[537] TLBs

Tyler Harter
9/21/14
Overview

Review Paging

TLBs (Chapter 18)

TLB measurement demo (if time)
Review: Paging
what must you know?
Virtual:  load 0x0000

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>

PTBR

P1 pagetable

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
</tr>
</thead>
</table>

P2 pagetable

<table>
<thead>
<tr>
<th>6</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>
Virtual
load 0x0000

Physical
load 0x0800 (2KB)
Virtual
load 0xQ000

Physical
load 0x0800 (2KB)
load 0x6000 (24KB)
A diagram illustrates a page table with virtual and physical memory mapping. The virtual memory is divided into pages, and the physical memory is divided into segments.

- **Virtual Memory**:
  - Load 0x0000
  - Load 0x0800 (2KB)
  - Load 0x6000 (24KB)

- **Physical Memory**:
  - P1 pagetable
  - P2 pagetable

The diagram shows how pages are mapped to physical memory addresses using page tables (PTs) and page table bases (PTBs).
Virtual
load 0x0000

Physical
load 0x0800 (2KB)
load 0x6000 (24KB)
what must you know?

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x0000</td>
<td>load 0x0800 (2KB)</td>
</tr>
<tr>
<td></td>
<td>load 0x6000 (24KB)</td>
</tr>
<tr>
<td></td>
<td>load 0x1444</td>
</tr>
</tbody>
</table>
assume 8-byte PTEs

Virtual | Physical
load 0x0000 | load 0x0800 (2KB)
load 0x1444 | load 0x6000 (24KB)
Virtual | Physical
---|---
load 0x0000 | load 0x0800 (2KB)
load 0x1444 | load 0x0808
load 0x6000 (24KB)
Virtual | Physical
---|---
load 0x0000 | load 0x0800 (2KB)
load 0x1444 | load 0x0808
load 0x2444 | load 0x2444
PTBR

Virtual | Physical
---|---
load 0x0000 | load 0x0800 (2KB)
load 0x1444 | load 0x0808
load 0x2444 | load 0x2444
Virtual | Physical
--- | ---
load 0x0000 | load 0x0800 (2KB)
load 0x1444 | load 0x0808

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x0000</td>
<td>load 0x0800 (2KB)</td>
</tr>
<tr>
<td>load 0x1444</td>
<td>load 0x6000 (24KB)</td>
</tr>
<tr>
<td>load 0x1444</td>
<td>load 0x0808</td>
</tr>
<tr>
<td>load 0x1444</td>
<td>load 0x2444</td>
</tr>
</tbody>
</table>
Load at virtual addresses:
- load 0x0000
- load 0x0800 (2 KB)
- load 0x1444
- load 0x0808
- load 0x2444
- load 0x0008

Translation from virtual to physical addresses:

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x0000</td>
<td>load 0x0800 (2 KB)</td>
</tr>
<tr>
<td>load 0x1444</td>
<td>load 0x0808</td>
</tr>
<tr>
<td>load 0x1444</td>
<td>load 0x2444</td>
</tr>
<tr>
<td>load 0x1444</td>
<td>load 0x0008</td>
</tr>
</tbody>
</table>
Virtual     | Physical
-------------|-------------
load 0x0000  | load 0x0800 (2KB)
load 0x1444  | load 0x0808
load 0x1444  | load 0x2444
load 0x1444  | load 0x0008
load 0x1444  | load 0x5444
Virtual | Physical
---|---
load 0x0000 | load 0x0800 (2KB)
load 0x1444 | load 0x0808
load 0x1444 | load 0x2444
load 0x1444 | load 0x0008
load 0x5444 | load 0x5444
Chapter 19: TLBs
Outline

What work can we eliminate?

Basic strategy.

Workloads, systems, metrics.

Context switching and security.
Paging Advantages

Flexible Addr Space
- don’t need to find contiguous RAM
- doesn’t waste whole data pages (valid bit)

Easy to manage
- fixed size pages
- simple free list for unused pages
- no need to coalesce
Paging Problems

Too big

Too slow
Paging Problems

Too big

Too slow [today’s focus]
Translation Steps

H/W: for each mem 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** (page frame num)
5. build **PA** (phys addr)
6. fetch **PA** to register
Translation Steps

H/W: for each mem 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** (page frame num)
5. build **PA** (phys addr)
6. fetch **PA** to register

Which steps are expensive?
Translation Steps

H/W: for each mem reference:

1. extract VPN (virt page num) from VA (virt addr) (cheap)
2. calculate addr of PTE (page table entry) (cheap)
3. fetch PTE (expensive)
4. extract PFN (page frame num) (cheap)
5. build PA (phys addr) (cheap)
6. fetch PA to register (expensive)

Which steps are expensive?
Translation Steps

H/W: for each mem reference:

1. extract VPN (virt page num) from VA (virt addr)  (cheap)
2. calculate addr of PTE (page table entry)  (cheap)
3. fetch PTE  (expensive)
4. extract PFN (page frame num)  (cheap)
5. build PA (phys addr)  (cheap)
6. fetch PA to register  (expensive)

Which expensive step can we avoid?
int sum = 0;
for (i=0; i<N; i++) {
    sum += a[i];
}

Array Iterator

Virt
load 0x3000
load 0x3004
load 0x3008
load 0x300C
...


Array Iterator

<table>
<thead>
<tr>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x3000</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x3004</td>
<td>load 0x7000</td>
</tr>
<tr>
<td>load 0x3008</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x300C</td>
<td>load 0x7004</td>
</tr>
<tr>
<td>...</td>
<td>load 0x100C</td>
</tr>
<tr>
<td></td>
<td>load 0x7008</td>
</tr>
<tr>
<td></td>
<td>load 0x100C</td>
</tr>
<tr>
<td></td>
<td>load 0x700C</td>
</tr>
</tbody>
</table>
**Array Iterator**

<table>
<thead>
<tr>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x3000</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x3004</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x3008</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x300C</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>...</td>
<td>load 0x7000C</td>
</tr>
</tbody>
</table>
# Array Iterator

<table>
<thead>
<tr>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x3000</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x3004</td>
<td>load 0x7000</td>
</tr>
<tr>
<td>load 0x3008</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x300C</td>
<td>load 0x7004</td>
</tr>
<tr>
<td>...</td>
<td>load 0x100C</td>
</tr>
<tr>
<td></td>
<td>load 0x7008</td>
</tr>
<tr>
<td></td>
<td>load 0x100C</td>
</tr>
<tr>
<td></td>
<td>load 0x700C</td>
</tr>
</tbody>
</table>
# Array Iterator

<table>
<thead>
<tr>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x3000</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x3004</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x3008</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>load 0x300C</td>
<td>load 0x100C</td>
</tr>
<tr>
<td>...</td>
<td>load 0x700C</td>
</tr>
</tbody>
</table>

Note: The values on the right side are the physical addresses.
Outline

What work can we eliminate?

Basic strategy.

Workloads, systems, metrics.

Context switching and security.
Strategy

Take advantage of repetition.
Use a CPU cache.
Strategy

Take advantage of repetition.
Use a CPU cache.

CPU

RAM

memory interconnect
Strategy

Take advantage of repetition.
Use a CPU cache.

CPU

RAM

memory interconnect
Strategy

Take advantage of repetition.
Use a CPU cache.

[Diagram of CPU and RAM with memory interconnect]
Strategy

Take advantage of repetition.
Use a CPU cache.

CPU

RAM

memory interconnect

popular PTEs often transferred
Strategy

Take advantage of repetition.
Use a CPU cache.
Strategy

Name? ATC: Address Translation Cache? [OSTEP]
Strategy

Name? ATC: **Address** Translation **Cache**?
Nope. TLB: **Translation** Lookaside **Buffer**

![Diagram of CPU and RAM with TLB and PT]
Strategy

Name? ATC: Address Translation Cache? [OSTEP]
Nope. TLB: Translation Lookaside Buffer
Strategy

Name? ATC: **Address Translation Cache?** [OSTEP]
Nope. TLB: **Translation Lookaside Buffer**

Air Traffic Controller
Strategy

Name? ATC: Address Translation Cache? [OSTEP]
Nope. TLB: Translation Lookaside Buffer

CPU

RAM

memory interconnect
Cache Types (more in CS 552)

**Direct-Mapped**: only one place to put entries

**Four-Way Set Associative**: 4 options

**Fully-Associative**: entries can go anywhere
Cache Types (more in CS 552)

Direct-Mapped: only one place to put entries

Four-Way Set Associative: 4 options

Fully-Associative: entries can go anywhere
- most common for TLBs
- must store whole key/value in cache
- search all in parallel
int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}
Array Iterator

Virt
load 0x1000
load 0x1004
load 0x1008
load 0x100C
...

<table>
<thead>
<tr>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>load 0x1000</td>
<td></td>
</tr>
<tr>
<td>load 0x1004</td>
<td></td>
</tr>
<tr>
<td>load 0x1008</td>
<td></td>
</tr>
<tr>
<td>load 0x100C</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
The diagram illustrates a page table and a CPU TLB (Translation Lookaside Buffer) with the following details:

**Page Table:**
- The page table contains entries for each page size, starting from 0 KB to 28 KB.
- Each page size has a corresponding page table entry (PT) and page number (P).
- The page table entries are organized in a hierarchical structure, with parent tables (PT) and child tables (P1, P2).

**CPU TLB:**
- The CPU TLB is a small memory buffer used to cache page translations.
- It contains entries for each load operation, with columns for Valid, Virt, and Phys.
- The entries shown are:
  - `0x1000`: Valid: 0, Virt: 0, Phys: 0
  - `0x1004`: Valid: 0, Virt: 0, Phys: 0
  - `0x1008`: Valid: 0, Virt: 0, Phys: 0
  - `0x100C`: Valid: 0, Virt: 0, Phys: 0

**Page Loads:**
- The diagram shows page loads at specific memory addresses:
  - `0x1000`
  - `0x1004`
  - `0x1008`
  - `0x100C`
Virt
load 0x1000
load 0x1004
load 0x1008
load 0x100C
...

Phys

CPU's TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
CPU's TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virt

load 0x1000

load 0x1004

load 0x1008

load 0x100C

Phys

load 0x0004
CPU's TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PTBR

P1 pagetable

0 KB

<table>
<thead>
<tr>
<th>0 KB</th>
<th>4 KB</th>
<th>8 KB</th>
<th>12 KB</th>
<th>16 KB</th>
<th>20 KB</th>
<th>24 KB</th>
<th>28 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT</td>
<td>PT</td>
<td>P1</td>
<td>P2</td>
<td>P2</td>
<td>P1</td>
<td>P1</td>
<td>P2</td>
</tr>
</tbody>
</table>

Virt

- load 0x1000
- load 0x1004
- load 0x1008
- load 0x100C

Phys

- load 0x0004
- load 0x5000
Virt

Phys

load 0x1000
load 0x0004
load 0x5000
load 0x1004
load 0x1008
load 0x100C

CPU's TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PTBR

0 KB
4 KB
8 KB
12 KB
16 KB
20 KB
24 KB
28 KB

PT

PT

P1 pagetable

0 1 2 3
### CPU's TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### P1 pagetable

| 0 | 1 | 2 | 3 |...
|---|---|---|---|...
| 1 | 5 | 4 |...

### PAGE TABLES

- **P1**
  - Phys: 28 KB
  - Load: 0x1000
  - Load: 0x1004
  - Load: 0x1008
  - Load: 0x100C

### PAGE TABLES

- **P2**
  - Phys: 28 KB

### PAGE TABLES

- **PT**
  - Phys: 28 KB

### PAGE TABLES

- **PT**
  - Phys: 28 KB

### PAGE TABLES

- **PT**
  - Phys: 28 KB
CPU’s TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PTBR

P1 pagetable

0 1 5 4 ...

Virt
load 0x1000
load 0x1004
load 0x1008
load 0x100C

Phys
load 0x0004 (TLB)
load 0x5000 (TLB)
load 0x5004 (TLB)
load 0x5008 (TLB)
load 0x500C
### CPU's TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Virt
- load 0x1000
- load 0x1004
- load 0x1008
- load 0x100C
- load 0x2000

### Phys
- load 0x0004 (TLB)
- load 0x5004 (TLB)
- load 0x5008 (TLB)
- load 0x500C
Virt
load 0x1000
load 0x1004
load 0x1008
load 0x100C
load 0x2000
load 0x0004
load 0x5000
load 0x5004
load 0x5008
load 0x500C
load 0x0008

Phys

CPU’s TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
P1 pagetable

CPU's TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PTBR

Virt

load 0x1000
load 0x1004
load 0x1008
load 0x100C
load 0x2000

Phys

load 0x0004 (TLB)
load 0x5000 (TLB)
load 0x5004 (TLB)
load 0x5008 (TLB)
load 0x500C
load 0x0008
load 0x4000
Virt
load 0x1000
load 0x1004
load 0x1008
load 0x100C
load 0x2000
load 0x2004

Phys
load 0x0004
load 0x5000
load 0x5004
load 0x5008
load 0x500C
load 0x0008
load 0x4000
### P1 Page Table

<table>
<thead>
<tr>
<th>0 KB</th>
<th>4 KB</th>
<th>8 KB</th>
<th>12 KB</th>
<th>16 KB</th>
<th>20 KB</th>
<th>24 KB</th>
<th>28 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT</td>
<td>PT</td>
<td>P1</td>
<td>P2</td>
<td>P2</td>
<td>P1</td>
<td>P1</td>
<td>P2</td>
</tr>
</tbody>
</table>

#### CPU's TLB

<table>
<thead>
<tr>
<th></th>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Virt

- load 0x1000
- load 0x1004
- load 0x1008
- load 0x100C

#### Phys

- load 0x0004
- load 0x05004 (TLB)
- load 0x05008 (TLB)
- load 0x0500C
- load 0x02000
- load 0x02004
- load 0x0008
- load 0x4000 (TLB)
The diagram illustrates a page table (PT) and its corresponding entries in the page table register (PTBR). The page table is divided into pages of 4 KB, 8 KB, 12 KB, 16 KB, 20 KB, and 24 KB. Each page contains a page table (PT) entry, and the PTBR points to the appropriate PT for each level of memory access.

The CPU's TLB (Translation Lookaside Buffer) is shown next to the page table, with entries for valid virtual addresses (Virt) and corresponding physical addresses (Phys). The load addresses are indicated, with some entries marked as (TLB) indicating a translation failure.

Key load addresses:
- `0x1000`
- `0x1004`
- `0x1008`
- `0x100C`
- `0x2000`
- `0x2004`
- `0x5000`
- `0x5004` (TLB)
- `0x5008` (TLB)
- `0x500C` (TLB)
- `0x0004` (TLB)
- `0x4000` (TLB)
- `0x4004`

The page table entries are as follows:

- **Page Table (PT):**
  - 0 KB: PT
  - 4 KB: PT
  - 8 KB: P1
  - 12 KB: P2
  - 16 KB: P2
  - 20 KB: P1
  - 24 KB: P1
  - 28 KB: P2

- **CPU's TLB:**
  - | Valid | Virt | Phys |
  - |------|------|------|
  - | 1    | 1    | 5    |
  - | 1    | 2    | 4    |
  - | 0    |      |      |
  - | 0    |      |      |
How many TLB lookups?

(assume 1KB pages)

```c
int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}
```
How many TLB lookups?
(assume 1KB pages)

```c
int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}

2048/sizeof(int) = 512
```
How many TLB “misses”?

(assume 1KB pages)

```c
int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}
```
How many TLB “misses”?
(assume 1KB pages)

```c
int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}
```

if a%4096 is 0, then 2 else 3
Miss rate?
(assume 1KB pages)

int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}

2/512 = 0.4\% or 3/512 = 0.6\%
Hit rate?
(assume 1KB pages)

```c
int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}
```

510/512 = **99.6%** or 509/512 = **99.4%**
Outline

What work can we eliminate?

Basic strategy.

Workloads, systems, metrics.

Context switching and security.
Reasoning about TLB

**Workload**: series of loads/stores to accesses

**TLB**: chooses entries to store in CPU

**Metric**: performance (i.e., hit rate)

TLB “algebra”, given 2 variables, find the 3rd:

\[ f(W, T) = M \]
Reasoning about TLB

**Workload**: series of loads/stores to accesses

**TLB**: chooses entries to store in CPU

**Metric**: performance (i.e., hit rate)

TLB “algebra”, given 2 variables, find the 3rd:

\[ f(W, T) = M \]
Sequential array accesses can almost always hit in the TLB, and so are very fast!

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

What pattern would be slow?
- highly random, with no repeat accesses
Workload Characteristics

Workload A

```c
int sum = 0;
for (i=0; i<2048; i++) {
    sum += a[i];
}
```

Workload B

```c
int sum = 0;
srand(1234);
for (i=0; i<1000; i++) {
    sum += a[rand() % N];
}
srand(1234);
for (i=0; i<1000; i++) {
    sum += a[rand() % N];
}
Spatial Locality

Temporal Locality
Workload Locality

**Spatial Locality**: future access will be to nearby addresses

**Temporal Locality**: future access will be repeats to the same data
Workload Locality

Spatial Locality: future access will be to nearby addresses

Temporal Locality: future access will be repeats to the same data

What TLB characteristics are best for each type?
A couple policies

**LRU**: evict least-recently used a TLB slot is needed

**Random**: randomly choose entries to evict

When is each better?
LRU Troubles

virtual addresses: 0 1 2 3 4

Valid  Virt  Phys
0      0     0
0      0     0
0      0     0
LRU Troubles

virtual addresses: 0 1 2 3 4

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: 0 1 2 3 4

miss!

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: [0 1 2 3 4]

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

miss!
LRU Troubles

<table>
<thead>
<tr>
<th>Virtual Addresses:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phys</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: 0 1 2 3 4

miss!

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: 0 1 2 3 4

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses:

0 1 2 3 4

miss!

Valid | Virt | Phys
--- | --- | ---
1 | 0 | ?
1 | 1 | ?
1 | 2 | ?
0 | 3 | ?
LRU Troubles

virtual addresses:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: 0 1 2 3 4

miss!

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: 0 1 2 3 4

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses:

0 1 2 3 4

miss!

Valid  Virt  Phys
1  4  ?
1  0  ?
1  2  ?
0  3  ?
LRU Troubles

virtual addresses: 0 1 2 3 4

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: 0 1 2 3 4
miss!

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Troubles

virtual addresses: [Slot 0, Slot 1, Slot 2, Slot 3, Slot 4]

miss!

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>?</td>
</tr>
</tbody>
</table>
A couple policies

**LRU**: evict least-recently used a TLB slot is needed

**Random**: randomly choose entries to evict

When is each better? Sometimes random is better than a “smart” policy!
Outline

What work can we eliminate?

Basic strategy.

Workloads, systems, metrics.

Context switching and security.
Context Switches

What happens if a process uses the cached TLB entries from another process?
Context Switches

What happens if a process uses the cached TLB entries from another process?

Solutions?
Context Switches

What happens if a process uses the cached TLB entries from another process?

Solutions?
- flush TLB on each switch
- remember which entries are for each process
Addres Space Identifier

Tag each TLB entry with an 8-bit ASID
- how many ASIDs do we get?
- why not use PIDs?
- what if there are more PIDs than ASIDs?
The diagram illustrates the relationship between virtual and physical memory addresses. The TLB (Translation Lookaside Buffer) is shown to manage the translation of virtual addresses to physical addresses.

### TLB:

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

### Virtual
- load 0x1444

### Physical
- load 0x2444

The page tables (PT) for ASIDs 11 and 12 are also shown, with pointers to PTBR (Page Table Base Register) for each ASID.

- P1 page table (ASID 11):
  - Valid Virt Phys ASID
  - 0 1 9 11
  - 1 1 5 11
  - 1 1 2 12
  - 1 0 1 11

- P2 page table (ASID 12):
  - Valid Virt Phys ASID
  - 6 2 3 ...

The diagram also shows the mapping from virtual to physical addresses, with a focus on the page table entries and the TLB entries.
TLB:

<table>
<thead>
<tr>
<th></th>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>9</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
Virtual load 0x1444
load 0x1444

Physical load 0x2444
load 0x5444

TLB:

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>
Address Space Identifier

Context switches are expensive.

Even with ASID, other processes “pollute” the TLB.
Who changes the TLB?

H/W or OS?
Who changes the TLB?

**H/W** or **OS**?

**H/W**: CPU must know where pagetables are
- CR3 on x86
- pagetable structure not flexible
- “walk” the pagetable

**OS**: CPU traps into OS upon TLB miss
- how to avoid double traps?
- more modern
Security

Modifying TLB entries is privileged
- otherwise what could you do?

Need same protection bits in TLB as pagetable
- rwx
Measurement Demo
(if enough time)