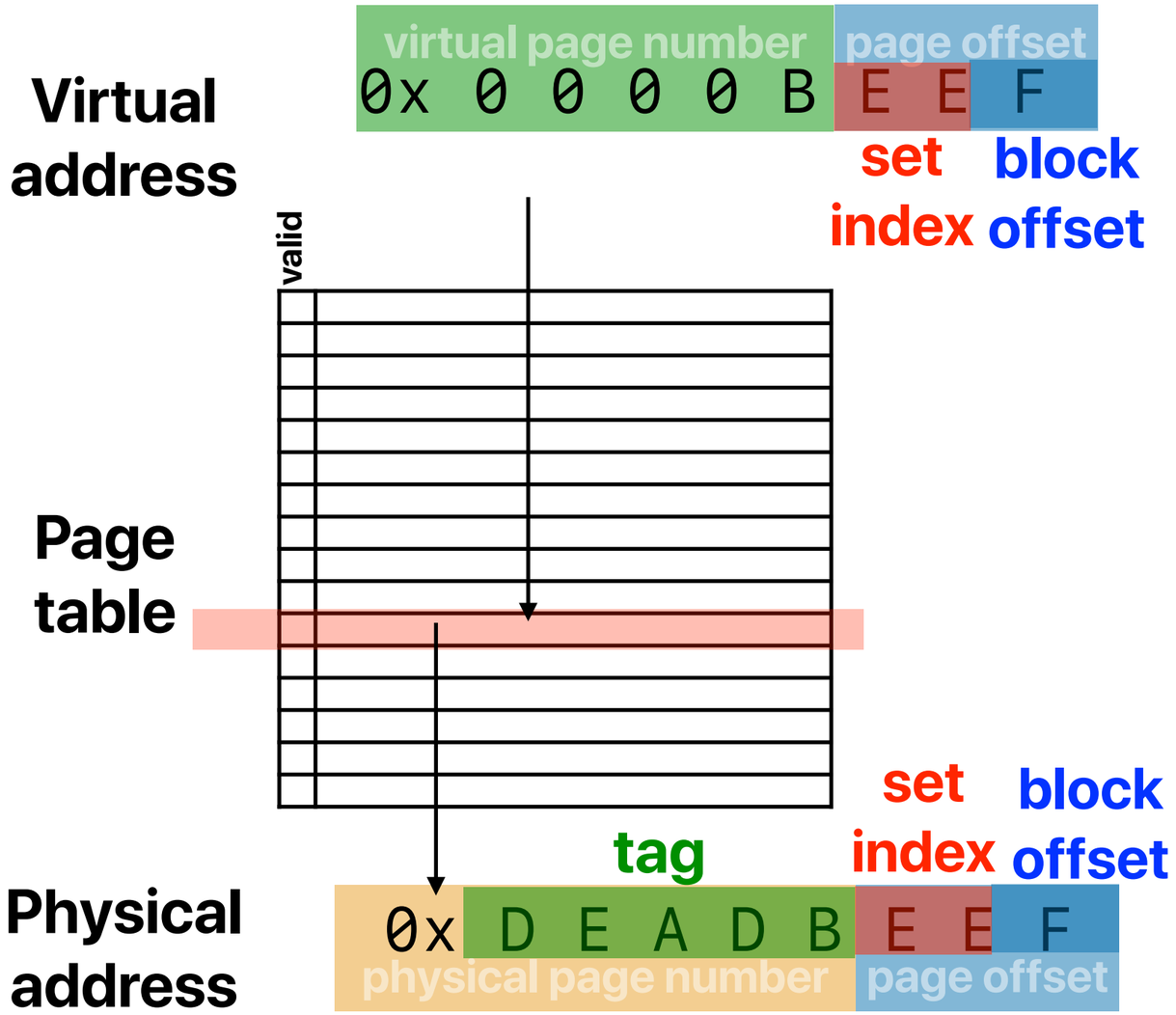
Virtually indexed, physically tagged cache

- Can we find physical address directly in the virtual address

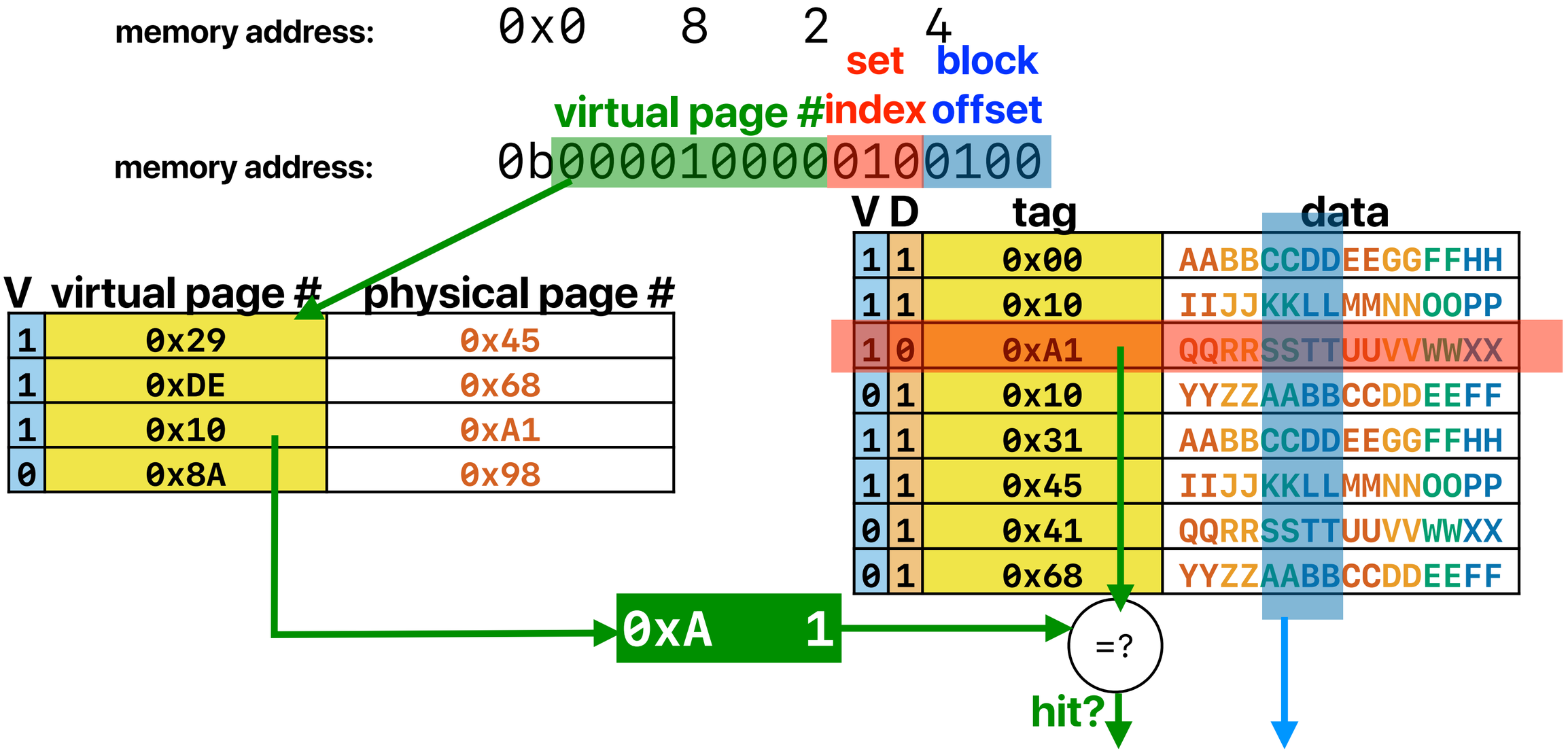
  — Not everything — but the page offset isn't changing!

- Can we indexing the cache using the "partial physical address"?

  — Yes — Just make set index + block set to be exactly the page offset



Virtually indexed, physically tagged cache



Virtually indexed, physically tagged cache

- If page size is 4KB —

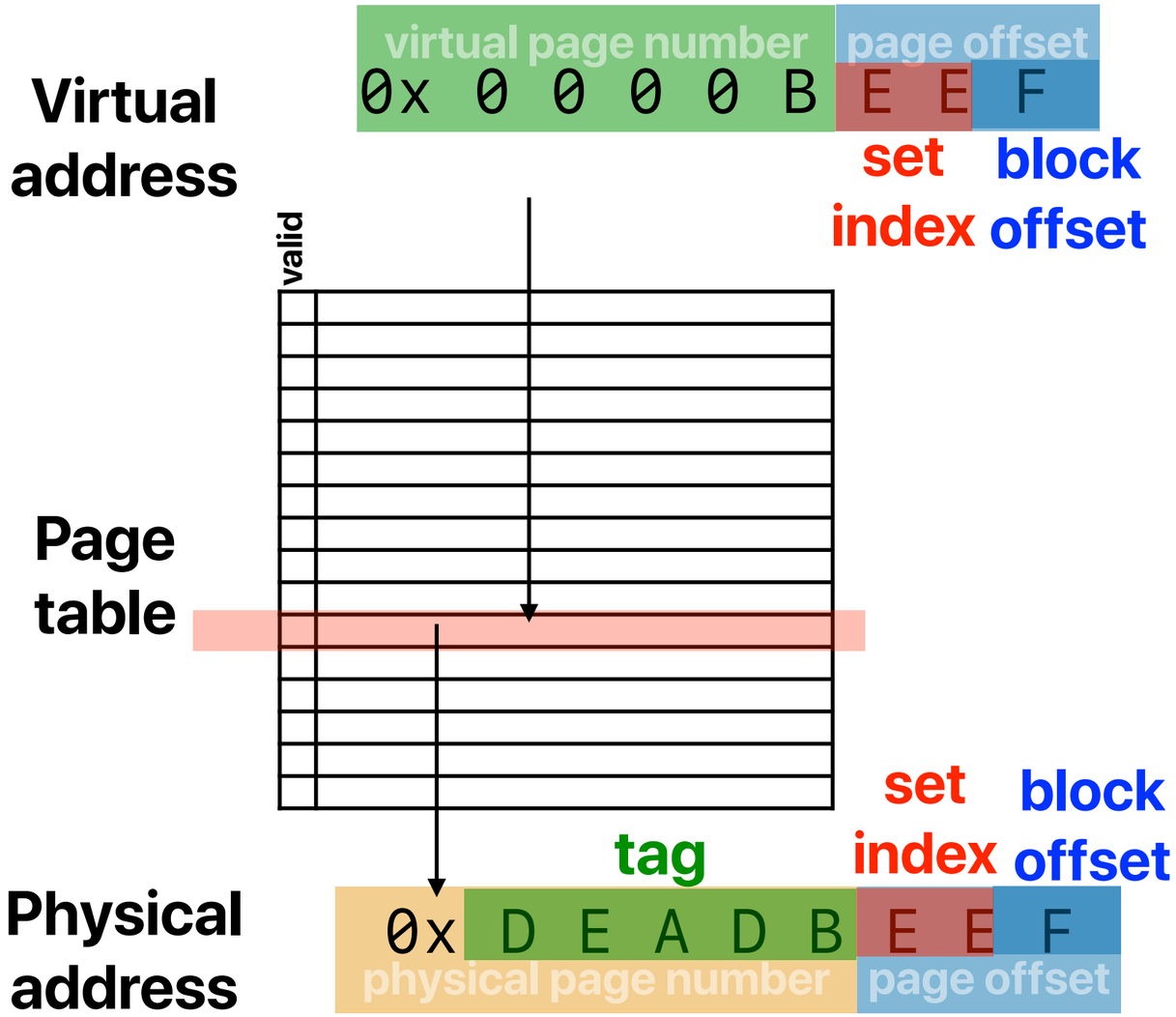
$$lg(B) + lg(S) = lg(4096) = 12$$

$$C = ABS$$

$$C = A \times 2^{12}$$

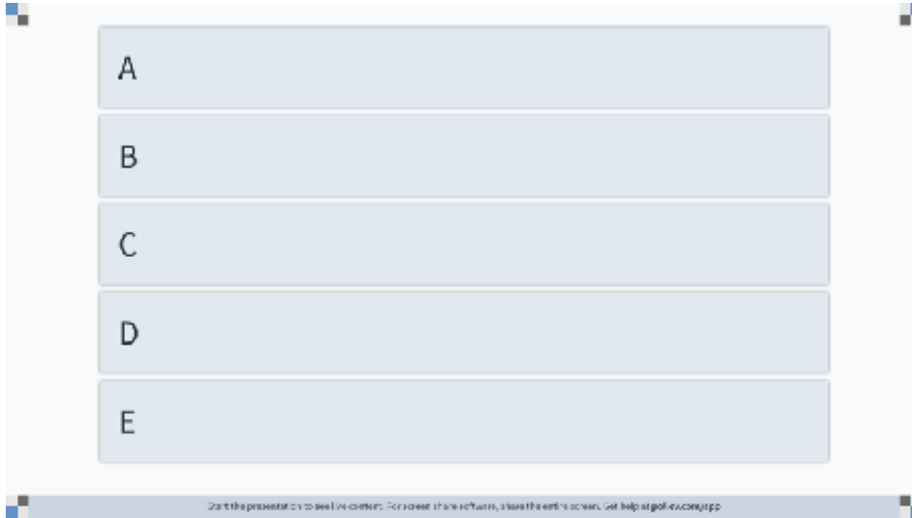
$$if\ A = 1$$

$$C = 4KB$$



Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the operating system use 4K pages.
  A. 32B blocks, 2-way
  B. 32B blocks, 4-way
  C. 64B blocks, 4-way
  D. 64B blocks, 8-way



Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the operating system use 4K pages.
  A. 32B blocks, 2-way
  B. 32B blocks, 4-way
  C. 64B blocks, 4-way
  D. 64B blocks, 8-way

$$lg(B) + lg(S) = lg(4096) = 12$$

$$C = ABS$$

$$32KB = A \times 2^{12}$$

$$A = 8$$

Exactly how Core i7 configures its own cache

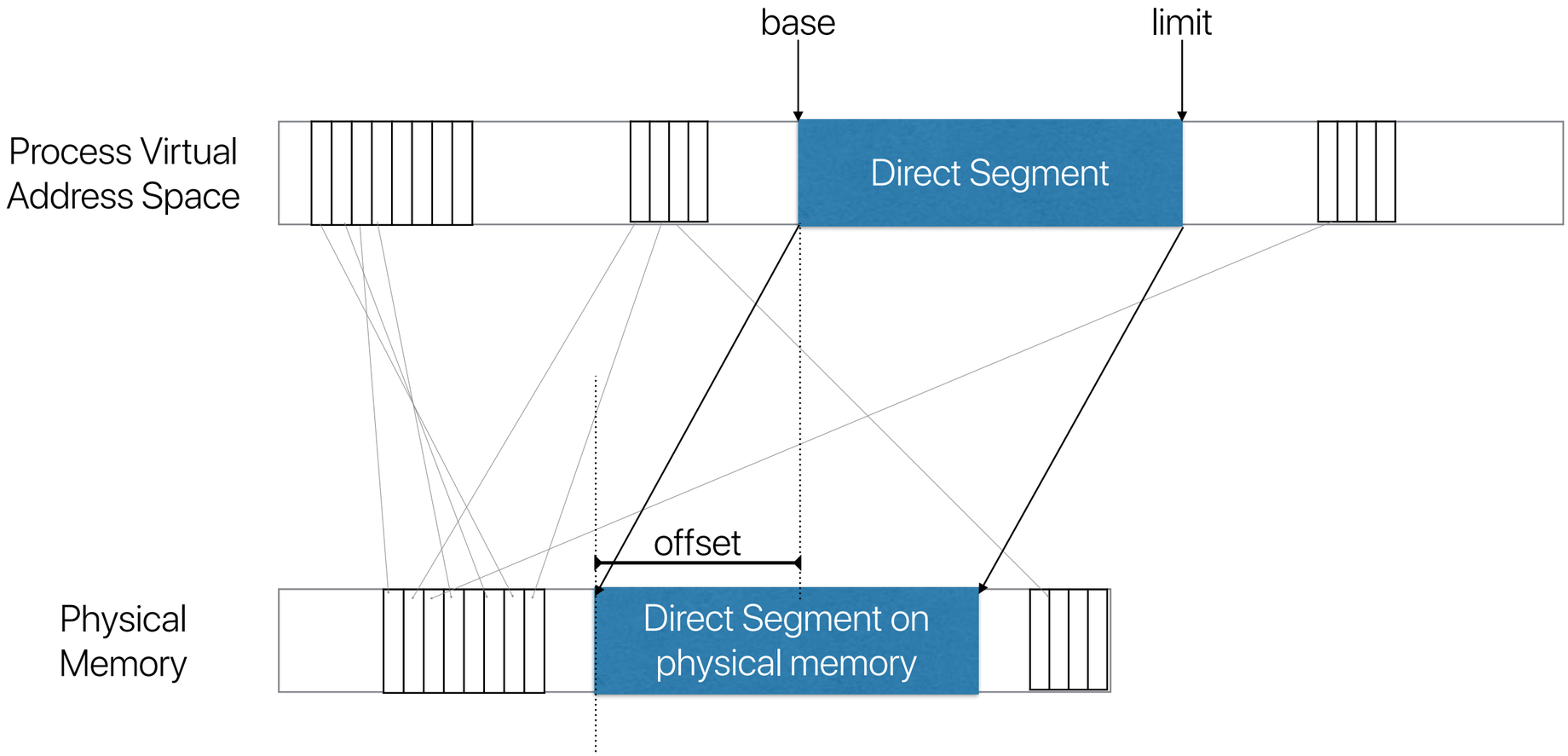
Efficient Virtual Memory for Big Memory Servers

Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill and Michael M. Swift

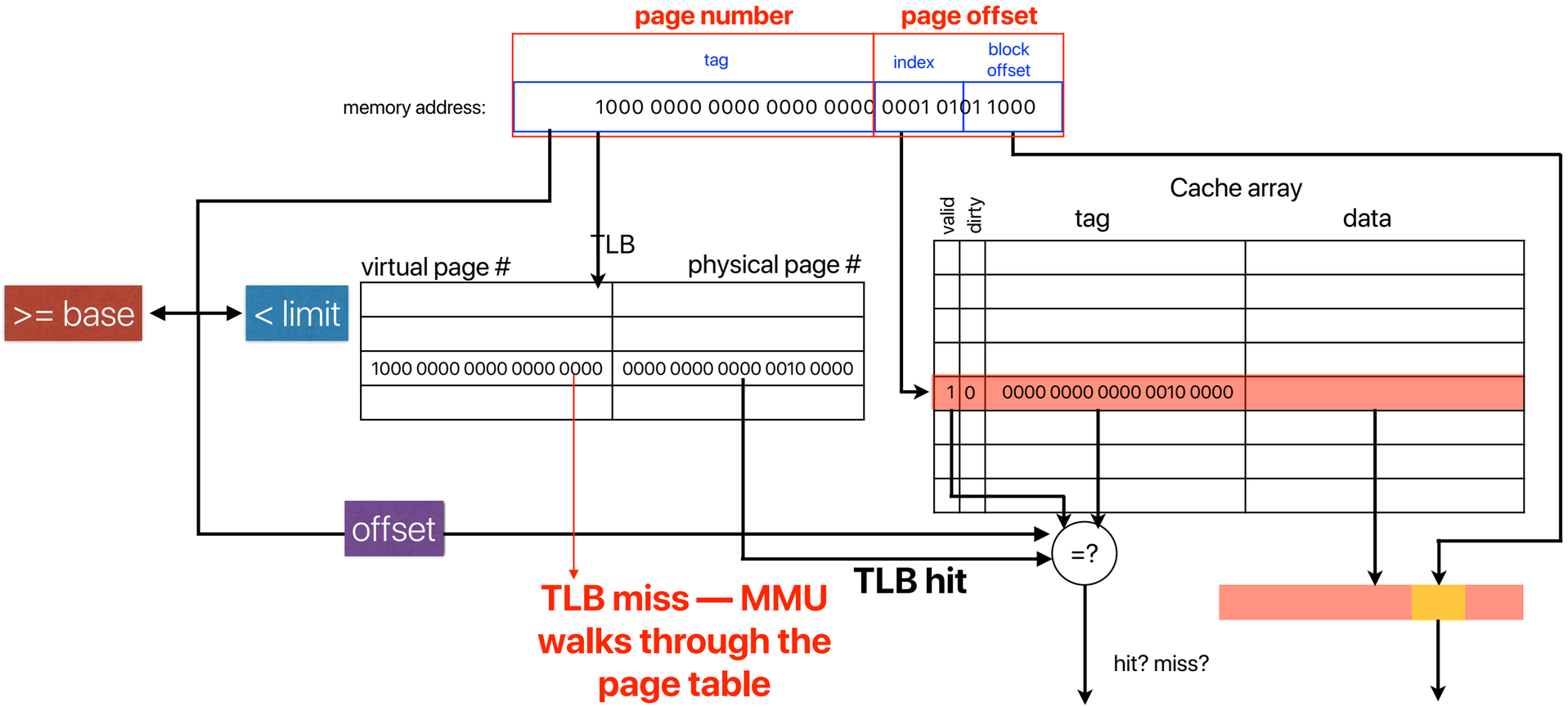
What this paper proposed?

- Mapping part of a process's linear virtual address to a "direct segment" rather than a page
- Direct segment
  - Similar to classic segmentation: adding base, offset, limit registers to each core
  - If the virtual address falls in the range between base and limit, no TLB access is necessary
- Virtual memory outside a direct segment still uses conventional demand paging

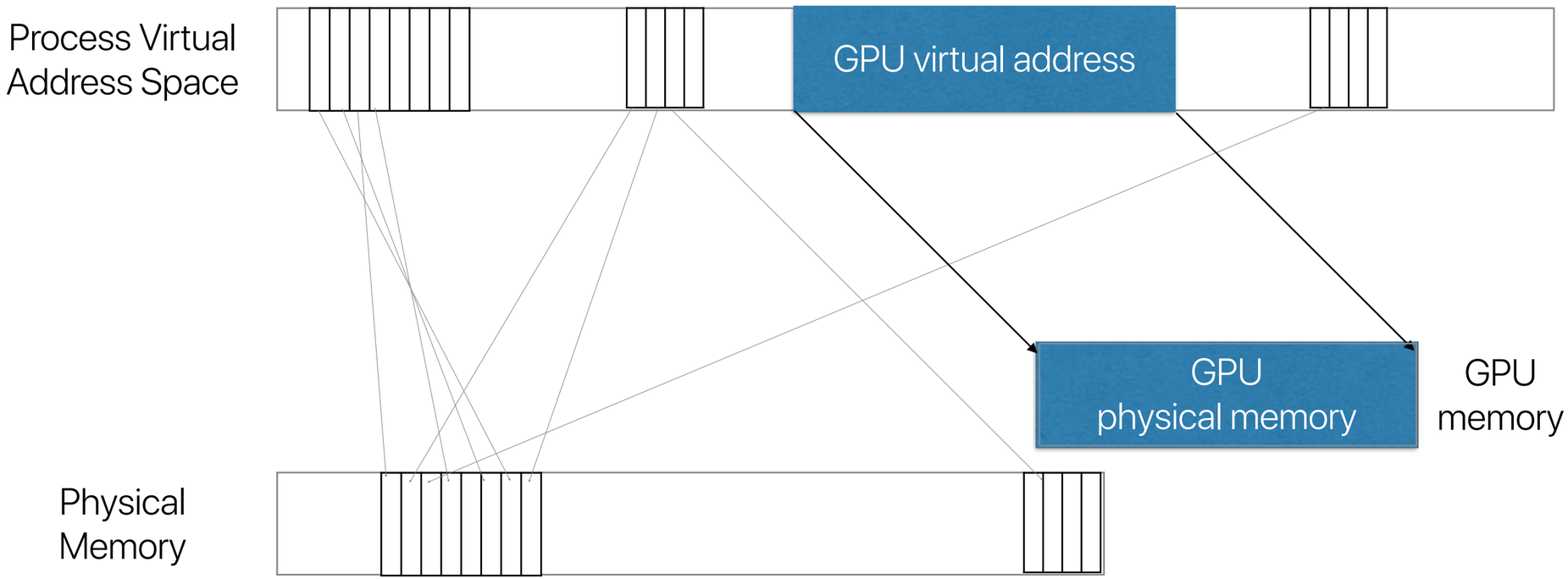
Direct Segment



Architecture overview



Nvidia's unified virtual memory



About midterm

Format of the midterm

- Multiple choices * 15 — like your poll/reading quizzes multiple choices questions
- Short answer questions * 4
- Homework style free-answer questions * 3
  - You need to clearly write down the original form of the applied equation/formula
  - You need to replace each term accordingly with numbers
  - You will have some credits for right equations even though the final number isn't correct
  - You will receive 0 credits if we only see the numbers

Sample Midterm

Identify the performance bottleneck

- Why does an Intel Core i7 @ 3.5 GHz usually perform better than an Intel Core i5 @ 3.5 GHz or AMD FX-8350@4GHz?

Sysbench 2014 from http://www.anandtech.com/

A. Because the instruction count of the program are different
B. Because the clock rate of AMD FX is higher
C. Because the CPI of Core i7 is better
D. Because the clock rate of AMD FX is higher and CPI of Core i7 is better
E. None of the above

Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?
  - ① If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded
  - ② With unlimited amount of parallel hardware units, single-core performance does not matter anymore
  - ③ With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
  - ④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

How programmer affects performance?

- Performance equation consists of the following three factors
  ① IC
  ② CPI
  ③ CT

  How many can a **programmer** affect?

  A. 0
  B. 1
  C. 2
  D. 3

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

- How many of the following make(s) the performance of A better than B?
  - ① IC
  - ② CPI
  - ③ CT
  - A. 0
  - B. 1
  - C. 2
  - D. 3

Fair comparison

- How many of the following comparisons are fair?
  - ① Comparing the frame rates of Halo 5 on AMD RyZen 1600X and civilization on Intel Core i7 7700X
  - ② Using bit torrent to compare the network throughput on two machines
  - ③ Comparing the frame rates of Halo 5 using medium settings on AMD RyZen 1600X and low settings on Intel Core i7 7700X
  - ④ Using the peak floating point performance to judge the gaming performance of machines using AMD RyZen 1600X and Intel Core i7 7700X

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Locality

- Which description about locality of arrays `sum` and `A` in the following code is the most accurate?

```
for(i = 0; i< 100000; i++)
{
    sum[i%10] += A[i];
}
```

A. Access of `A` has temporal locality, `sum` has spatial locality
B. Both `A` and `sum` have temporal locality, and sum also has spatial locality
C. Access of `A` has spatial locality, `sum` has temporal locality
D. Both `A` and `sum` have spatial locality
E. Both `A` and `sum` have spatial locality, and sum also has temporal locality

3Cs and A, B, C

- Regarding 3Cs: compulsory, conflict and capacity misses and A, B, C: associativity, block size, capacity
  How many of the following are correct?
  - ① Increasing associativity can reduce conflict misses
  - ② Increasing associativity can reduce hit time
  - ③ Increasing block size can increase the miss penalty
  - ④ Increasing block size can reduce compulsory misses

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

intel Core i7

- L1 data (D-L1) cache configuration of Core i7
  - Size 32KB, 8-way set associativity, 64B block
  - Assume 64-bit memory address
  - Which of the following is NOT correct?

    A. Tag is 52 bits

    B. Index is 6 bits

    C. Offset is 6 bits

    D. The cache has 128 sets

Virtual indexed, physical tagged cache limits the cache size

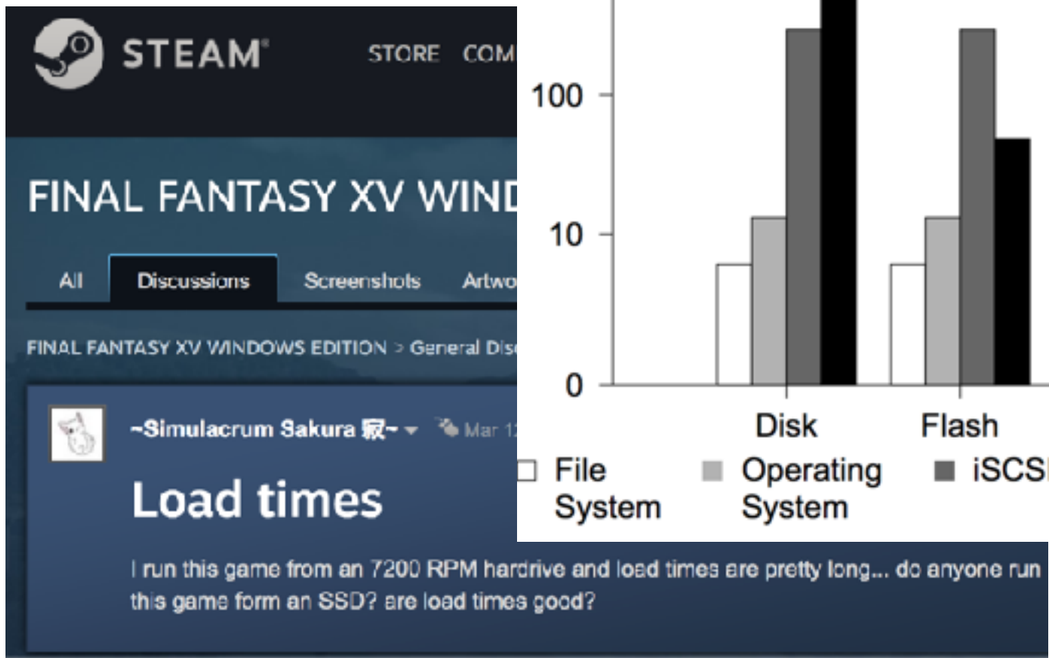
- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the system use 4K pages.
  A. 32B blocks, 2-way
  B. 32B blocks, 4-way
  C. 64B blocks, 4-way
  D. 64B blocks, 8-way

When we have virtual memory…

- In a modern x86-64 processor supports virtual memory through, how many memory accesses can an instruction incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. More than 10

Practicing Amdahl's Law (2)

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?
  A. ~7x
  B. ~10x
  C. ~17x
  D. ~29x
  E. ~100x



AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

How many of the cache misses are **conflict** misses?

A. 6.25%
B. 66.67%
C. 68.75%
D. 93.75%
E. 100%

The result of `sizeof(struct student)`

- Consider the following data structure:

```
struct student {
    int id;
    double *homework;
    int participation;
    double midterm;
    double average;
};
```

What's the output of `printf("%lu\n",sizeof(struct student))`?

A. 20
B. 28
C. 32
D. 36
E. 40

What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

Block + Transpose

```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}


for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

What data structure is performing better

	Array of objects	object of arrays

	`struct grades`
	`{`
	`  int id;`
	`  double *homework;`
	`  double average;`
	`};`
		`struct grades`
		`{`
		`  int *id;`
		`  double **homework;`
		`  double *average;`
		`};`

average of each homework	`for(i=0;i<homework_items; i++)`
	`{`
	`gradesheet[total_number_students].homework[i] = 0.0;`
	`  for(j=0;j<total_number_students;j++)`
	`gradesheet[total_number_students].homework[i]`
	`+=gradesheet[j].homework[i];`
	`  gradesheet[total_number_students].homework[i] /=`
	`(double)total_number_students;`
	`}`
		`for(i = 0;i < homework_items; i++)`
		`{`
		`  gradesheet.homework[i][total_number_students] = 0.0;`
		`  for(j = 0; j <total_number_students;j++)`
		`  {`
		`  gradesheet.homework[i][total_number_students] +=`
		`gradesheet.homework[i][j];`
		`  }`
		`  gradesheet.homework[i][total_number_students] /=`
		`total_number_students;`
		`}`

- Considering your workload would like to calculate the average score of **one of the homework** for **all students**, which data structure would deliver better performance?
  A. Array of objects
  B. Object of arrays

Which of the following schemes can help Phenom II?

- How many of the following schemes mentioned in “improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers” would help AMD Phenom II for the code in vector addition code?
  - ① Missing cache
  - ② Victim cache
  - ③ Prefetch
  - ④ Stream buffer
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Sample short answer questions (< 30 words)

- What are the limitations of compiler optimizations? Can you list two?
- Please define Amdahl's Law and explain each term in it
- Please define the CPU performance equation and explain each term.
- Can you list two things affecting each term in the performance equation?
- What's the difference between latency and throughput? When should you use latency or throughput to judge performance?
- What's "benchmark" suite? Why is it important?
- Why TFLOPS or inferences per second is not a good metrics?

Amdahl's Law for multiple optimizations

- Assume that memory access takes 30% of execution time.
  - Cache can speedup 80% of memory operation by a factor of 4
  - L2 cache can speedup 50% of the remaining 20% by a factor of 2
- What's the total speedup?

Speedup of Y over X

- Consider the same program on the following two machines, X and Y. By how much Y is faster than X?

	Clock Rate	Instructions	Percentage of Type-A	CPI of Type-A	Percentage of Type-B	CPI of Type-B	Percentage of Type-C	CPI of Type-C
Machine X	3 GHz	5000000000	20%	8	20%	4	60%	1
Machine Y	5 GHz	5000000000	20%	13	20%	4	60%	1

Performance evaluation with cache

- Consider the following cache configuration on RISC-V processor:

	I-L1	D-L1	L2	DRAM
size	32K	32K	256K	Big enough
block size	64 Bytes	64 Bytes	64 Bytes	4KB pages
associativity	2-way	2-way	8-way	
access time	1 cycle (no penalty if it's a hit)	1 cycle (no penalty if it's a hit)	10 cycles	100 cycles
local miss rate	2%	10%, 20% dirty	15% (i.e., 15% of L1 misses, also miss in the L2), 30% dirty	
Write policy	N/A	Write-back, write allocate		
Replacement	LRU replacement policy			

The application has 20% branches, 10% loads/stores, 70% integer instructions. Assume that TLB miss rate is 2% and it requires 100 cycles to handle a TLB miss. Also assume that the branch predictor has a hit rate of 87.5%, what's the CPI of branch, L/S, and integer instructions? What is the average CPI?

Cache simulation

- The processor has a 8KB, 256B blocked, 2-way L1 cache. Consider the following code:

```
for(i=0;i<256;i++) {
    a[i] = b[i] + c[i];
// load a[i] and load b[i], store to c[i]
// &a[0] = 0x10000, &b[0] = 0x20000, &c[0] = 0x30000
}
```

- What's the total miss rate? How many of the misses are compulsory misses? How many of the misses are conflict misses?
- How can you improve the cache performance of the above code through changing hardware?
- How can you improve the performance **without** changing hardware?

Announcement

- Assignment #2 due this Wednesday
- Midterm next Monday
  - Will release a sample midterm this Wednesday
  - You may review/focus on the materials/topics covered in lectures
  - You should review your assignments
  - Cover topics including this Wednesday
- Project will be up by the end of the week