Paging Problems

- Page tables are too slow
- Page tables are too big
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
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)

```plaintext
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```
Trace the Memory Accesses (w/ TBL)

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

Trace the Memory Accesses (w/ TBL)

P1’s page table

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

CPU’s TLB cache

<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>
</tbody>
</table>

Virt
- load 0x3000
- load 0x3004
- load 0x3008
- load 0x300C

Phys
- ...

...
Trace the Memory Accesses (w/ TBL)

### P1’s page table

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

### CPU’s TLB cache

<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>

**Virt**
- load 0x3000
- load 0x3004
- load 0x3008
- load 0x300C

**Phys**
- Miss
- ...

---

10
Trace the Memory Accesses (w/ TBL)

P1’s page table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</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 0x3000
- load 0x3004
- load 0x3008
- load 0x300C
- ...

Phys
- load 0x100C
Trace the Memory Accesses (w/ TBL)

P1’s page table

\[
\begin{array}{c|c|c}
\text{Virt} & \text{Phys} \\
0 & 5 & \text{x100C} \\
1 & 3 & \text{x7000} \\
2 & 4 & \text{} \\
3 & 7 & \text{} \\
\end{array}
\]

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</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 0x3000
- load 0x3004
- load 0x3008
- load 0x300C
- ...

Phys
- load 0x100C
- load 0x7000
- ...

Trace the Memory Accesses (w/ TBL)

P1’s page table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</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 0x3000
- load 0x3004
- load 0x3008
- load 0x300C

Phys
- load 0x100C
- load 0x7000
Trace the Memory Accesses (w/ TBL)

### P1’s page table

<table>
<thead>
<tr>
<th></th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

### CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</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>

**Hit**

**Virt**
- load 0x3000
- load 0x3004
- load 0x3008
- load 0x300C

**Phys**
- load 0x100C
- load 0x7000 (TLB hit)
- ...

---
Trace the Memory Accesses (w/ TBL)

P1’s page table

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Phys
load 0x100C (TLB hit)
load 0x7000
load 0x7004

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</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>

...
Trace the Memory Accesses (w/ TBL)

P1’s page table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</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 0x3000
- load 0x3004
- load 0x3008
- load 0x300C

Phys
- load 0x100C
- load 0x7000 (TLB hit)
- load 0x7004 (TLB hit)
- load 0x7008 (TLB hit)
- load 0x700C
Trace the Memory Accesses (w/ TBL)

P1’s page table

<table>
<thead>
<tr>
<th>virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</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 0x3000
load 0x3004
load 0x3008
load 0x300C
...
load 0x1000

Phys
load 0x100C
load 0x7000 (TLB hit)
load 0x7004 (TLB hit)
load 0x7008 (TLB hit)
load 0x700C
Trace the Memory Accesses (w/ TBL)

P1’s page table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virt
- load 0x3000
- load 0x3004 (TLB hit)
- load 0x3008 (TLB hit)
- load 0x300C
- ... (Miss)
- load 0x1000

Phys
- load 0x100C
- load 0x7000 (TLB hit)
- load 0x7004 (TLB hit)
- load 0x7008 (TLB hit)
- load 0x700C
- load 0x100F
Trace the Memory Accesses (w/ TBL)

P1’s page table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

Valid

Virt
- load 0x3000
- load 0x3004 (TLB hit)
- load 0x3008 (TLB hit)
- load 0x300C (TLB hit)
- ... (TLB hit)
- load 0x1000

Phys
- load 0x100C
- load 0x7000 (TLB hit)
- load 0x7004 (TLB hit)
- load 0x7008 (TLB hit)
- load 0x700C
- load 0x100F
- load 0x3000
Trace the Memory Accesses (w/ TBL)

P1’s page table

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU’s TLB cache

Virt
- load 0x3000
- load 0x3004
- load 0x3008
- load 0x300C
- load 0x1000

Phys
- load 0x100C
- load 0x7000 (TLB hit)
- load 0x7004 (TLB hit)
- load 0x7008 (TLB hit)
- load 0x700C
- load 0x100F
- load 0x3000
Trace the Memory Accesses (w/ TBL)

P1’s page table

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU’s TLB cache

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virt</th>
<th>Phys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virt

- load 0x3000
- load 0x3004
- load 0x3008
- load 0x300C
- ... (TLB hit)
- load 0x1000
- load 0x1004

Phys

- load 0x100C (TLB hit)
- load 0x7000 (TLB hit)
- load 0x7004 (TLB hit)
- load 0x7008 (TLB hit)
- load 0x700C (TLB hit)
- load 0x1000F (TLB hit)
How Many TLB Lookups

Assume 4KB pages

```c
int sum = 0;
for (i=0; i<1024; i++) {
    sum += a[i];
}
```
How Many TLB Lookups

Assume 4KB pages

```c
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

```c
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

```c
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

TLB miss rate: 1 / 1024 = 0.09%
How Many TLB Lookups

Assume 4KB pages

```c
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
- 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

CPU

logical address

p  d

logical address

p  d

offset

TLB

physical address

physical memory

TLB hit

page table

TLB miss

page number  frame number
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?

<table>
<thead>
<tr>
<th></th>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>100</td>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>P2</td>
<td>10</td>
<td>170</td>
<td>1</td>
<td>rwx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>—</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>1</td>
<td>rwx</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>170</td>
<td>1</td>
<td>rwx</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>VPN</th>
<th>PFN</th>
<th>valid</th>
<th>prot</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>101</td>
<td>1</td>
<td>r-x</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>101</td>
<td>1</td>
<td>r-x</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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

Workload B

```c
int sum = 0;
srand(1234);
for (i=0; i<512; i++) {
    sum += a[rand()] % N;
}
srand(1234);
for (i=0; i<512; i++) {
    sum += a[rand()] % N;
}
```
Access Patterns

Workload A

Workload B
Access Patterns

Workload A

Workload B

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?
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

Virt addr
0
1
2
3
4

CPU’s TLB cache

<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 Trouble

Virt addr
0
1
2
3
4

CPU’s TLB cache

<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 Trouble

Virt addr

0
1
2
3
4

CPU’s TLB cache

<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>

TLB miss
LRU Trouble

Virt addr

0
1
2
3
4

CPU's TLB cache

<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 Trouble

CPU’s TLB cache

<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>

Virt addr

0 1 2 3

TLB miss
LRU Trouble

Virt addr

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

CPU’s TLB cache

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Virt</td>
<td>Phys</td>
</tr>
<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 Trouble

Virt addr
0
1
2
3
4

CPU’s TLB cache

<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>

TLB miss
LRU Trouble

Virt addr
0
1
2
3
4

CPU’s TLB cache

<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>1</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Trouble

CPU’s TLB cache

<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>1</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

Virt addr

TLB miss
LRU Trouble

Virt addr

0
1
2
3
4

CPU’s TLB cache

<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>1</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Trouble

<table>
<thead>
<tr>
<th>Virt addr</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

CPU’s TLB cache

<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>1</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Trouble

### CPU’s TLB cache

<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>1</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

TLB miss
LRU Trouble

Virt addr
0
1
2
3
4

CPU’s TLB cache

<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>1</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>
LRU Trouble

Virt addr
0
1
2
3
4

CPU’s TLB cache

<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>1</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

TLB miss
Takeaway

- LRU

- Random

- When is each better?
  - Sometimes random is better than a “smart” policy!
Review: Page Table
Virtual => Physical Addr Mapping

- We need a general mapping mechanism

- What data structure is good?
  - Big array
Virtual => Physical Addr Mapping

- We need a general mapping mechanism

- What data structure is good?
  - Big array
  - (aka linear page table)
A Simple Page Table Example

Virtual mem

Phys mem

Page tables

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

P1

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

P2

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Virtual mem

Phys mem

Page tables
A Simple Page Table Example

Virtual mem

physical translation info

important page status info

Page tables

1st PTE of P1

Phys

1st PTE
of P1

P1

P2

PFN

G

PAT

D

A

PCD

PWT

U/S

R/W

P

0

1

2

3

4

5

6

7
Paging Problems

- Page tables are too slow *(just covered)*
- Page tables are too big *(now)*
How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: \textbf{4MB} assuming each page-table entry is of 4 bytes
    - $2^{32 - \log(4\text{KB})} \times 4 = 4\text{MB}$
How Large are Page Tables?

- A linear page table array for 32-bit address space (\(2^{32}\) bytes) and 4KB page (\(2^{12}\) bytes)
  - How many pages: \(2^{20}\) pages
  - How much memory: \textbf{4MB} assuming each page-table entry is of 4 bytes
    - \(2^{(32-\log(4KB))} \times 4 = 4MB\)
How Large are Page Tables?

- A linear page table array for 32-bit address space \(2^{32}\) bytes) and 4KB page \(2^{12}\) bytes)
  - How many pages: \(2^{20}\) pages
  - How much memory: \(4\text{MB}\) assuming each page-table entry is of 4 bytes
    - \(2^{\log(4\text{KB})} = 4\text{MB}\)

How Large are Page Tables?

- A linear page table array for 32-bit address space \((2^{32} \text{ bytes})\) and 4KB page \((2^{12} \text{ bytes})\)
  - How many pages: \(2^{20}\) pages
  - How much memory: \(4\text{MB}\) assuming each page-table entry is of 4 bytes
    - \(2^{^{32-\log(4\text{KB})}} \times 4 = 4\text{MB}\)

VPN bits
How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: **4MB** assuming each page-table entry is of 4 bytes
    - $2^{(32-\log(4KB))} \times 4 = 4\text{MB}$

Num of virt pages
How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - $2^{(32-\log(4\text{KB}))} \times 4 = 4\text{MB}$
Page Tables are Too Big

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - $2^{(32-\log(4\text{KB}))} \times 4 = 4\text{MB}$

- One page table for one process!
  - A system with 100 processes: 400MB only for storing page tables in memory

- Solution??
Naïve Solution

- Reduce the granularity
  - by increasing the page size
Naïve Solution

- Reduce the granularity
  - by *increasing* the page size

- Why are 4MB pages bad?
  - *Internal fragmentation!*
Fragmentation

Phys Mem

free

P1

free

P3

free

P2

free

P4

An allocated 4MB huge page of P4

allocated data

free

External frag.

Internal frag.
Approaches

- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table
Approaches

- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table
Linear Inverted Page Table

- Idea: Instead of keeping one page table per process, the system keeps a single page table that has an entry for each physical frame of the system.

- Each entry tells which process owns the page, and VPN to PFN translation.
Linear Inverted Page Table Example

The diagram illustrates the process of converting a logical address into a physical address using a linear inverted page table. The logical address is first passed through the linear inverted page table, which contains page directory entries. Each entry contains the page directory index (id) and page index (p) for the corresponding page. The page directory index is then used to look up the page in the page table, which provides the physical address. The physical address is then used to access the memory location.
Linear Inverted Page Table

- Idea: Instead of keeping one page table per process, the system keeps a single page table that has an entry for each physical frame of the system.
- Each entry tells which process owns the page, and VPN to PFN translation.

- Pros: Extreme memory savings
- Cons: A linear search is expensive
  - Solution??
Approaches

- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table
Hashed Inverted Page Table

- For large address spaces, a hashed page table can be used, with the hash value being the VPN.

- Idea:
  - A PTE contains a chain of elements hashing to the same location (to handle collisions) within PT.
  - Each element has three fields: (a) VPN, (b) PFN, (c) a pointer to the next element in the linked list.
  - VPNs are compared in this chain searching for a match. If a match is found, the corresponding PFN is extracted.
Hashed Inverted Page Table Example

logical address

p d

hash function

hash table

q s

r d

p r

physical address

physical memory

...
Approaches

- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table
Multi-level Page Table

- Idea:
  - Break the page table into pages (the entire page table is **paged**!)
  - Only have pieces with $>0$ valid entries
    - Don’t allocate the page of page table of the entire page of page-table entries is invalid

- Used by x86

- A simple technique is a two-level page table
Two-level Paging

- A logical address (on 32-bit machine with 4KB page size) is divided into
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- A page table entry is 4 bytes
- Since the page table is paged, the page number is further divided into
  - a 10-bit page number
  - a 10-bit page offset
- The logical address is as follows:

<table>
<thead>
<tr>
<th>page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

where $p_1$ is an index into the outer page table, and $p_2$ is the displacement within the page of the outer page table
Two-level Page Table Example

also called a page directory
Address Translation Scheme

- Address translation scheme for a two-level 32-bit paging architecture
> 2 Levels

- Problem: page directories may not fit in a page

- Solution:
  - Split page directories into pieces
  - Use another page dir to refer to the page dir pieces
> 2 Levels

- Problem: page directories **may not fit** in a page

- Solution:
  - Split page directories into pieces
  - Use another page dir to refer to the page dir pieces

- Possible to extend to 3- or 4-level

- E.g., 64-bit Ultra-SPARC would need 7 levels of paging
> 2 Levels

- Problem: page directories **may not fit** in a page

- Solution:
  - Split page directories into pieces
  - Use another page dir to refer to the page dir pieces

```
<table>
<thead>
<tr>
<th>VPN</th>
<th>PD idx 0</th>
<th>PD idx 1</th>
<th>PT idx</th>
<th>OFFSET</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>
```
Multi-level Page Table Example

http://web.eecs.utk.edu/~mbeck/classes/cs560/560/oldtests/t2/2003/Answers.html