Betriebssysteme

8. Tutorium - Caches, IPC

Peter Bohner

20. Dezember 2023

ITEC - Operating Systems Group

Caches

Caches

How many do you typically have in your computer?

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

And why do you have multiple?

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

And why do you have multiple?

- Fast caches are expensive (and size limited) $\Rightarrow$ Small

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

And why do you have multiple?

- Fast caches are expensive (and size limited) $\Rightarrow$ Small
- Multiple levels of slower but larger caches arranged in a *cache hierarchy*

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

And why do you have multiple?

- Fast caches are expensive (and size limited) $\Rightarrow$ Small
- Multiple levels of slower but larger caches arranged in a *cache hierarchy*

Why do caches even work?

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

And why do you have multiple?

- Fast caches are expensive (and size limited) $\Rightarrow$ Small
- Multiple levels of slower but larger caches arranged in a *cache hierarchy*

Why do caches even work?

Two main principles:

- Temporal locality: Memory that was accessed will be accessed again soon

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

And why do you have multiple?

- Fast caches are expensive (and size limited) $\Rightarrow$ Small
- Multiple levels of slower but larger caches arranged in a *cache hierarchy*

Why do caches even work?

Two main principles:

- Temporal locality: Memory that was accessed will be accessed again soon
- Spatial locality: If address X was accessed, $X \pm 1$ will likely be accessed soon $\Rightarrow$ Don't just cache single words but entire *lines*. How large?

Caches

How many do you typically have in your computer?

Quite a few! Typically L1, L2, L3 <maybe more> and your RAM

And why do you have multiple?

- Fast caches are expensive (and size limited) $\Rightarrow$ Small
- Multiple levels of slower but larger caches arranged in a *cache hierarchy*

Why do caches even work?

Two main principles:

- Temporal locality: Memory that was accessed will be accessed again soon
- Spatial locality: If address X was accessed, $X \pm 1$ will likely be accessed soon $\Rightarrow$ Don't just cache single words but entire *lines*. How large? Probably ≥ 64 Bytes for *reasons*

Caches

Cache lines

Why is a cache line useful? Can't we just make N memory accesses?

Caches

Cache lines

Why is a cache line useful? Can't we just make N memory accesses? Reading a chunk from DRAM is *much* more efficient than reading them one after another

Cache Placement Policies

What policies do you know?

- Fully Associative

Cache Placement Policies

What policies do you know?

- Fully Associative
- Set-Associative

Cache Placement Policies

What policies do you know?

- Fully Associative
- Set-Associative
- Direct Mapped

Cache Placement Policies - Fully Associative

What is that?

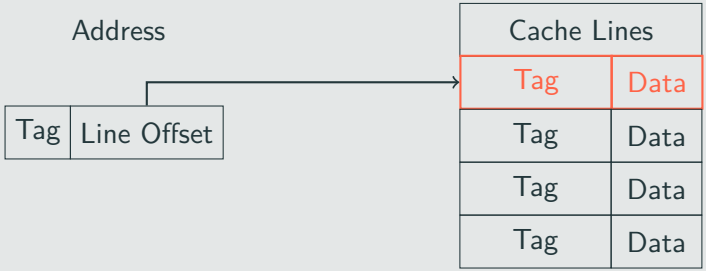
Address

Tag	Line Offset

Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

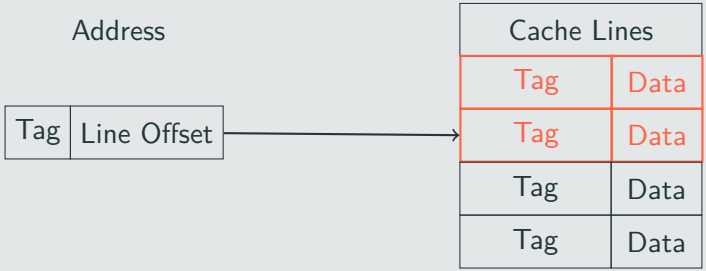
Cache Placement Policies - Fully Associative

What is that?



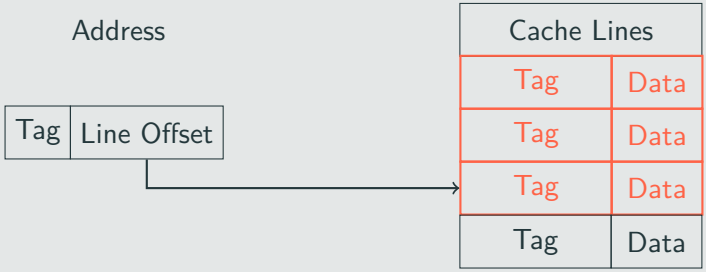
Cache Placement Policies - Fully Associative

What is that?



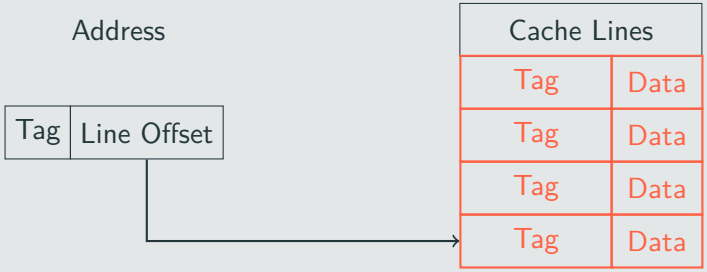
Cache Placement Policies - Fully Associative

What is that?



Cache Placement Policies - Fully Associative

What is that?



Cache Placement Policies - Fully Associative

What is that?

When can cache misses happen here?

Cache Placement Policies - Fully Associative

What is that?

When can cache misses happen here?

- Never accessed that address before. Called a

Cache Placement Policies - Fully Associative

What is that?

When can cache misses happen here?

- Never accessed that address before. Called a Compulsory Miss

Cache Placement Policies - Fully Associative

What is that?

When can cache misses happen here?

- Never accessed that address before. Called a Compulsory Miss
- Cache was full and it was evicted. Called a

Cache Placement Policies - Fully Associative

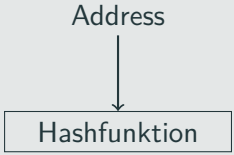
What is that?

When can cache misses happen here?

- Never accessed that address before. Called a Compulsory Miss
- Cache was full and it was evicted. Called a Capacity Miss

Cache Placement Policies - Direct Mapped

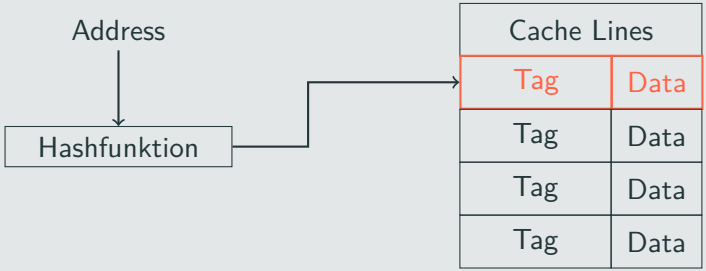
What is that?



Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

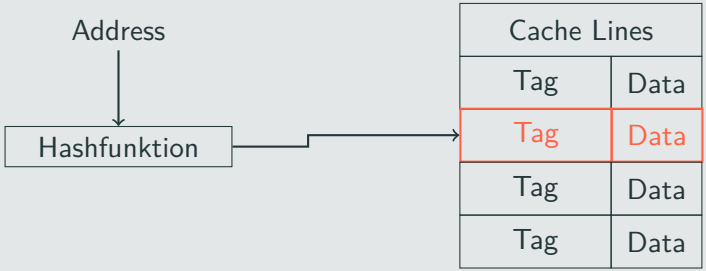
Cache Placement Policies - Direct Mapped

What is that?



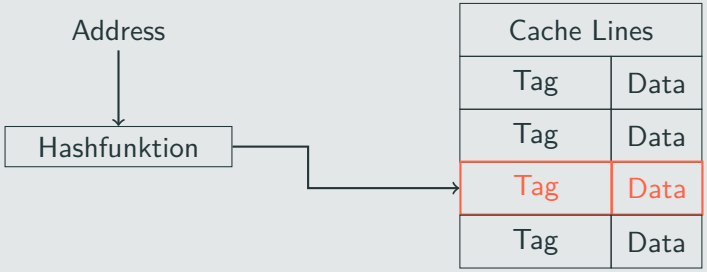
Cache Placement Policies - Direct Mapped

What is that?



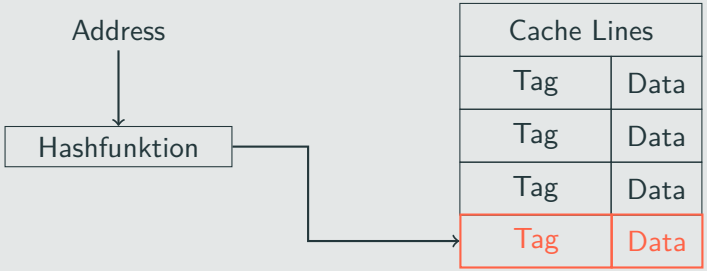
Cache Placement Policies - Direct Mapped

What is that?



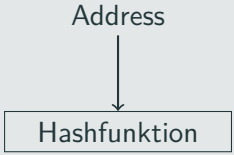
Cache Placement Policies - Direct Mapped

What is that?



Cache Placement Policies - Direct Mapped

What is that?

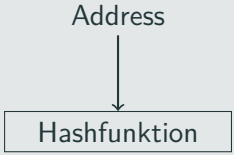


Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

When can cache misses happen here?

Cache Placement Policies - Direct Mapped

What is that?



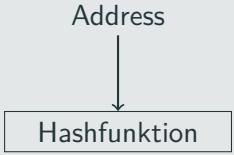
Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

When can cache misses happen here?

- Never accessed that address before. Called a

Cache Placement Policies - Direct Mapped

What is that?



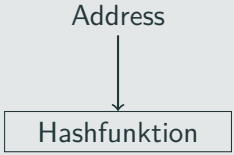
Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

When can cache misses happen here?

- Never accessed that address before. Called a Compulsory Miss

Cache Placement Policies - Direct Mapped

What is that?



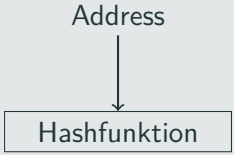
Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

When can cache misses happen here?

- Never accessed that address before. Called a Compulsory Miss
- Cache was full and it was evicted. Called a

Cache Placement Policies - Direct Mapped

What is that?



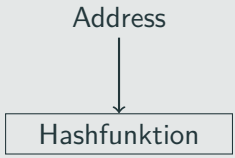
Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

When can cache misses happen here?

- Never accessed that address before. Called a Compulsory Miss
- Cache was full and it was evicted. Called a Capacity Miss

Cache Placement Policies - Direct Mapped

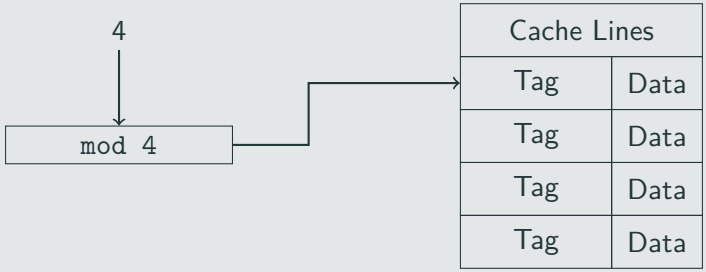
What is that?



Cache Lines	
Tag	Data
Tag	Data
Tag	Data
Tag	Data

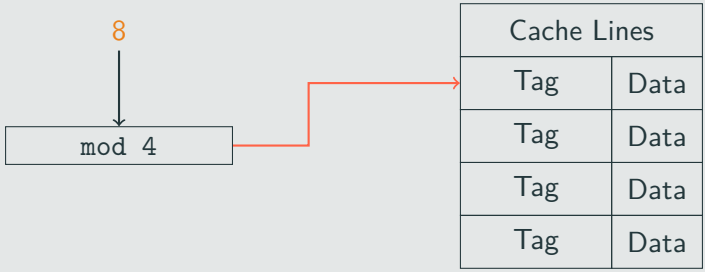
Cache Placement Policies - Direct Mapped

What is that?



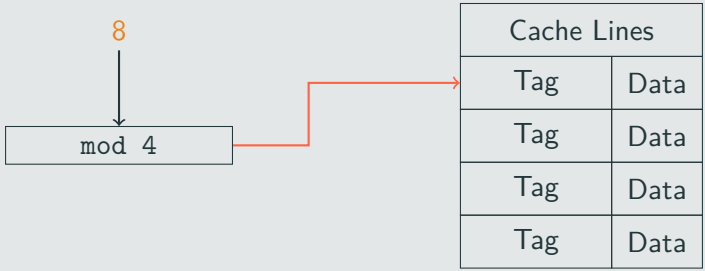
Cache Placement Policies - Direct Mapped

What is that?



Cache Placement Policies - Direct Mapped

What is that?

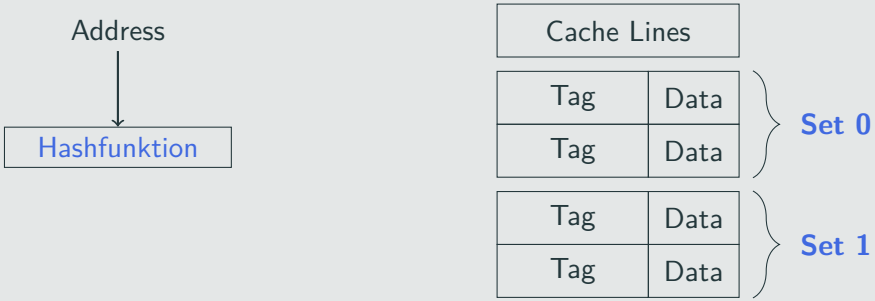


Conflict misses

There was space, but it was mapped to the same slot!

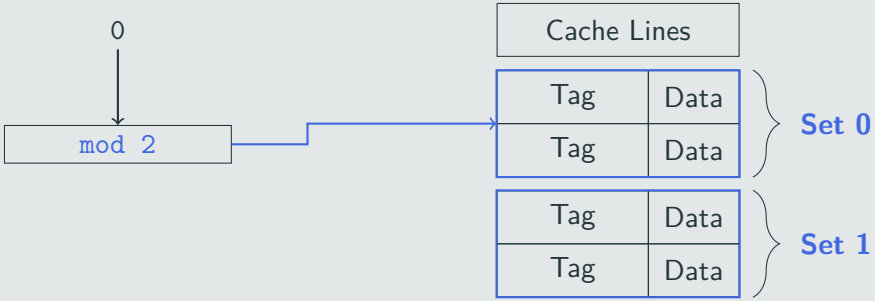
Cache Placement Policies - Set associative

What is that?



Cache Placement Policies - Set associative

What is that?

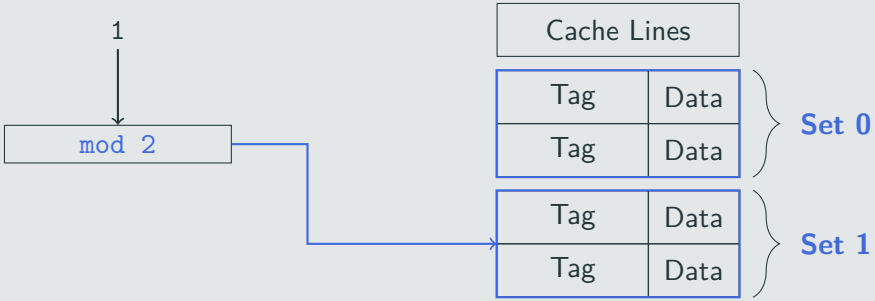


Insertion

Find the correct *set* using the index, then treat each set as a *Fully Associative* cache.

Cache Placement Policies - Set associative

What is that?

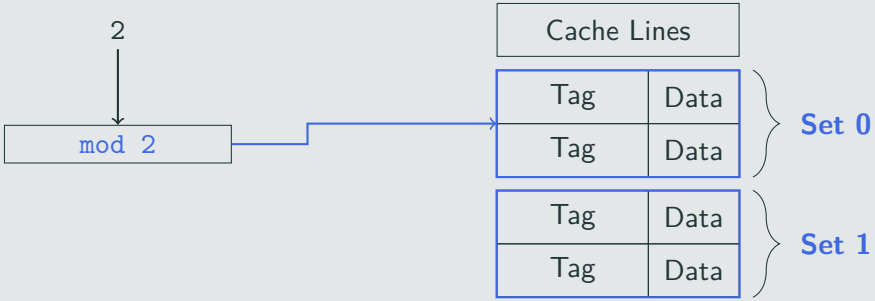


Insertion

Find the correct *set* using the index, then treat each set as a *Fully Associative* cache.

Cache Placement Policies - Set associative

What is that?

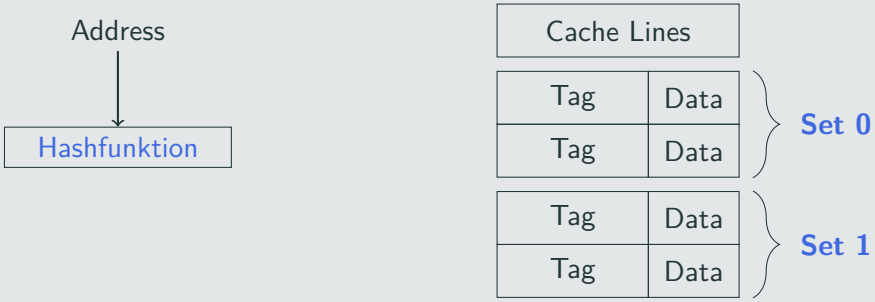


Insertion

Find the correct *set* using the index, then treat each set as a *Fully Associative* cache.

Cache Placement Policies - Set associative

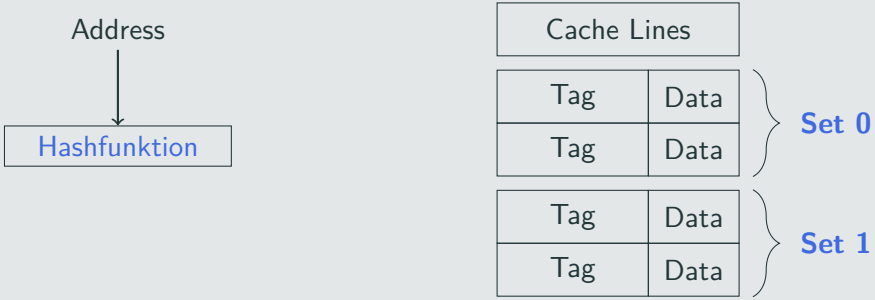
What is that?



Conflict misses?

Cache Placement Policies - Set associative

What is that?



Conflict misses!

There was space, but it was mapped to the same *set*!

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

- Write Through:

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

- Write Through: Update the cache *and the main memory*

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

- Write Through: Update the cache *and the main memory*
- Write Back:

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

- Write Through: Update the cache *and the main memory*
- Write Back: Update only the cache. Update main memory *on eviction*

... an address *not* in the cache?

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

- Write Through: Update the cache *and the main memory*
- Write Back: Update only the cache. Update main memory *on eviction*

... an address *not* in the cache?

- Write-allocate:

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

- Write Through: Update the cache *and the main memory*
- Write Back: Update only the cache. Update main memory *on eviction*

... an address *not* in the cache?

- Write-allocate: First load it into the cache. Then see above

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

- Write Through: Update the cache *and the main memory*
- Write Back: Update only the cache. Update main memory *on eviction*

... an address *not* in the cache?

- Write-allocate: First load it into the cache. Then see above
- Write-to-memory:

Modifying Caches

You have a Cache and real memory behind

What do you do when you write to ...

... an address in the cache?

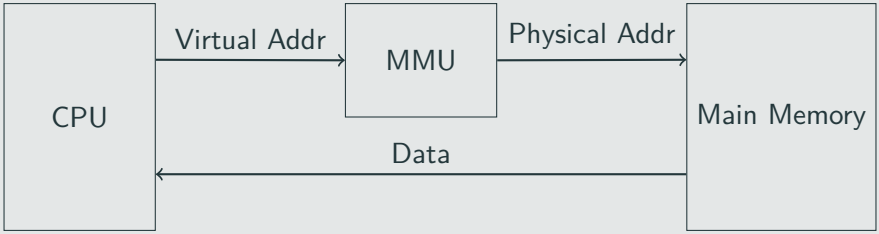
- Write Through: Update the cache *and the main memory*
- Write Back: Update only the cache. Update main memory *on eviction*

... an address *not* in the cache?

- Write-allocate: First load it into the cache. Then see above
- Write-to-memory: Don't load into cache, modify in memory

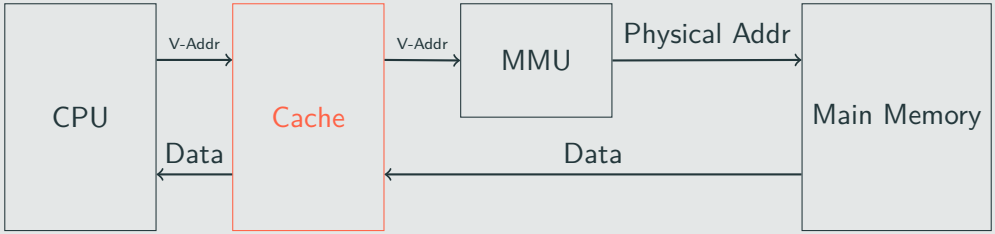
Placing The Cache

At which position do you put the cache?



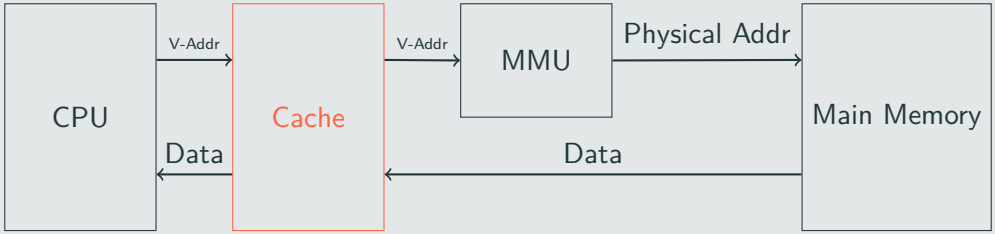
Placing The Cache

At which position do you put the cache?



Placing The Cache

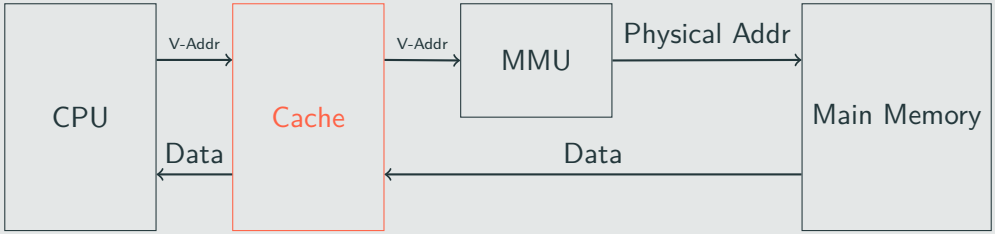
At which position do you put the cache?



What kind of addresses does this use for Index/Tag?

Placing The Cache

At which position do you put the cache?



What kind of addresses does this use for Index/Tag?

Virtually Indexed, Virtually Tagged (VIVT)

Placing The Cache

What kind of addresses does this use for Index/Tag?

Virtually Indexed, Virtually Tagged (VIVT)

Benefits? Drawbacks?

Placing The Cache

What kind of addresses does this use for Index/Tag?

Virtually Indexed, Virtually Tagged (VIVT)

Benefits? Drawbacks?

+ Fast, no address translation

Placing The Cache

What kind of addresses does this use for Index/Tag?

Virtually Indexed, Virtually Tagged (VIVT)

Benefits? Drawbacks?

+ Fast, no address translation
- Ambiguity Problem (The same virtual address might point to different physical addresses over time)

Placing The Cache

What kind of addresses does this use for Index/Tag?

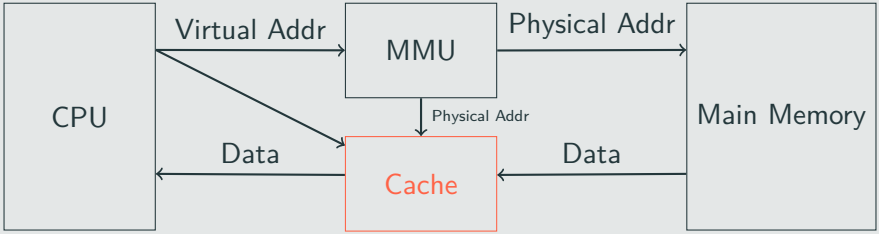
Virtually Indexed, Virtually Tagged (VIVT)

Benefits? Drawbacks?

+ Fast, no address translation
- Ambiguity Problem (The same virtual address might point to different physical addresses over time)
- Alias Problem (Multiple virtual addresses might point to the same physical address)

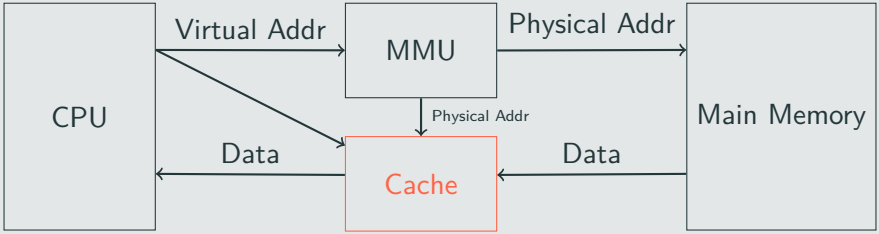
Placing The Cache

At which position do you put the cache?



Placing The Cache

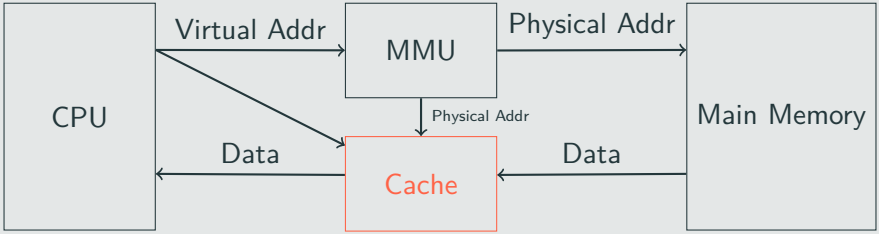
At which position do you put the cache?



What kind of addresses does this use for Index/Tag?

Placing The Cache

At which position do you put the cache?



What kind of addresses does this use for Index/Tag?

Virtually Indexed, Physically Tagged (VIPT)

Placing The Cache

What kind of addresses does this use for Index/Tag?

Virtually Indexed, Physically Tagged (VIPT)

Benefits? Drawbacks?

Placing The Cache

What kind of addresses does this use for Index/Tag?

Virtually Indexed, Physically Tagged (VIPT)

Benefits? Drawbacks?

+ Still relatively fast: Can lookup the index while the MMU works
+ Ambiguity solved, as it is tagged by physical address (iff

Placing The Cache

What kind of addresses does this use for Index/Tag?

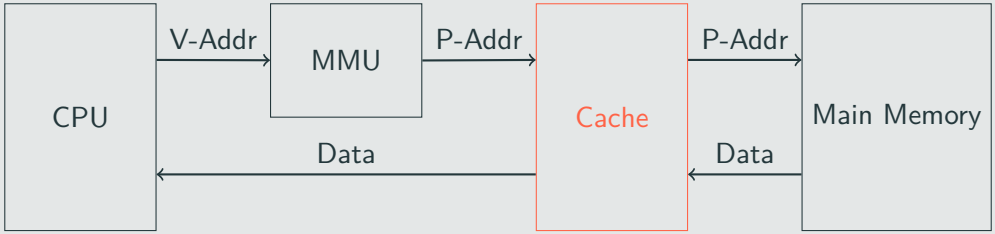
Virtually Indexed, Physically Tagged (VIPT)

Benefits? Drawbacks?

+ Still relatively fast: Can lookup the index while the MMU works
+ Ambiguity solved, as it is tagged by physical address (iff the entire PFN is used as a tag!)
- Alias Problem (sometimes!)

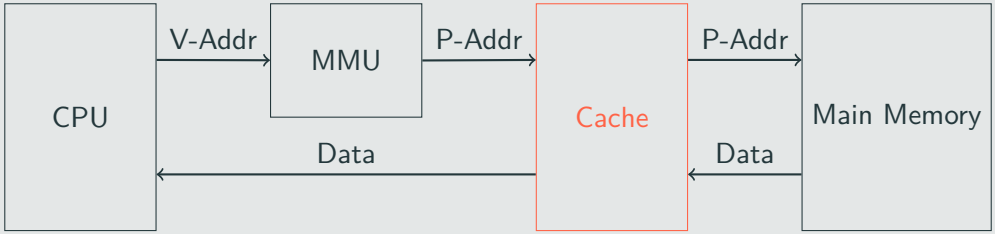
Placing The Cache

At which position do you put the cache?



Placing The Cache

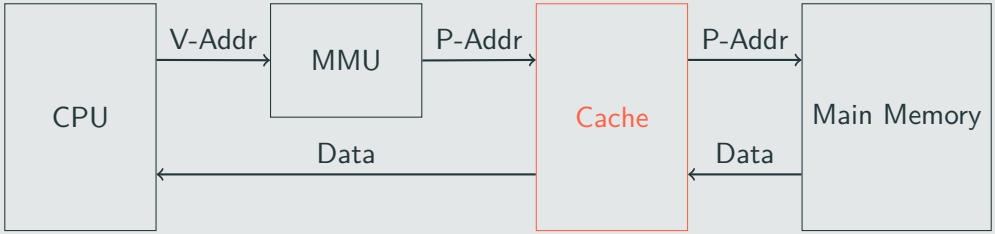
At which position do you put the cache?



What kind of addresses does this use for Index/Tag?

Placing The Cache

At which position do you put the cache?



What kind of addresses does this use for Index/Tag?

Physically Indexed, Physically Tagged (PIPT)

Placing The Cache

What kind of addresses does this use for Index/Tag?

Physically Indexed, Physically Tagged (PIPT)

Benefits? Drawbacks?

Placing The Cache

What kind of addresses does this use for Index/Tag?

Physically Indexed, Physically Tagged (PIPT)

Benefits? Drawbacks?

+ Transparent to CPU: no alias, no ambiguity

Placing The Cache

What kind of addresses does this use for Index/Tag?

Physically Indexed, Physically Tagged (PIPT)

Benefits? Drawbacks?

+ Transparent to CPU: no alias, no ambiguity
+ Coherency can be implemented in hardware

Placing The Cache

What kind of addresses does this use for Index/Tag?

Physically Indexed, Physically Tagged (PIPT)

Benefits? Drawbacks?

+ Transparent to CPU: no alias, no ambiguity
+ Coherency can be implemented in hardware
- Allocation conflicts

Placing The Cache

What kind of addresses does this use for Index/Tag?

Physically Indexed, Physically Tagged (PIPT)

Benefits? Drawbacks?

+ Transparent to CPU: no alias, no ambiguity
+ Coherency can be implemented in hardware
- Allocation conflicts

Allocation conflicts

- How well do you think *contiguous* virtual pages fit into the cache?

Placing The Cache

What kind of addresses does this use for Index/Tag?

Physically Indexed, Physically Tagged (PIPT)

Benefits? Drawbacks?

+ Transparent to CPU: no alias, no ambiguity
+ Coherency can be implemented in hardware
- Allocation conflicts

Allocation conflicts

- How well do you think *contiguous* virtual pages fit into the cache?
- Maybe not so well. The cache operates on *physical* addresses

$\Rightarrow$ Sequential virtual pages might be mapped to *conflicting* physical ones!

Inter-Process-Communication

IPC

How can different processes interact?

IPC

How can different processes interact?

- Shared memory (implicitly

IPC

How can different processes interact?

- Shared memory (implicitly as they share an address space and explicitly via a shared memory area)

IPC

How can different processes interact?

- Shared memory (implicitly as they share an address space and explicitly via a shared memory area)
- OS facilities (e.g. messages, pipes, Signals, sockets)

IPC

How can different processes interact?

- Shared memory (implicitly as they share an address space and explicitly via a shared memory area)
- OS facilities (e.g. messages, pipes, Signals, sockets)
- High level abstractions (files, database entries)

IPC - Getting Stuck

What mechanism do you need for IPC in an imperfect world?

IPC - Getting Stuck

What mechanism do you need for IPC in an imperfect world?

Timeouts! You don't want to wait for buggy programs or poor dead ones :(

IPC - Buffering

Asynchronous `send` Operations require a buffer. Where do you put that?

- Actually, *why do they need a buffer*?

IPC - Buffering

Asynchronous `send` Operations require a buffer. Where do you put that?

- Actually, *why do they need a buffer*? Sent but not yet received messages

IPC - Buffering

Asynchronous `send` Operations require a buffer. Where do you put that?

- Actually, *why do they need a buffer*? Sent but not yet received messages
- They can be in the *receiver's* address space, the *sender's* AS or in the kernel

IPC - Buffering

Asynchronous `send` Operations require a buffer. Where do you put that?

- Actually, *why do they need a buffer*? Sent but not yet received messages
- They can be in the *receiver's* address space, the *sender's* AS or in the kernel

Buffering in the Receiver's Address Space

- Is that a good idea?

IPC - Buffering

Asynchronous `send` Operations require a buffer. Where do you put that?

- Actually, *why do they need a buffer*? Sent but not yet received messages
- They can be in the *receiver's* address space, the *sender's* AS or in the kernel

Buffering in the Receiver's Address Space

- Is that a good idea? What happens when you have many clients?

IPC - Buffering

Asynchronous `send` Operations require a buffer. Where do you put that?

- Actually, *why do they need a buffer*? Sent but not yet received messages
- They can be in the *receiver's* address space, the *sender's* AS or in the kernel

Buffering in the Receiver's Address Space

- Is that a good idea? What happens when you have many clients?

$\Rightarrow$ Does not scale well. You either allow every sender to allocate memory for you (*ouch*) or you might run out with many clients

IPC - Buffering

Buffering in the Kernel

- Is that a good idea?

IPC - Buffering

Buffering in the Kernel

- Is that a good idea? What happens when you have many clients?

IPC - Buffering

Buffering in the Kernel

- Is that a good idea? What happens when you have many clients?

$\Rightarrow$ Does not scale well. You either allow every sender to allocate kernel memory (***ouch***) or you might run out with many clients

IPC - Buffering

Buffering in the Kernel

- Is that a good idea? What happens when you have many clients?

$\Rightarrow$ Does not scale well. You either allow every sender to allocate kernel memory (***ouch***) or you might run out with many clients

Buffering in the Sender's Address Space

- Sounds good?

IPC - Buffering

Buffering in the Kernel

- Is that a good idea? What happens when you have many clients?

$\Rightarrow$ Does not scale well. You either allow every sender to allocate kernel memory (***ouch***) or you might run out with many clients

Buffering in the Sender's Address Space

- Sounds good? Yea, we are running out of options.

IPC - Buffering

Buffering in the Kernel

- Is that a good idea? What happens when you have many clients?

$\Rightarrow$ Does not scale well. You either allow every sender to allocate kernel memory (***ouch***) or you might run out with many clients

Buffering in the Sender's Address Space

- Sounds good? Yea, we are running out of options. But it's mostly alright!

IPC - Buffering

Buffering in the Kernel

- Is that a good idea? What happens when you have many clients?

⇒ Does not scale well. You either allow every sender to allocate kernel memory (***ouch***) or you might run out with many clients

Buffering in the Sender's Address Space

- Sounds good? Yea, we are running out of options. But it's mostly alright!

\+ We only need to store it once: The sender has it in a buffer somewhere anyways

IPC - Buffering

Buffering in the Kernel

- Is that a good idea? What happens when you have many clients?

⇒ Does not scale well. You either allow every sender to allocate kernel memory (***ouch***) or you might run out with many clients

Buffering in the Sender's Address Space

- Sounds good? Yea, we are running out of options. But it's mostly alright!

\+ We only need to store it once: The sender has it in a buffer somewhere anyways

\+ Scales better, as each sender keeps their messages

± We need to tell the client when it can reclaim the buffer

The Joys of Synchronous IPC

You are a very popular process that receives and handles many messages.

- For simplicities sake you chose *Synchronous IPC*. What problem might occur with many clients?

The Joys of Synchronous IPC

You are a very popular process that receives and handles many messages.

- For simplicities sake you chose *Synchronous IPC*. What problem might occur with many clients?
- You spent your whole life *waiting for timeouts to expire*

The Joys of Synchronous IPC

You are a very popular process that receives and handles many messages.

- For simplicities sake you chose *Synchronous IPC*. What problem might occur with many clients?
- You spent your whole life *waiting for timeouts to expire*
- How could you solve that with a new syscall?

The Joys of Synchronous IPC

You are a very popular process that receives and handles many messages.

- For simplicities sake you chose *Synchronous IPC*. What problem might occur with many clients?
- You spent your whole life *waiting for timeouts to expire*
- How could you solve that with a new syscall? How does `send-and-receive`, which sends and instantly receives help?

The Joys of Synchronous IPC

You are a very popular process that receives and handles many messages.

- For simplicities sake you chose *Synchronous IPC*. What problem might occur with many clients?
- You spent your whole life *waiting for timeouts to expire*
- How could you solve that with a new syscall? How does `send-and-receive`, which sends and instantly receives help?
- The server can assume you are using it and set a zero timeout. After all, if you are using that syscall you *will* be waiting

Emulating Async IPC

How can you emulate asynchronous IPC using synchronous IPC?

Emulating Async IPC

How can you emulate asynchronous IPC using synchronous IPC?

- Use a Proxy thread

Emulating Async IPC

How can you emulate asynchronous IPC using synchronous IPC?

- Use a Proxy thread
  1. Copy message to proxy thread
  2. Proxy threads sends synchronously and might block until recipient calls receive

Emulating Async IPC

How can you emulate asynchronous IPC using synchronous IPC?

- Use a Proxy thread
  1. Copy message to proxy thread
  2. Proxy threads sends synchronously and might block until recipient calls receive

\+ Allows async I/O

\-

Emulating Async IPC

How can you emulate asynchronous IPC using synchronous IPC?

- Use a Proxy thread
  1. Copy message to proxy thread
  2. Proxy threads sends synchronously and might block until recipient calls receive

\+ Allows async I/O

\- How many messages can you send?

Emulating Async IPC

How can you emulate asynchronous IPC using synchronous IPC?

- Use a Proxy thread
  1. Copy message to proxy thread
  2. Proxy threads sends synchronously and might block until recipient calls receive

\+ Allows async I/O

\- How many messages can you send? Yea, one per thread...

Emulating Async IPC

How can you emulate Synchronous IPC using asynchronous IPC?

Emulating Async IPC

How can you emulate Synchronous IPC using asynchronous IPC?

What about

```
1  do {
2    res = async_send(message);
3  } while(res != MESSAGE_SENT);
```

Emulating Async IPC

How can you emulate Synchronous IPC using asynchronous IPC?

What about

```
1  do {
2    res = async_send(message);
3  } while(res != MESSAGE_SENT);
```

Will wait until it was *sent*, not until it was *received*.

Emulating Async IPC

How can you emulate Synchronous IPC using asynchronous IPC?

What about

```
1  do {
2    res = async_send(message);
3  } while(res != MESSAGE_SENT);
```

Will wait until it was *sent*, not until it was *received*.

⇒ Implement a very simple protocol involving multiple messages

Emulating Async IPC

How can you emulate Synchronous IPC using asynchronous IPC?

What about

```
1  do {
2    res = async_send(message);
3  } while(res != MESSAGE_SENT);
```

Will wait until it was *sent*, not until it was *received*.

⇒ Implement a very simple protocol involving multiple messages

⇒ Send `ACK` message on receiver side, wait for `ACK` to be received.

And receive?

Emulating Async IPC

How can you emulate Synchronous IPC using asynchronous IPC?

What about

```
1  do {
2    res = async_send(message);
3  } while(res != MESSAGE_SENT);
```

Will wait until it was *sent*, not until it was *received*.

⇒ Implement a very simple protocol involving multiple messages

⇒ Send `ACK` message on receiver side, wait for `ACK` to be received.

And receive?

Just loop until `async_receive` receives a message (that is not an `ACK`)

Pipes

`ls | less`

Write a C-program for Linux that creates two child processes, *ls* and *less* and uses an ordinary pipe to redirect the standard output of ls to the standard input of less.

Ende

XKCD 361 - Christmas Back Home

F R A G E N?

https://forms.gle/9CwJSKidKibubran9

Bis nächste Woche