Memory Management:
Translation Lookaside Buffer

CS 571: Operating Systems (Spring 2020)
Lecture 7b

Yue Cheng

Some material taken/derived from:
- Wisconsin CS-537 materials created by Remzi Arpaci-Dusseau.

Licensed for use under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

Paging Problems

- Page tables are too slow

- Page tables are too big

Address Translation Steps

- Hardware: for each memory reference
    1. Extract VPN (virt page num) from VA (virt addr)
    2. Calculate addr of PTE (page table entry)
    3. Fetch PTE
    4. Extract PFN (phys page frame num)
    5. Build PA (phys addr)
    6. Fetch PA to register

Address Translation Steps

- Hardware: for each memory reference
  1. Extract VPN (virt page num) from VA (virt addr)
  2. Calculate addr of PTE (page table entry)
  3. Fetch PTE
  4. Extract PFN (phys page frame num)
  5. Build PA (phys addr)
  6. Fetch PA to register

- Q: Which steps are expensive??

Address Translation Steps

- Hardware: for each memory reference

cheap 1. Extract VPN (virt page num) from VA (virt addr)
cheap 2. Calculate addr of PTE (page table entry)
expensive 3. Fetch PTE
cheap 4. Extract PFN (phys page frame num)
cheap 5. Build PA (phys addr)
expensive 6. Fetch PA to register

- Q: Which steps are expensive??

Address Translation Steps

- Hardware: for each memory reference

cheap 1. Extract VPN (virt page num) from VA (virt addr)
cheap 2. Calculate addr of PTE (page table entry)
expensive 3. Fetch PTE
cheap 4. Extract PFN (phys page frame num)
cheap 5. Build PA (phys addr)
expensive 6. Fetch PA to register

- Q: Which expensive steps can we avoid??

Array Iterator

- A simple code snippet in array.c

```
int sum = 0;
for (i=0; i<N; i++) {
    sum += a[i];
}
```

- Compile it using gcc

```
prompt> gcc -o array array.c -Wall -O
prompt> ./array
```

- Dump the assembly code
  - `objdump` (Linux) or `otool` (Mac)

Trace the Memory Accesses

Virt

load 0x3000

load 0x3004

load 0x3008

load 0x300C

…

Trace the Memory Accesses

Virt	**Phys**
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	load 0x100C
	load 0x7004
load 0x3008	load 0x100C
	load 0x7008
load 0x300C	load 0x100C
…	load 0x700C

Trace the Memory Accesses

Virt	**Phys**
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	load 0x100C
	load 0x7004
load 0x3008	load 0x100C
	load 0x7008
load 0x300C	load 0x100C
…	load 0x700C

1st mem access: Fetch PTE

Trace the Memory Accesses

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	load 0x100C
	load 0x7004
load 0x3008	load 0x100C
	load 0x7008
load 0x300C	load 0x100C
…	load 0x700C

Map VPN to PFN: 3 → 7

Trace the Memory Accesses

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	load 0x100C
	load 0x7004
load 0x3008	load 0x100C
	load 0x7008
load 0x300C	load 0x100C
…	load 0x700C

2nd mem access: access **a[i]**

Trace the Memory Accesses

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	load 0x100C
	load 0x7004
load 0x3008	load 0x100C
	load 0x7008
load 0x300C	load 0x100C
…	load 0x700C

Note: 1. Each virt mem access → **two** phys mem accesses
2. **Repeated** memory accesses!

Translation Lookaside Buffer (TLB)

Performance Problems of Paging

- A basic memory access protocol
  1. Fetch the translation from in-memory page table
  2. Explicit load/store access on a memory address

- In this scheme every data/instruction access requires two memory accesses
  - One for the page table
  - and one for the data/instruction

- Too much performance overhead!

Speeding up Translation

- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called translation lookaside buffer (TLB)
- A TLB is part of the memory-management unit (MMU)
- A TLB is a **hardware** cache
- Algorithm sketch
  - For each virtual memory reference, hardware first checks the TLB to see if the desired translation is held therein

TLB Basic Algorithm

1. Extract VPN from VA

TLB Basic Algorithm

1. Extract VPN from VA
2. Check if TLB holds the translation

TLB Basic Algorithm

1. Extract VPN from VA
2. Check if TLB holds the translation
3. If it is a TLB hit – extract PFN from the TLB entry, concatenate it onto the offset to form the PA

TLB Basic Algorithm

1. Extract VPN from VA
2. Check if TLB holds the translation
3. If it is a TLB hit – extract PFN from the TLB entry, concatenate it onto the offset to form the PA

Fast path

TLB Basic Algorithm

1. Extract VPN from VA
2. Check if TLB holds the translation
3. If it is a TLB hit – extract PFN from the TLB entry, concatenate it onto the offset to form the PA

   Fast path

4. If it is a TLB miss – access *page table* to get the translation, update the TLB entry with the translation

TLB Basic Algorithm

1. Extract VPN from VA
2. Check if TLB holds the translation
3. If it is a TLB hit – extract PFN from the TLB entry, concatenate it onto the offset to form the PA

Fast path

4. If it is a TLB miss – access *page table* to get the translation, update the TLB entry with the translation

Slow path

Array Iterator (w/ TLB)

```
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```

Trace the Memory Accesses (w/ TLB)

Virt

load 0x3000

load 0x3004

load 0x3008

load 0x300C

…

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
0		
0		
0		
0		

Virt	Phys
load 0x3000	
load 0x3004	
load 0x3008	
load 0x300C	
…	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Miss

Valid	Virt	Phys
0		
0		
0		
0		

Virt **Phys**

load 0x3000

load 0x3004

load 0x3008

load 0x300C

…

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
0		
0		
0		
0		

Miss

Virt	Phys
load 0x3000	load 0x100C
load 0x3004	
load 0x3008	
load 0x300C	
…	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
0		
0		
0		

Virt	**Phys**
load 0x3000	load 0x100C
load 0x3004	
load 0x3008	
load 0x300C	
…	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
0		
0		
0		

Virt	**Phys**
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	
load 0x3008	
load 0x300C	
…	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
0		
0		
0		

Virt	**Phys**
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	
load 0x3008	
load 0x300C	
…	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

	Valid	Virt	Phys
Hit →	1	3	7
	0		
	0		
	0		

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	(TLB hit)
load 0x3008	
load 0x300C	
…	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

Virt	**Phys**
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	(TLB hit)
	load 0x7004
load 0x3008	
load 0x300C	
…	

CPU's TLB cache

Valid	Virt	Phys
1	3	7
0		
0		
0		

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
0		
0		
0		

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	(TLB hit)
	load 0x7004
load 0x3008	(TLB hit)
	load 0x7008
load 0x300C	(TLB hit)
…	load 0x700C

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
0		
0		
0		

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	(TLB hit)
	load 0x7004
load 0x3008	(TLB hit)
	load 0x7008
load 0x300C	(TLB hit)
	load 0x700C
…	
load 0x2000	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
1	2	4
0		
0		

Miss

Virt	Phys
load 0x3000	load 0x100C load 0x7000
load 0x3004	(TLB hit) load 0x7004
load 0x3008	(TLB hit) load 0x7008
load 0x300C	(TLB hit) load 0x700C
…	
load 0x2000	load 0x100F

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
1	2	4
0		
0		

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	(TLB hit)
	load 0x7004
load 0x3008	(TLB hit)
	load 0x7008
load 0x300C	(TLB hit)
…	load 0x700C
load 0x2000	load 0x100F
	load 0x4000

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
1	2	4
0		
0		

Virt	**Phys**
load 0x3000	load 0x100C load 0x7000
load 0x3004	(TLB hit) load 0x7004
load 0x3008	(TLB hit) load 0x7008
load 0x300C	(TLB hit) load 0x700C
…	
load 0x2000	load 0x100F load 0x4000
load 0x2004	

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

	Valid	Virt	Phys
	1	3	7
Hit	1	2	4
	0		
	0		

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	(TLB hit)
	load 0x7004
load 0x3008	(TLB hit)
	load 0x7008
load 0x300C	(TLB hit)
	load 0x700C
…	
load 0x2000	load 0x100F
	load 0x4000
load 0x2004	(TLB hit)

Trace the Memory Accesses (w/ TLB)

P1's page table

0	5
1	3
2	4
3	1

CPU's TLB cache

Valid	Virt	Phys
1	3	7
1	2	4
0		
0		

Virt	Phys
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	(TLB hit)
	load 0x7004
load 0x3008	(TLB hit)
	load 0x7008
load 0x300C	(TLB hit)
…	load 0x700C
load 0x2000	load 0x100F
	load 0x4000
load 0x2004	(TLB hit)
	load 0x4004

How Many TLB Lookups

Assume 4KB pages

```
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```

How Many TLB Lookups

Assume 4KB pages

```
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```

Array `a[]` has 1024 items, each item is 4 bytes:

`Size(a) = 4096`

How Many TLB Lookups

Assume 4KB pages

```
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```

Array `a[]` has 1024 items, each item is 4 bytes:

`Size(a) = 4096`

Num of TLB miss: `4096/4096 = 1 or 2`

How Many TLB Lookups

Assume 4KB pages

```
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```

Array `a[]` has 1024 items, each item is 4 bytes:

`Size(a) = 4096`

Best case: Num of TLB miss: `4096/4096 = 1`

TLB miss rate: `1/1024 = 0.09%`

How Many TLB Lookups

Assume 4KB pages

```
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```

Array `a[]` has 1024 items, each item is 4 bytes:

`Size(a) = 4096`

Best case: Num of TLB miss: `4096/4096 = 1`

TLB miss rate: `1/1024 = 0.09%`

TLB hit rate : `99.91%` (almost 100%)

TLB Content

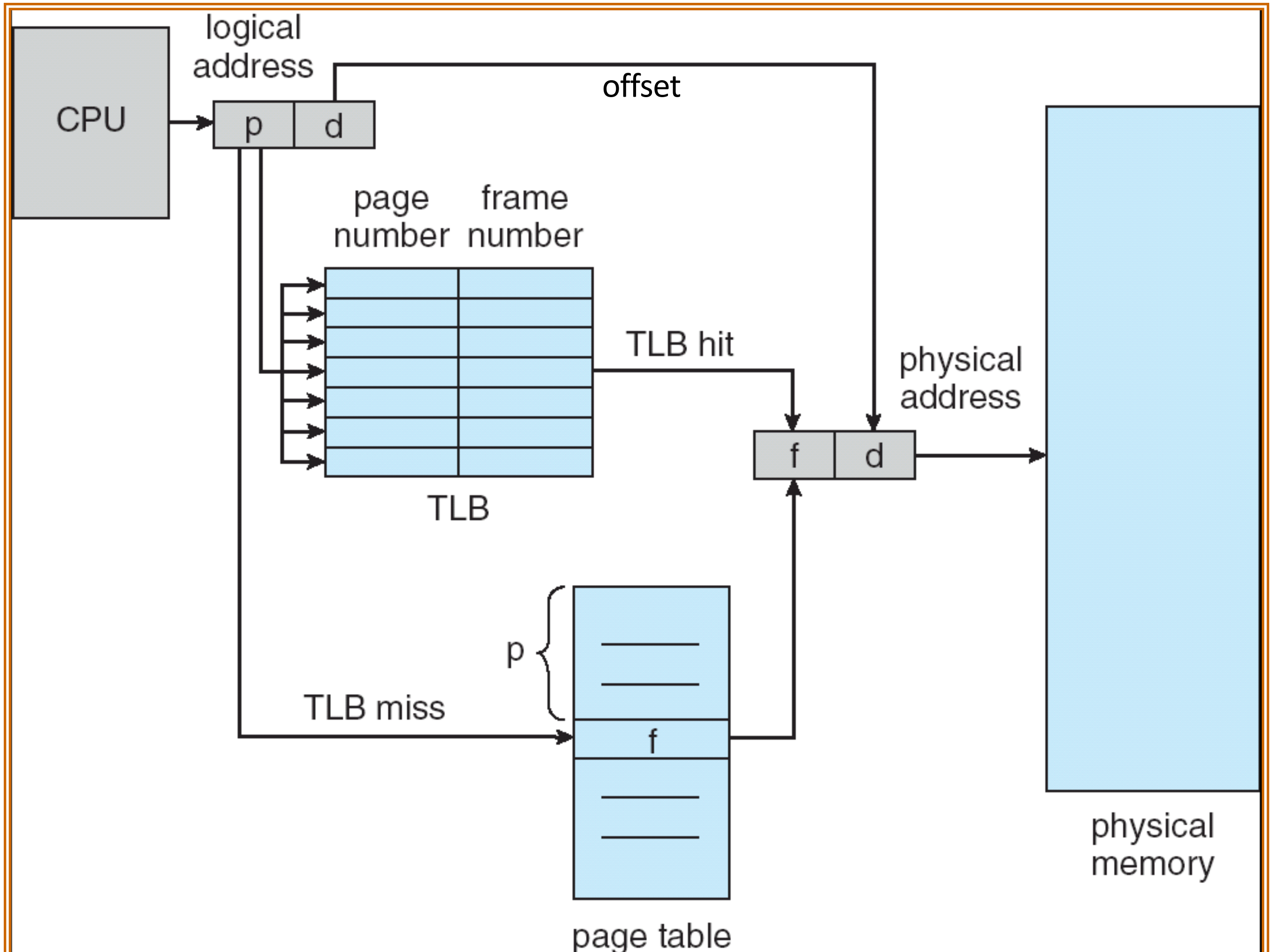
- Some entries are [wired down or reserved] for permanently valid translations

- TLB is a fully associative cache
  - Any given translation can be anywhere in the TLB
  - Hardware searches entire TLB in parallel to find a match

- A typical TLB entry

```
VPN | PFN | other bits
```

Paging Hardware w/ TLB



TLB Issue: Context Switch

- TLB contains translations only valid for the currently running process

- Switching from one process to another requires OS or hardware to do more work

One Example

- How does OS distinguish which entry is for which process?

	VPN	PFN	valid	prot
P1	10	100	1	rwx
	—	—	0	—
P2	10	170	1	rwx
	—	—	0	—

One Simple Solution: Flush

- OS flushes the whole TLB on context switch

- Flush operation sets all valid bit to 0

One Simple Solution: Flush

- OS flushes the whole TLB on context switch
- Flush operation sets all valid bit to 0
- Problem: the overhead is too high if OS switches processes too frequently

Optimization: ASID

- Some hardware systems provide an address space identifier (ASID) field in the TLB

- Think of ASID as a process identifier (PID)
  - An 8-bit field

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
—	—	0	—	—
10	170	1	rwx	2
—	—	0	—	—

Page Sharing

- Leveraging ASID for supporting page sharing

- In this example, two entries from two processes with two different VPNs point to the same physical page

VPN	PFN	valid	prot	ASID
10	101	1	r-x	1
—	—	0	—	—
50	101	1	r-x	2
—	—	0	—	—

Page Sharing (cont.)

- Shared code
  - One copy of read-only (reentrant) code shared among processes (e.g., text editors, compilers, window systems)
  - Particularly important for time-sharing environments
- Private code and data
  - Each process keeps a separate copy of the code and data

TLB Replacement Policy

- Cache: When we want to add a new entry to a full TLB, an old entry must be evicted and replaced

- Least-recently-used (LRU) policy
  - Intuition: A page entry that has not recently been used implies it won't likely to be used in the near future

- Random policy
  - Evicts an entry at random

TLB Workloads

- Sequential array accesses can almost always hit in the TLB, and hence are very fast
- What pattern would be slow?

TLB Workloads

- Sequential array accesses can almost always hit in the TLB, and hence are very fast
- What pattern would be slow?
  - Highly random, with no repeat accesses

Workload Characteristics

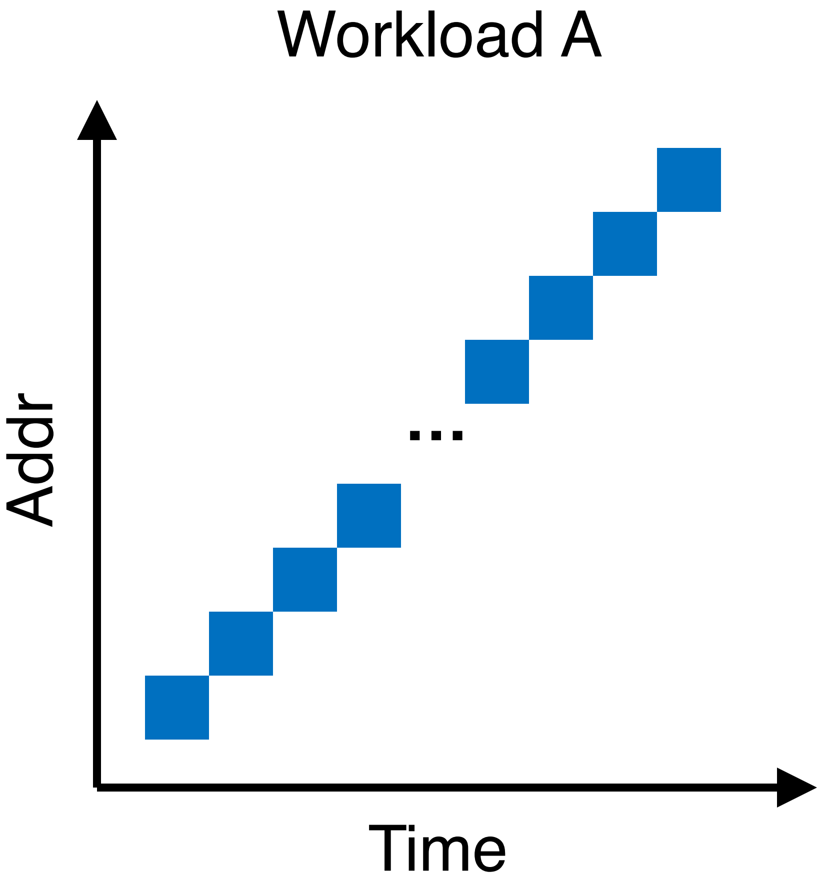
Workload A

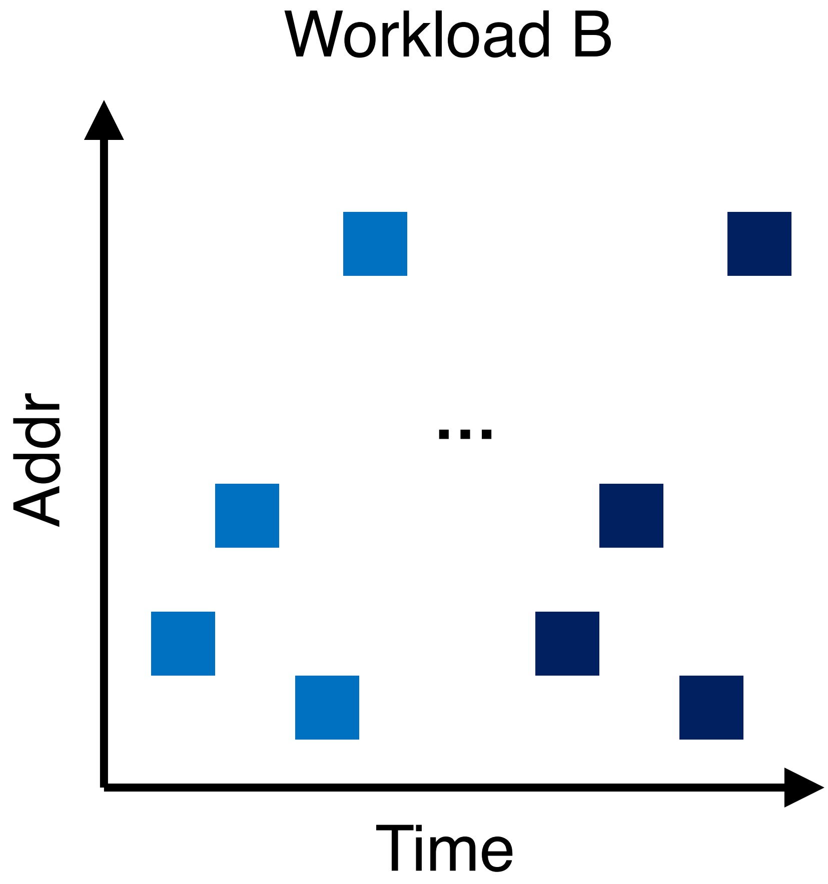
```
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```

Workload B

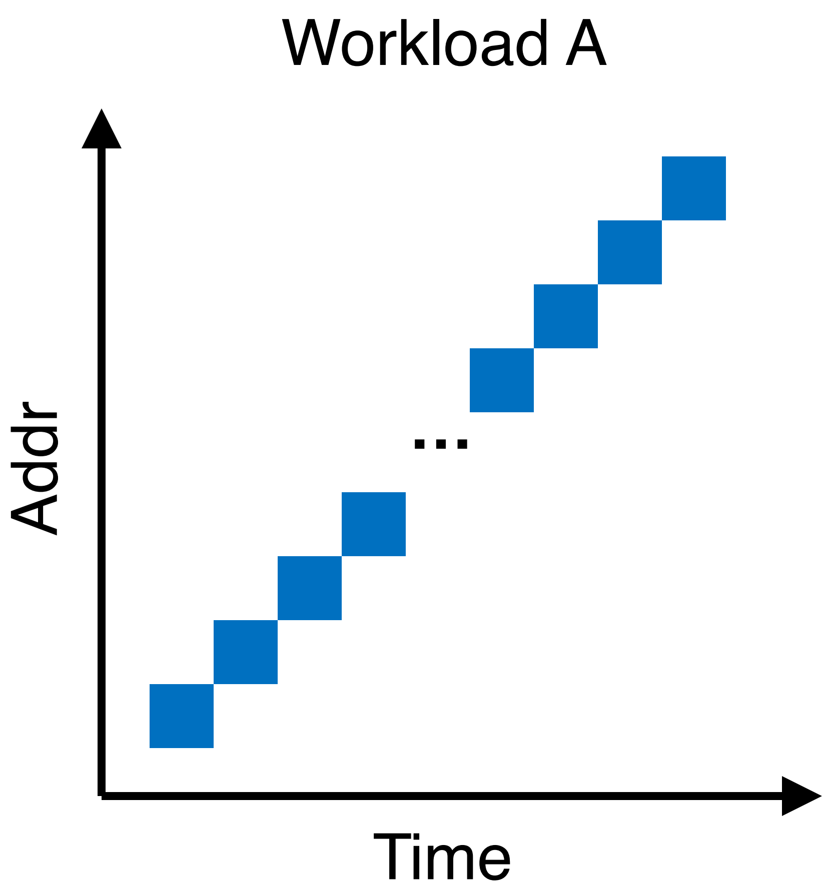
```
int sum = 0;
srand(1234);
for (i=0; i<512; i++) {
    sum += a[rand() % N];
}
srand(1234); // same seed
for (i=0; i<512; i++) {
    sum += a[rand() % N];
}
```

Access Patterns

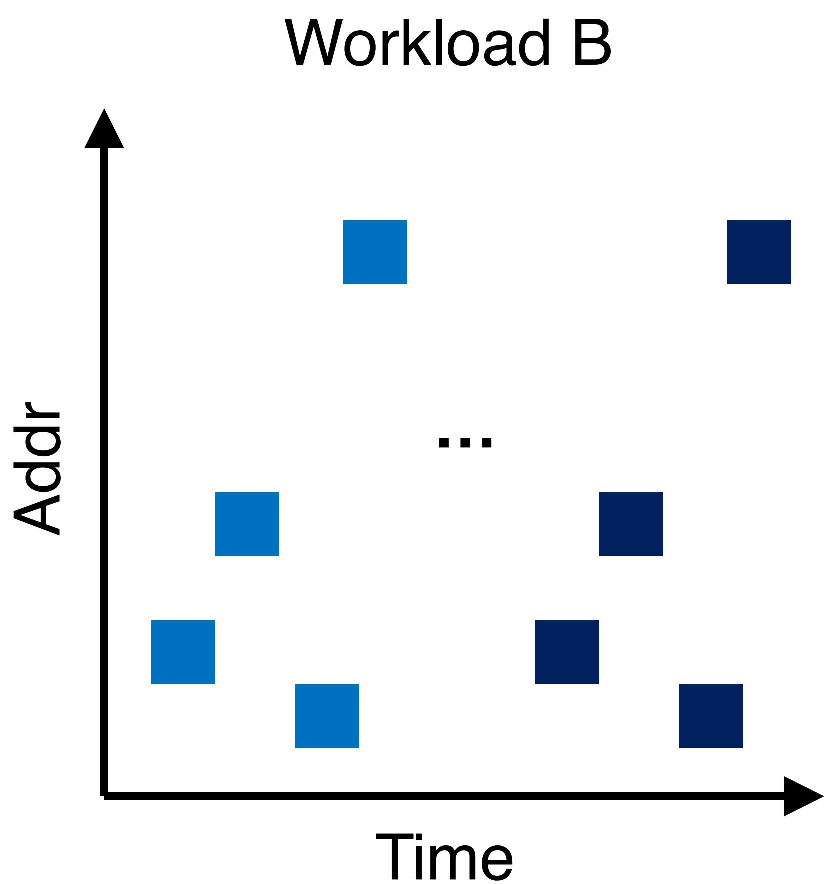




Access Patterns



Spatial Locality



Temporal Locality

Workload Locality

- Spatial locality:
  - Future access will be to nearby addresses

- Temporal locality:
  - Future access will be repeated to the same data

Workload Locality

- Spatial locality:
  - Future access will be to nearby addresses

- Temporal locality:
  - Future access will be repeated to the same data

- Q: What TLB characteristics are best for each type?

Workload Locality

- Spatial locality:
  - Future access will be to nearby addresses
- Temporal locality:
  - Future access will be repeated to the same data
- Q: What TLB characteristics are best for each type?
  - One TLB entry holds the translation for one memory page: all accesses to that particular page benefit from this single TLB entry (**spatial** locality)
  - TLB is a small cache (if supporting LRU): memory accesses with **temporal** locality benefit

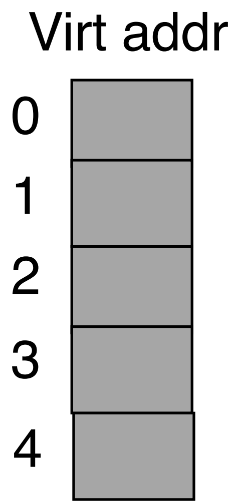
TLB Replacement Policy

- Cache: When we want to add a new entry to a full TLB, an old entry must be evicted and replaced

- Least-recently-used (LRU) policy
  - Intuition: A page entry that has not recently been used implies it won't likely to be used in the near future

- Random policy
  - Evicts an entry at random

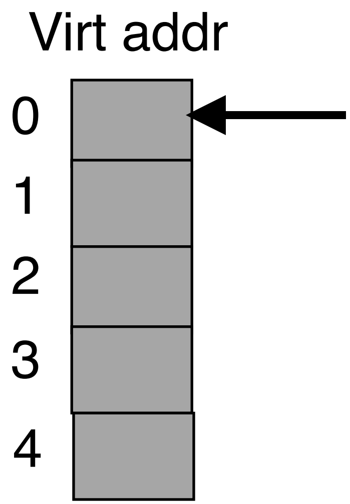
LRU Trouble



CPU's TLB cache

Valid	Virt	Phys
0		
0		
0		
0		

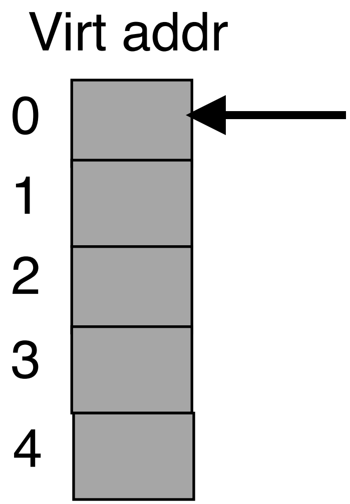
LRU Trouble



CPU's TLB cache

Valid	Virt	Phys
1	0	?
0		
0		
0		

LRU Trouble

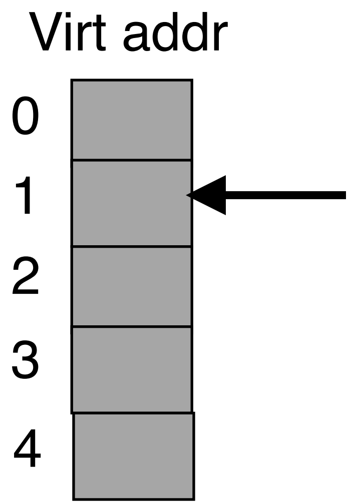


CPU's TLB cache

Valid	Virt	Phys
1	0	?
0		
0		
0		

TLB miss

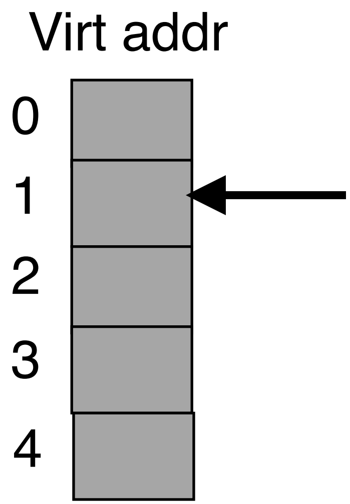
LRU Trouble



CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
0		
0		

LRU Trouble

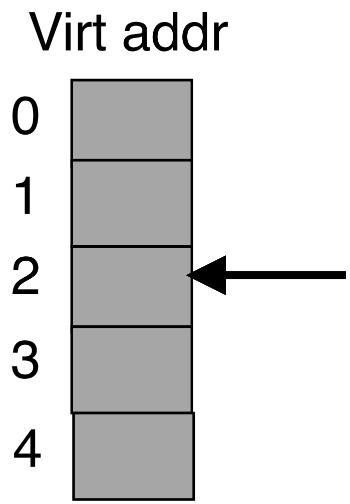


CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
0		
0		

TLB miss

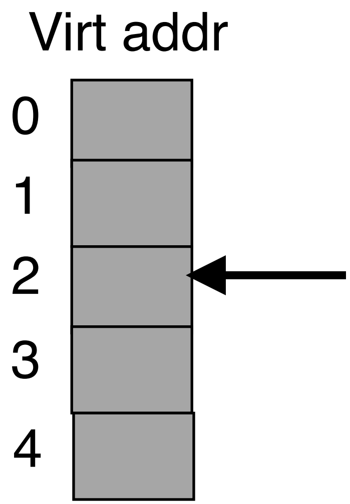
LRU Trouble



CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
1	2	?
0		

LRU Trouble

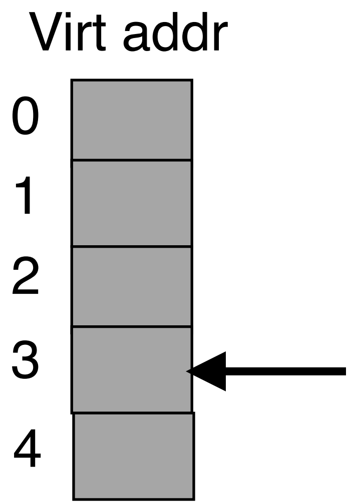


CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
1	2	?
0		

TLB miss

LRU Trouble

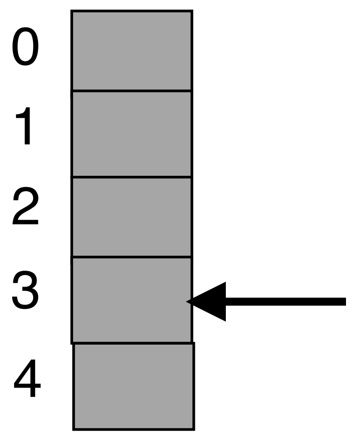


CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
1	2	?
1	3	?

LRU Trouble

Virt addr

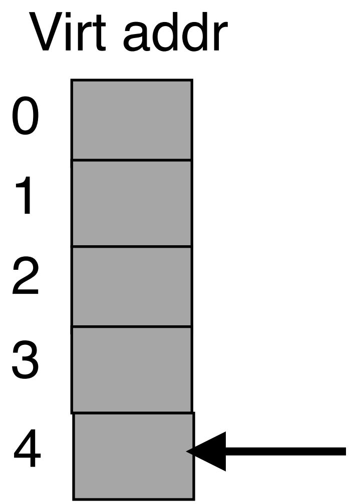


CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
1	2	?
1	3	?

TLB miss

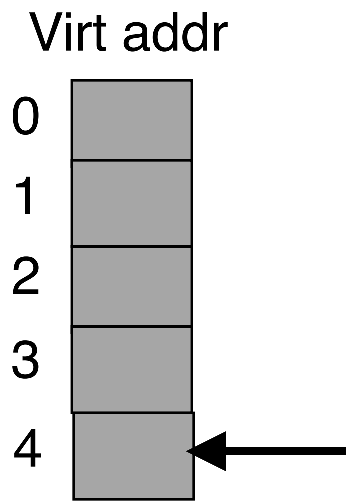
LRU Trouble



CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
1	2	?
1	3	?

LRU Trouble

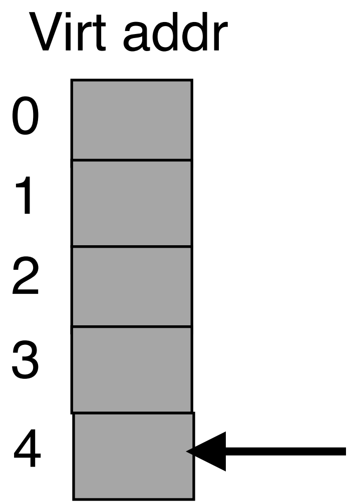


CPU's TLB cache

Valid	Virt	Phys
1	0	?
1	1	?
1	2	?
1	3	?

Now, **0** is the least-recently used item in TLB

LRU Trouble

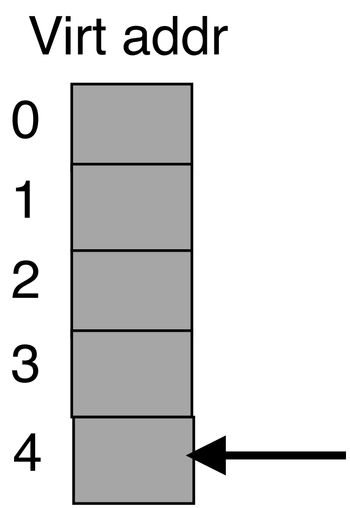


CPU's TLB cache

Valid	Virt	Phys
1	**4**	**?**
1	1	?
1	2	?
1	3	?

Replace 0 with 4

LRU Trouble



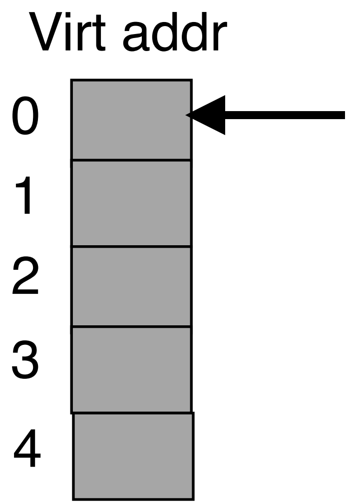
CPU's TLB cache

Valid	Virt	Phys
1	**4**	**?**
1	1	?
1	2	?
1	3	?

TLB miss

Replace 0 with 4

LRU Trouble

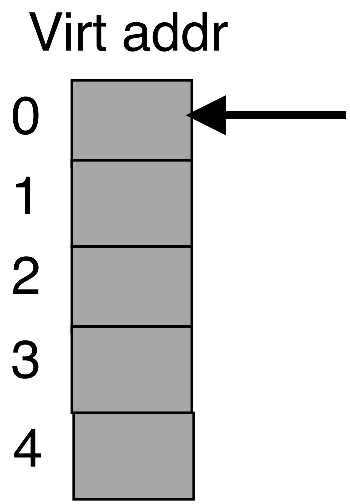


CPU's TLB cache

Valid	Virt	Phys
1	4	?
1	1	?
1	2	?
1	3	?

Accessing 0 again, which was unfortunately just evicted…

LRU Trouble



CPU's TLB cache

Valid	Virt	Phys
1	4	?
1	**0**	**?**
1	2	?
1	3	?

TLB miss

Accessing 0 again, which was unfortunately just evicted… Replace 1 (which is the least-recently used item at this point) with 0…

Takeaway

- LRU
- Random
- When is each better?
  - Sometimes random is better than a “smart” policy!