

Operating Systems: On-disk paging

Raymond Namyst

Dept. of Computer Science

University of Bordeaux, France

<https://gforgeron.gitlab.io/se/>

Motivation

- When main memory (i.e. RAM) is full
 - No more page can be allocated, so
 - Process creation fails

Motivation

- When main memory (i.e. RAM) is full
 - No more page can be allocated, so
 - Process creation fails
 - Process growth fails

Motivation

- When main memory (i.e. RAM) is full
 - No more page can be allocated, so
 - Process creation fails
 - Process growth fails
 - Lazy allocation fails

Motivation

- When main memory (i.e. RAM) is full
 - No more page can be allocated, so
 - Process creation fails
 - Process growth fails
 - Lazy allocation fails
 - Copy-on-Write fails

Motivation

- When main memory (i.e. RAM) is full
 - No more page can be allocated, so
 - Process creation fails
 - Process growth fails
 - Lazy allocation fails
 - Copy-on-Write fails
 - What a mess! 🤯

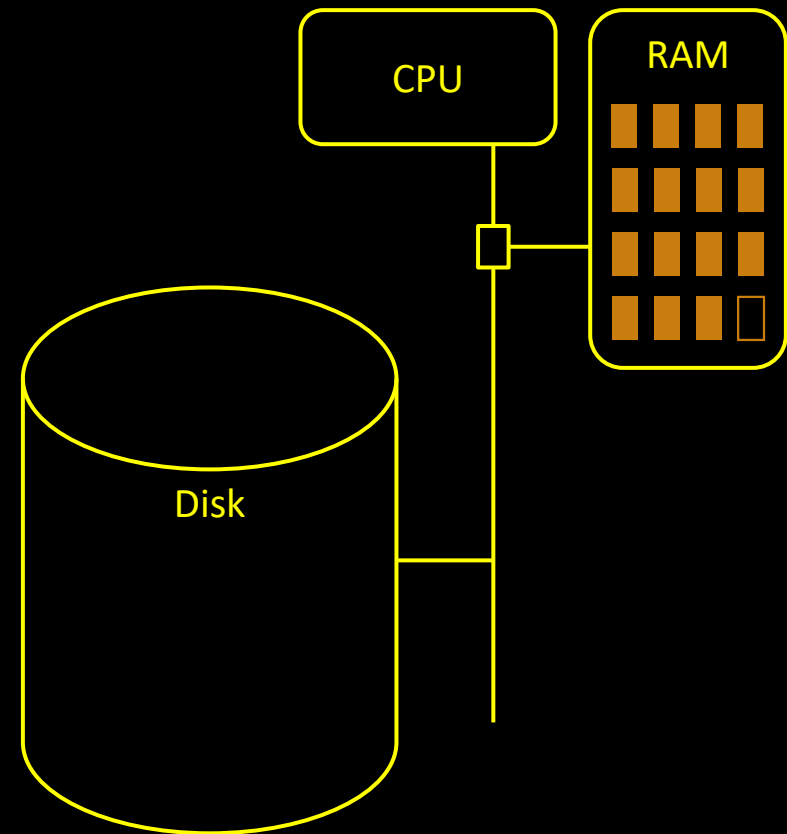
Motivation

- When main memory (i.e. RAM) is full
 - No more page can be allocated, so
 - Process creation fails
 - Process growth fails
 - Lazy allocation fails
 - Copy-on-Write fails
 - What a mess! 🤯
- Disks have a larger capacity
 - We could use disk space as a RAM extension

Motivation

- When the RAM is full, we could satisfy upcoming allocations on disk

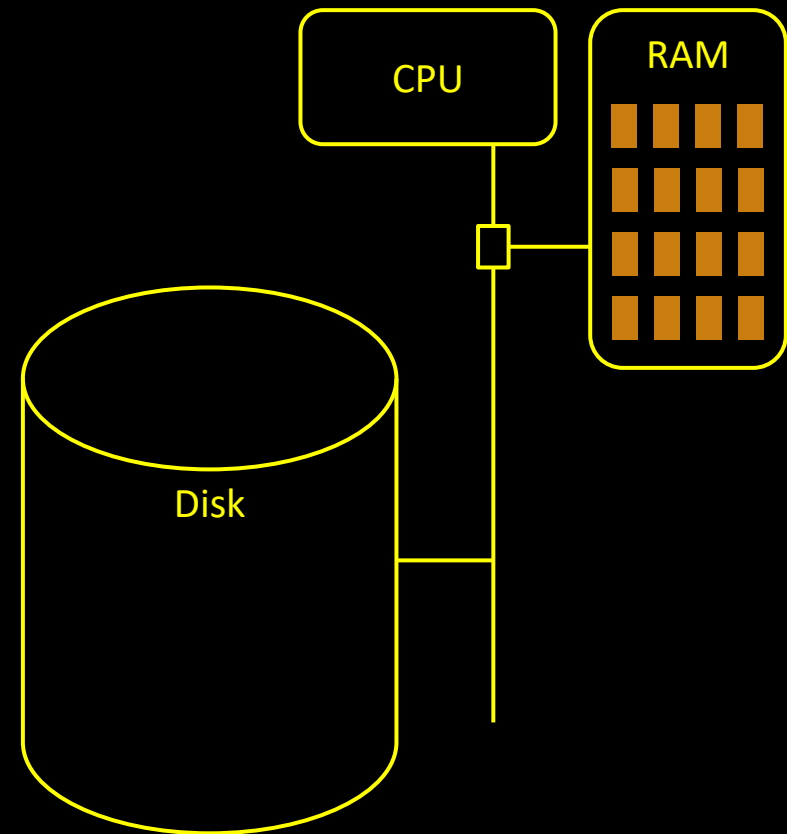
```
kernel    get_free_page ()  
          {  
            ...  
          }
```



Motivation

- When the RAM is full, we could satisfy upcoming allocations on disk

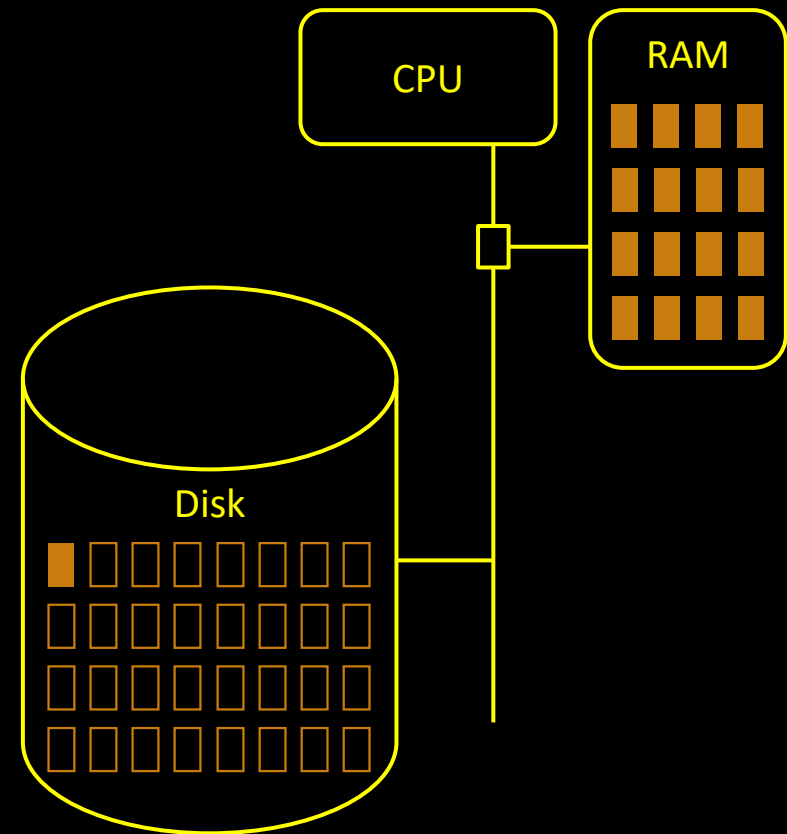
```
kernel    get_free_page ()  
          {  
            ...  
          }
```



Motivation

- When the RAM is full, we could satisfy upcoming allocations on disk

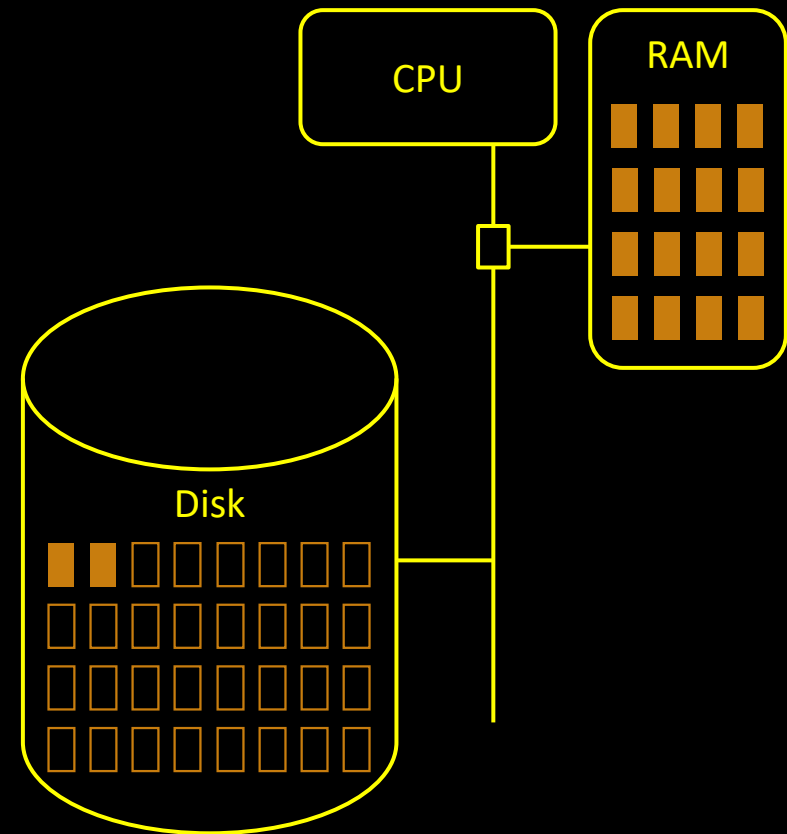
```
kernel    get_free_page ()  
          {  
            ...  
          }
```



Motivation

- When the RAM is full, we could satisfy upcoming allocations on disk

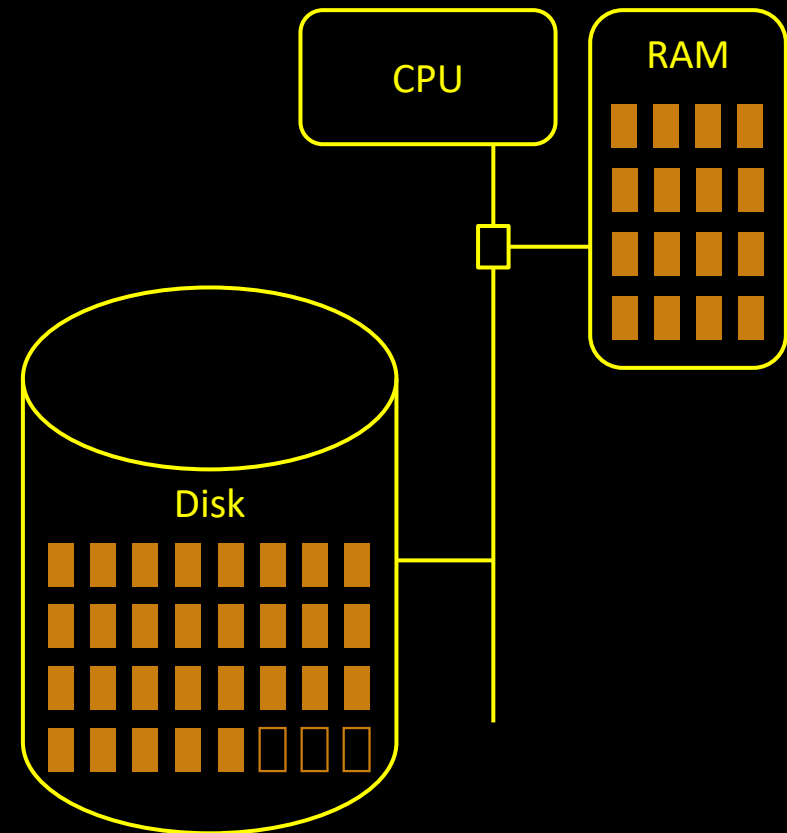
```
kernel    get_free_page ()  
          {  
            ...  
          }
```



Motivation

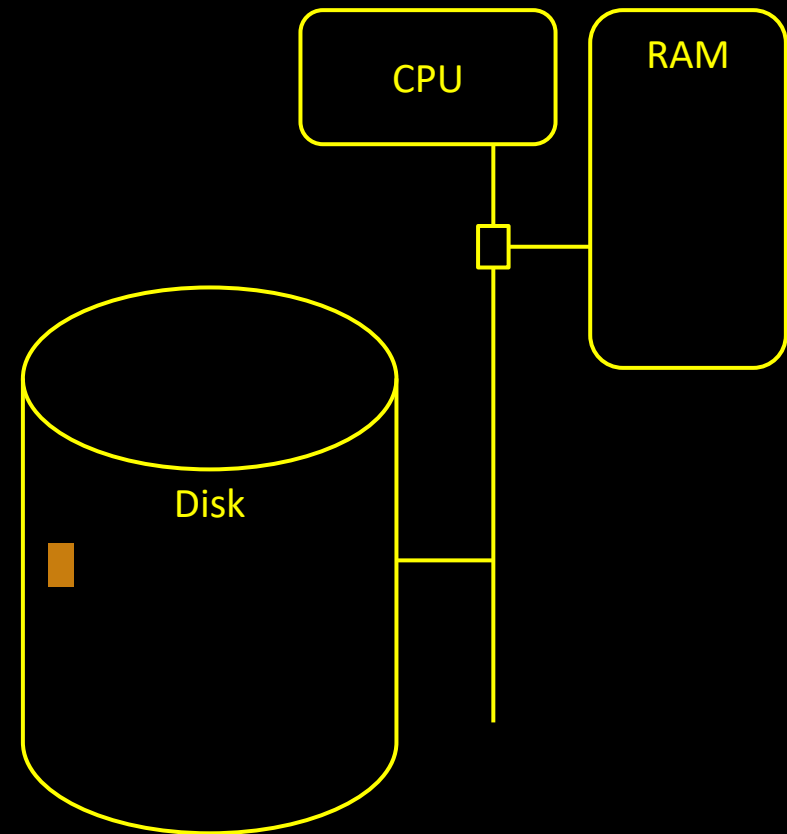
- When the RAM is full, we could satisfy upcoming allocations on disk
 - Is it possible for a CPU to access disk storage transparently?
 - Could we choose a smarter RAM/disk distribution?

```
kernel    get_free_page ()  
          {  
          ...  
          }
```



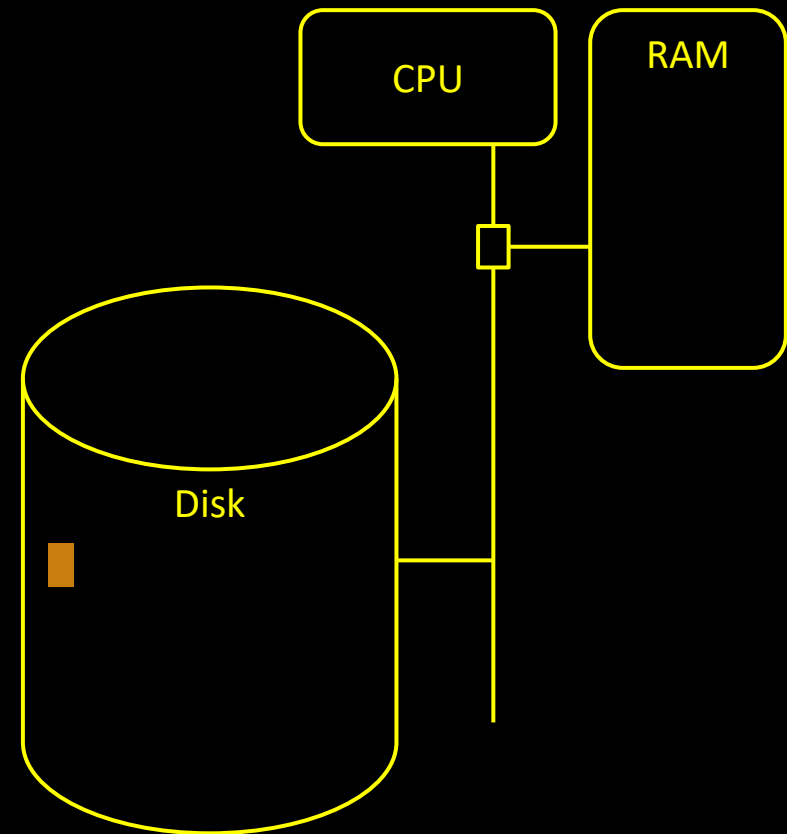
Technical considerations

- Is it possible for a CPU to access disk storage transparently?
 - Devices can be assigned a range of physical addresses on the I/O bus
 - With 39-bit physical addresses, we can not only address RAM, but also any other addressable device
 - A CPU read/write inside this range is redirected to the device
 - Typically used to access device configuration registers
 - This range can be mapped to a process address space
 - Eg. Dolphin “Scalable Coherent Interface” [1992]



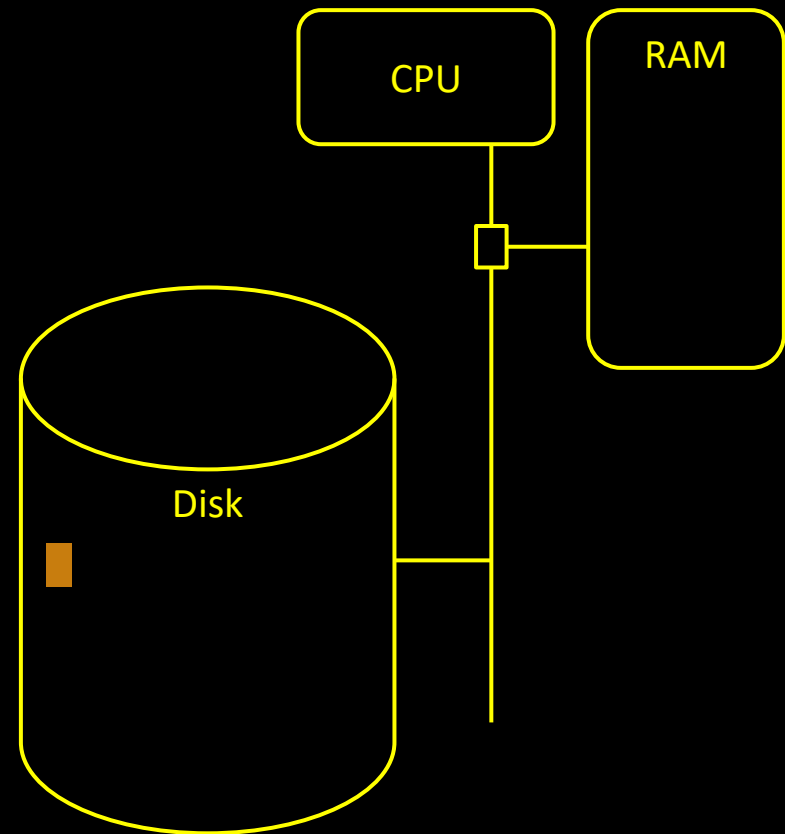
Technical considerations

- Is it possible for a CPU to access disk storage transparently?
 - Yes, in theory, this would be possible... however
 - This would involve sending PCI requests for each individual access
 - Coalescing is possible, but only for small bursts
 - Would lead to poor performance!
 - Disks (either hard drives or SSD) perform block-based transfers
 - E.g. 512B or 1K



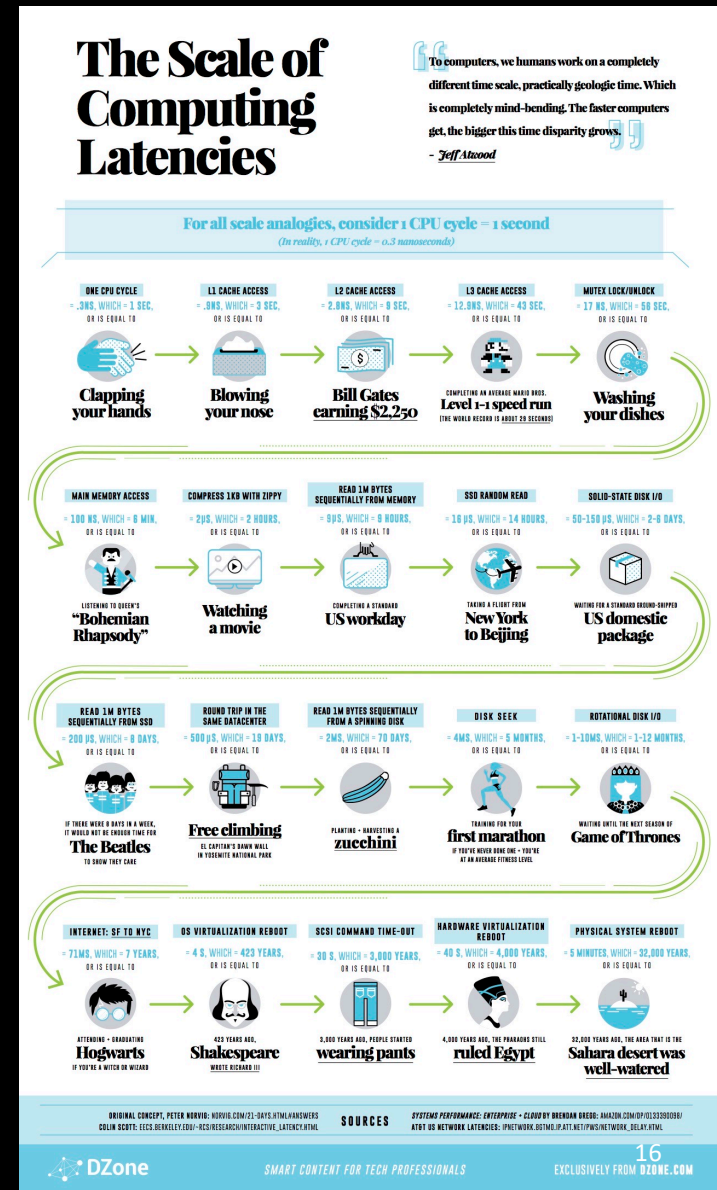
Technical considerations

- Is it possible for a CPU to access disk storage transparently?
 - Even if it was possible
 - Latency would kill us!
 - Accessing a word on a disk is *significantly* slower than accessing a variable in RAM
 - How much slower?



Scale of Computer Latencies

- CPU cycle: 0.3 ns (Core i7 3 GHz)
- L1 cache: 1 ns
- L2 cache: 5 ns
- L3 cache: 20 ns
- RAM: 50 ns
- NVMe SSD: 20 μ s
- SATA SSD: 150 μ s
- Hard Drive: 3 ms



Vous ne pouvez plus voter



En parlant de disque dur, quel est donc cet objet intrigant ?



1 Il s'agit du premier pied à coulisse de l'histoire, exposé au Musée du pied à coulisse de Dax 11% 7

2 C'est un outil utilisé par le SAV Apple pour estimer la durée de vie d'une batterie qui va bientôt exploser sera prise en charge sous garantie 3% 2



3 C'est un outil permettant de vérifier rapidement si un disque dur est au format 2.5 ou 3.5 pouces, et les bricoleurs savent à quel point c'est précieux 21% 13



Cliquez sur l'écran projeté pour lancer la question

4 C'est un gabarit permettant de vérifier la distance entre les plateaux d'un disque dur 49% 31

5 Il s'agit d'un instrument pour se faire (très) mal au nez 16% 10 ✓

wooclap



10%



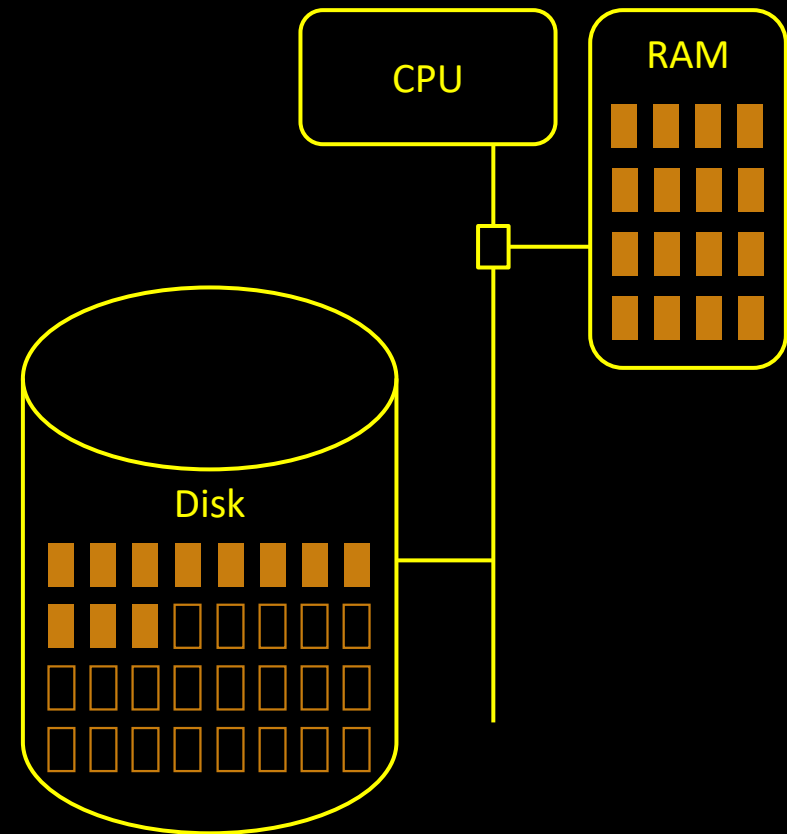
16% correct

63 / 923



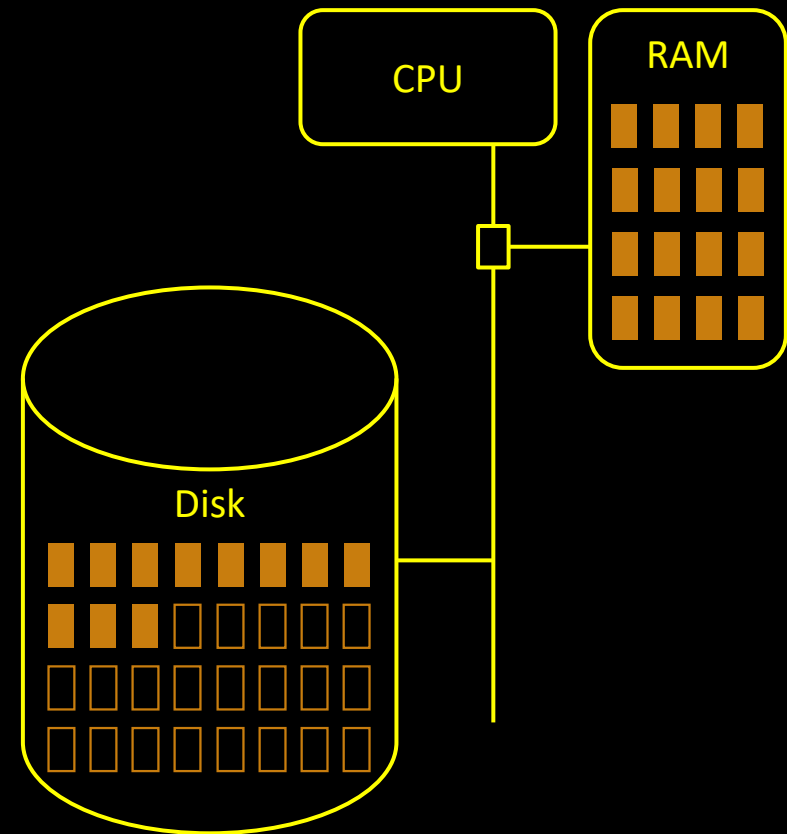
On-disk paging

- As a result, pages stored on disk will become...
“temporarily unreachable”
 - What does this mean?
 - *Swapped-out* pages must be marked “invalid” in their owner’s page table



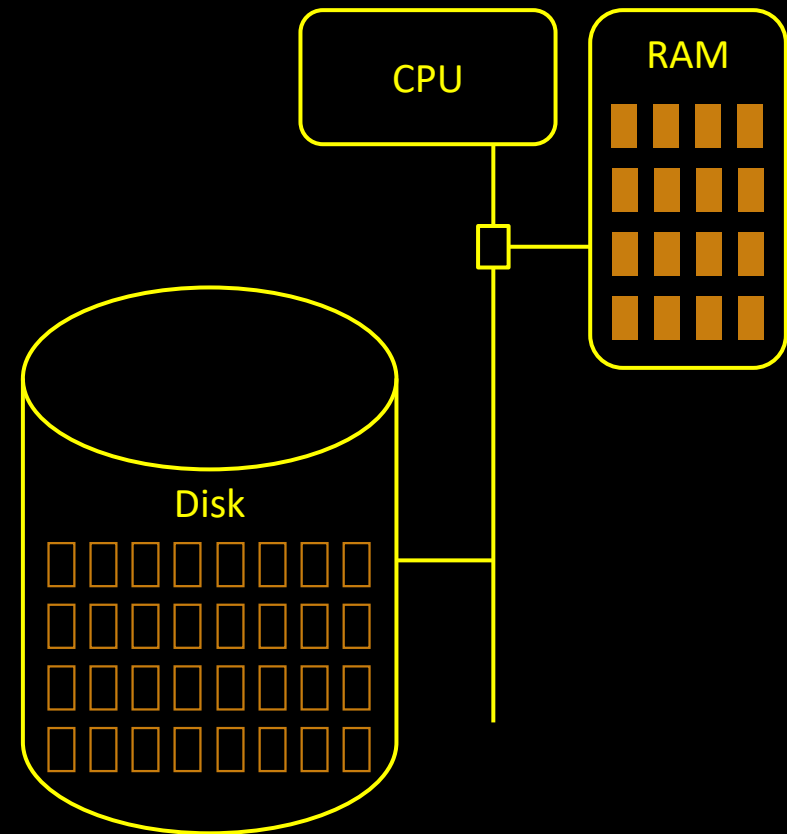
On-disk paging

- As a result, pages stored on disk will become...
“temporarily unreachable”
 - What does this mean?
 - *Swapped-out* pages must be marked “invalid” in their owner’s page table
 - A page fault corresponding to a swapped-out page must bring the page back to memory (swap-in)
 - What if the RAM is still full?
 - What if it happens too frequently?



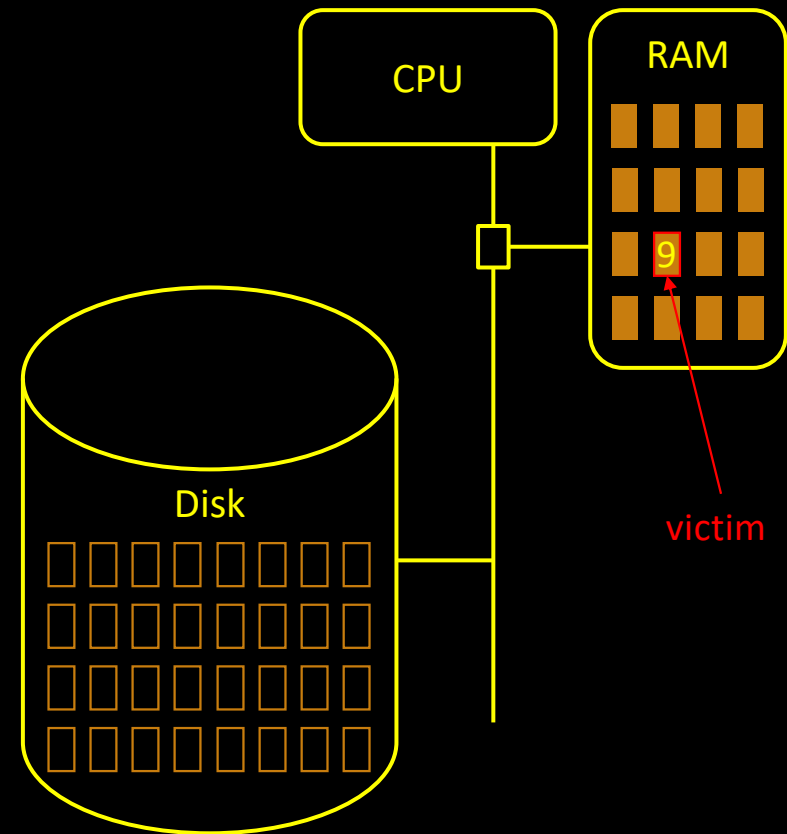
On-disk paging

- Let's calm down and rewind...
 - When the RAM is full and `get_free_page()` is called
 - The new page is expected to be used... immediately
 - So it must be allocated in RAM



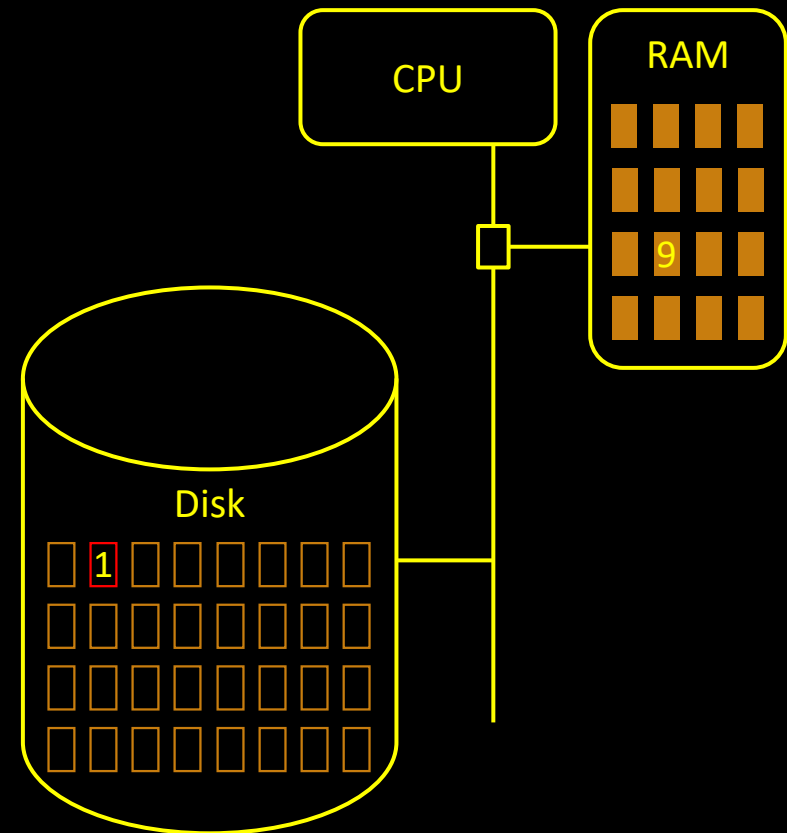
On-disk paging

- Let's calm down and rewind...
 - When the RAM is full and `get_free_page()` is called
 - The new page is expected to be used... immediately
 - So it must be allocated in RAM
 - Another (victim) page should be swapped out



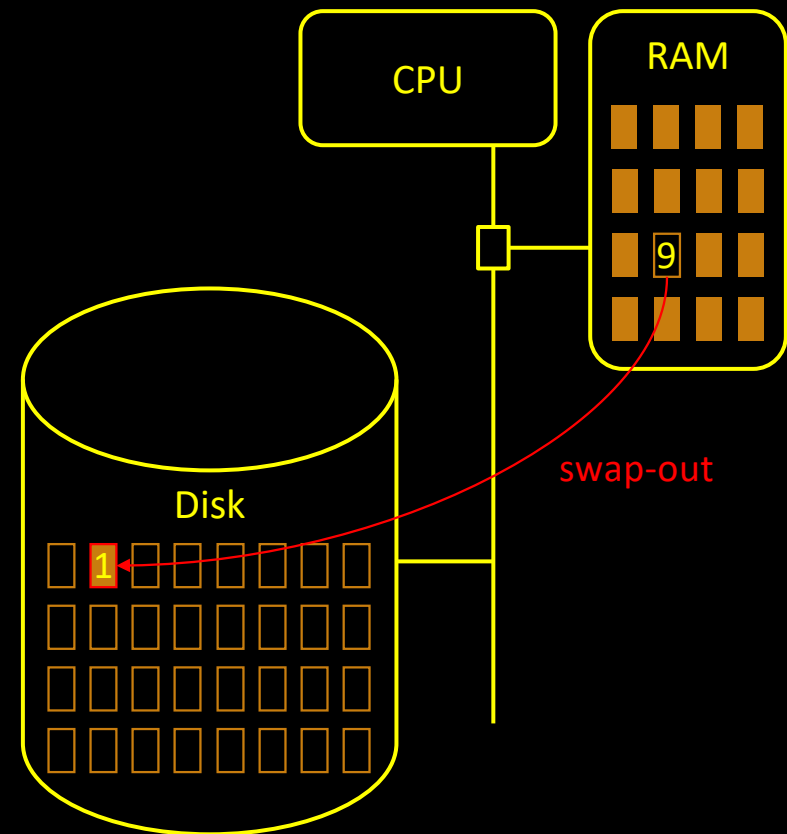
On-disk paging

- Let's calm down and rewind...
 - When the RAM is full and `get_free_page()` is called
 - The new page is expected to be used... immediately
 - So it must be allocated in RAM
 - Another (victim) page should be swapped out
 - A disk slot is allocated



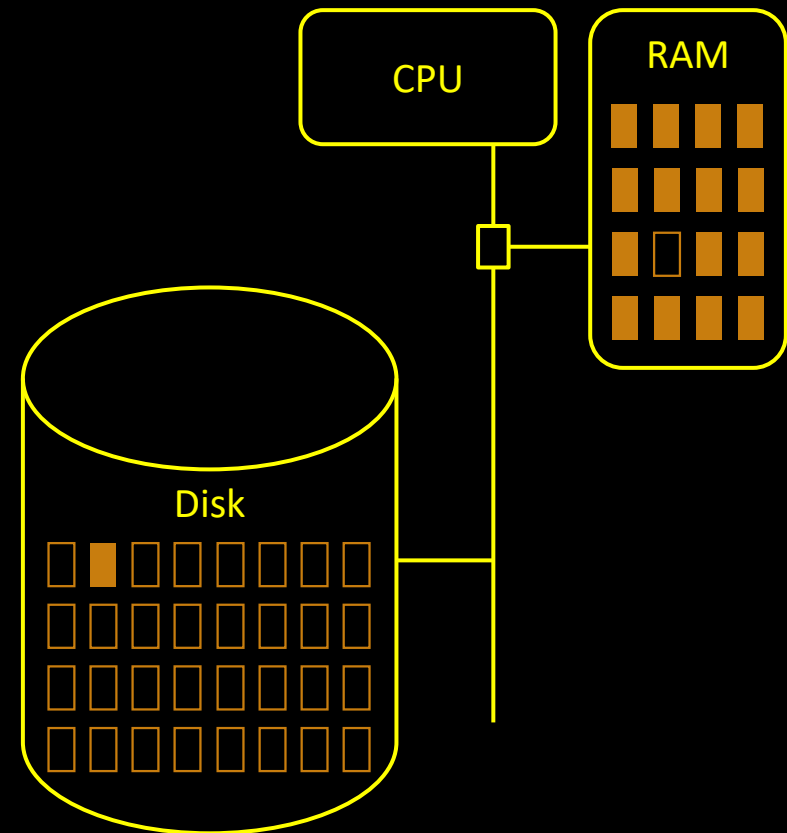
On-disk paging

- Let's calm down and rewind...
 - When the RAM is full and `get_free_page()` is called
 - The new page is expected to be used... immediately
 - So it should be allocated in RAM
 - Another (victim) page should be swapped out
 - A disk slot is allocated
 - Page is written to disk
 - The *page table entry* is marked invalid



On-disk paging

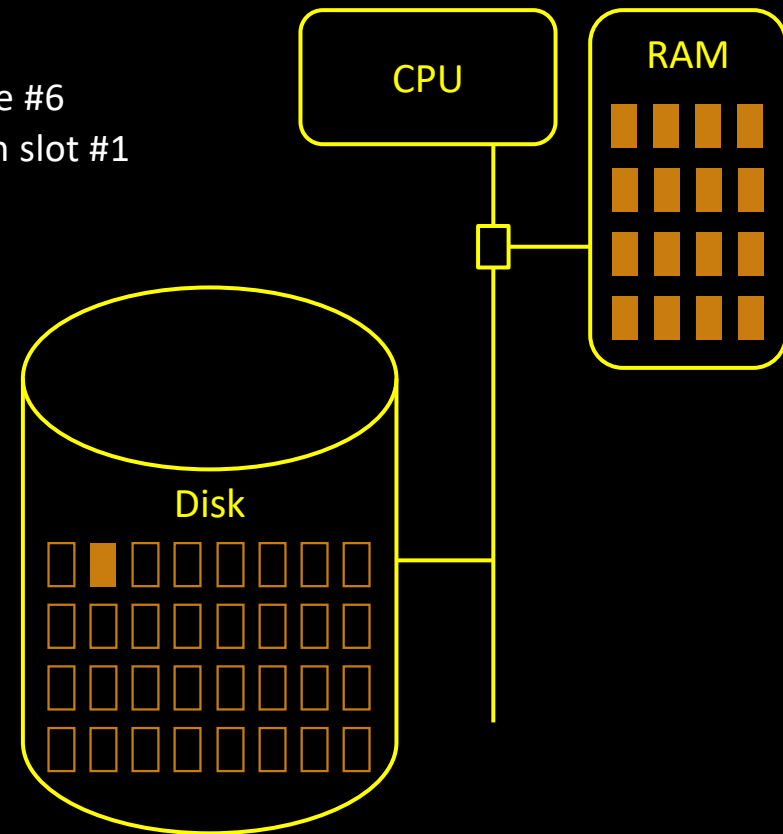
- Let's calm down and rewind...
 - When the RAM is full and `get_free_page()` is called
 - The new page is expected to be used... immediately
 - So it should be allocated in RAM
 - Another (victim) page should be swapped out
 - A disk slot is allocated
 - Page is written to disk
 - The *page table entry* is marked invalid



Handling swapped-out pages

	Phys. Page	Valid	R	W	X
0	2	1	1	0	1
1	4	1	1	0	1
2	5	1	1	0	1
3		0			
4	0	1	1	1	0
5		0			
6	9	0			
7		0			
8		0			
9		0			
10		0			
11	11	1	1	1	0
12		0			
13		0			
14		0			

Let's assume virtual page #6
is swapped out and stored in slot #1

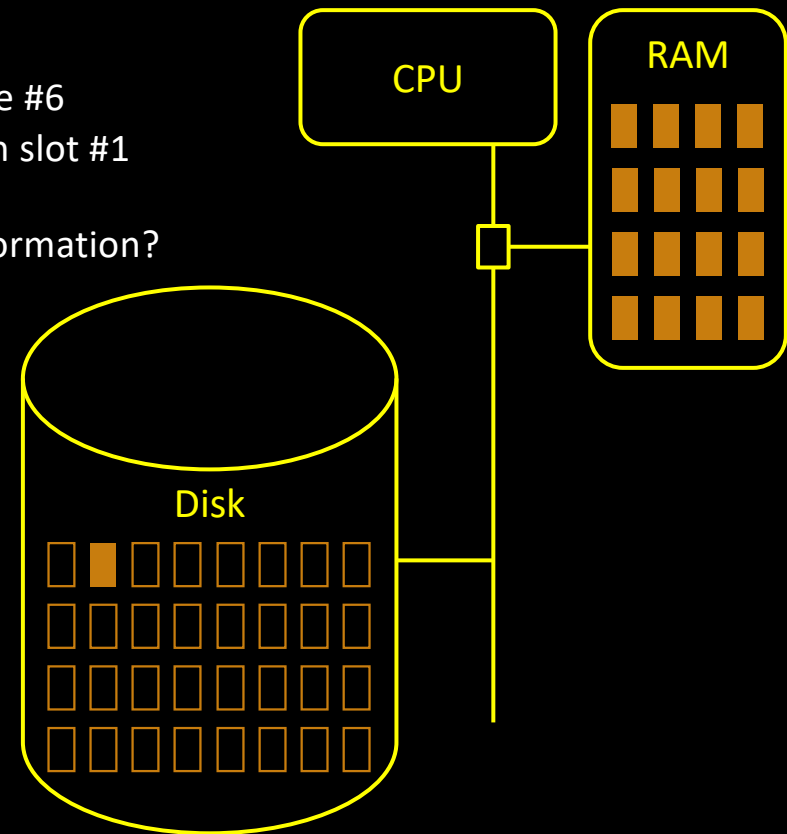


Handling swapped-out pages

	Phys. Page	Valid	R	W	X
0	2	1	1	0	1
1	4	1	1	0	1
2	5	1	1	0	1
3		0			
4	0	1	1	1	0
5		0			
6	???	0			
7		0			
8		0			
9		0			
10		0			
11	11	1	1	1	0
12		0			
13		0			
14		0			

Let's assume virtual page #6
is swapped out and stored in slot #1

How do we memorize this information?



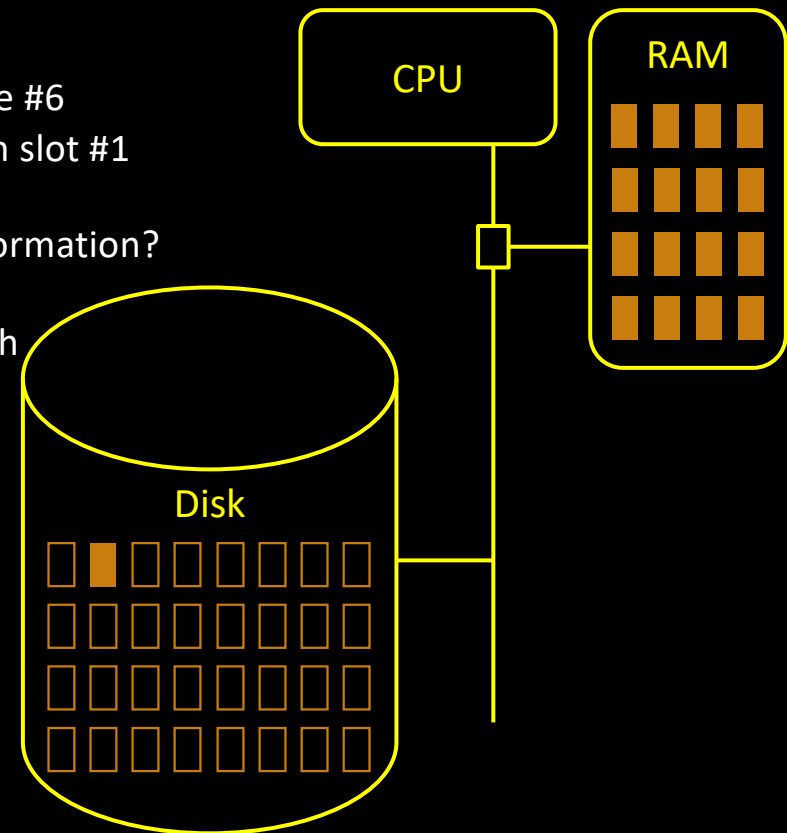
Handling swapped-out pages

	Phys. Page	Valid	R	W	X
0	2	1	1	0	1
1	4	1	1	0	1
2	5	1	1	0	1
3		0			
4	0	1	1	1	0
5		0			
6	???	0			
7		0			
8		0			
9		0			
10		0			
11	11	1	1	1	0
12		0			
13		0			
14		0			

Let's assume virtual page #6
is swapped out and stored in slot #1

How do we memorize this information?

How do we distinguish
swapped-out pages
and
lazily allocated
pages?

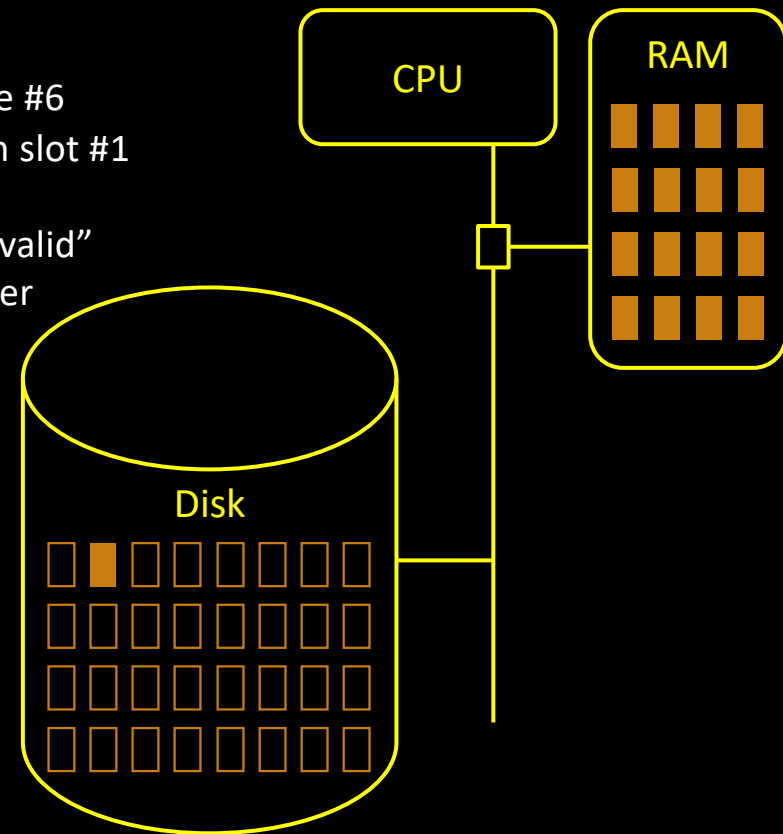


Handling swapped-out pages

	Phys. Page	Valid	R	W	X
0	2	1	1	0	1
1	4	1	1	0	1
2	5	1	1	0	1
3		0			
4	0	1	1	1	0
5		0			
6	1	0			
7		0			
8		0			
9		0			
10		0			
11	11	1	1	1	0
12		0			
13		0			
14		0			

Let's assume virtual page #6
is swapped out and stored in slot #1

We can use all fields but "valid"
to store the slot number



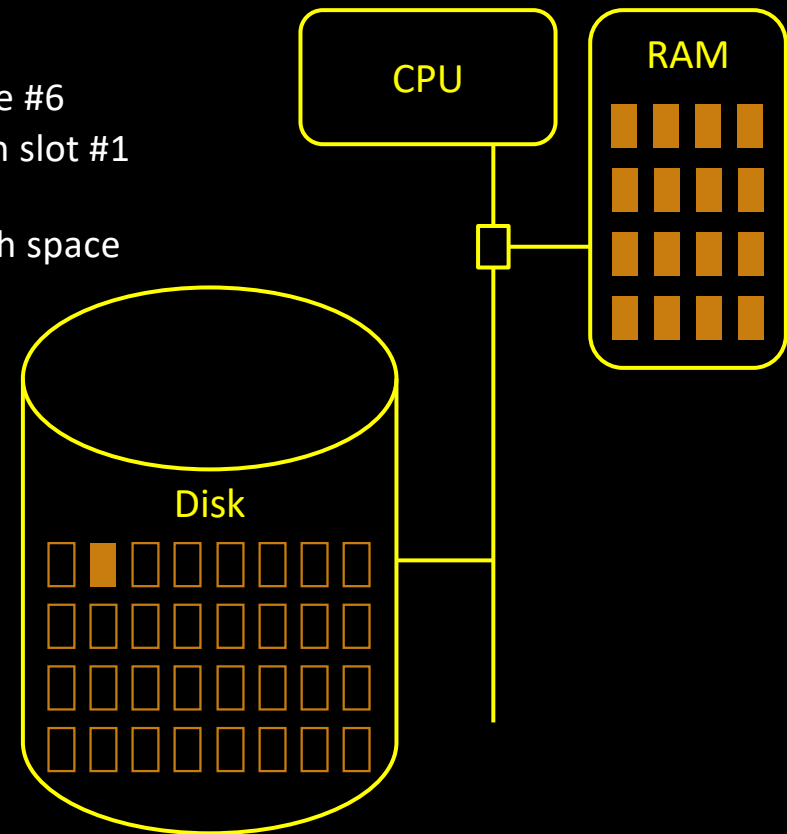
Handling swapped-out pages

	Phys. Page	Valid	R	W	X
0	2	1	1	0	1
1	4	1	1	0	1
2	5	1	1	0	1
3		0			
4	0	1	1	1	0
5		0			
6	index	0			
7		0			
8		0			
9		0			
10		0			
11	11	1	1	1	0
12		0			
13		0			
14		0			

Let's assume virtual page #6
is swapped out and stored in slot #1

Actually, there is not enough space
to store
<dev, block>

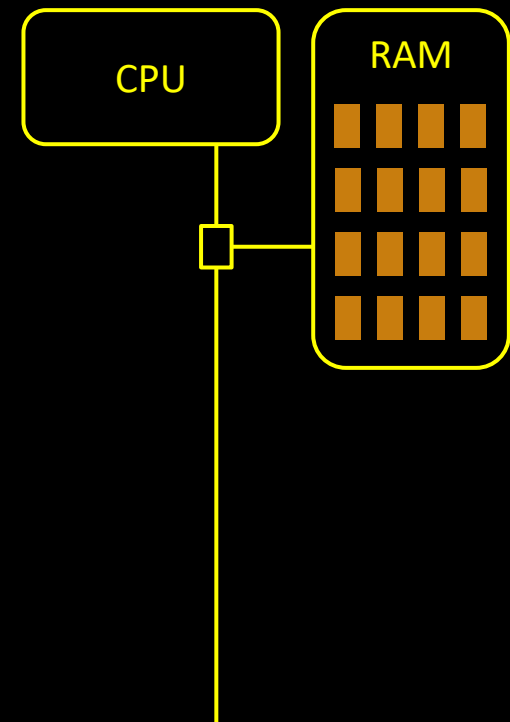
So an index in a
"swap_info" array
is stored



Finding a good victim



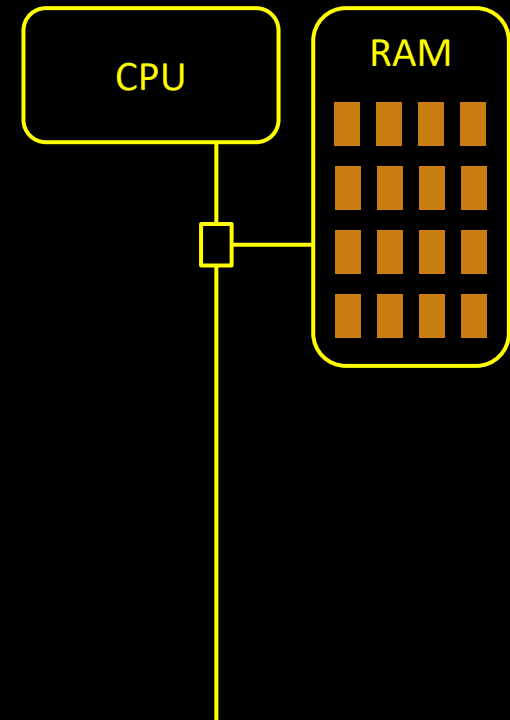
- We should choose a page
 - Which has a low probability to be requested again soon
 - Because we don't have fun swapping pages in & out



Finding a good victim

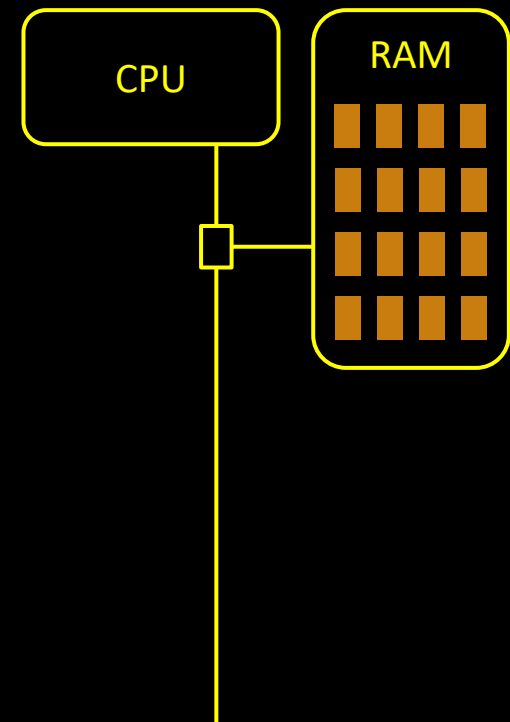


- We should choose a page
 - Which has a low probability to be requested again soon
 - Because we don't have fun swapping pages in & out
 - More precisely
 - The optimal algorithm should select the page whose next use will occur farthest in the future
 - Sounds good!
 - But in practice, we have no idea...



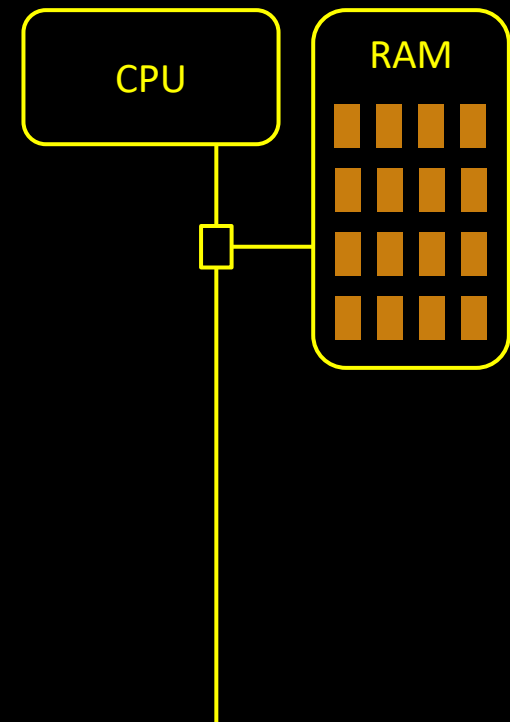
Page replacement algorithms

- Simple algorithms are often the best ones
 - How about using FIFO?
 - The OS keeps the date of "arrival" of pages in RAM



Page replacement algorithms

- Simple algorithms are often the best ones
 - How about using FIFO?
 - The OS keeps the date of "arrival" of pages in RAM
 - Unfortunately, the FIFO replacement strategy has a severe flaw
 - FIFO does not belong to the class of Stack-based algorithms
 - *"The set of pages in memory for N frames is always a subset of the set of pages that would be in memory with $N + 1$ frames"*



Belady's anomaly with FIFO

- Belady, *An anomaly in space-time characteristics of certain programs running in a paging machine*, Communications of the ACM, 1969.
- Let us consider the following series of page reclaims
 - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- It can be shown that this reference list causes more page faults with a 4-frame RAM than with a 3-frame one...
 - The more RAM you have, the worser it behaves ☹

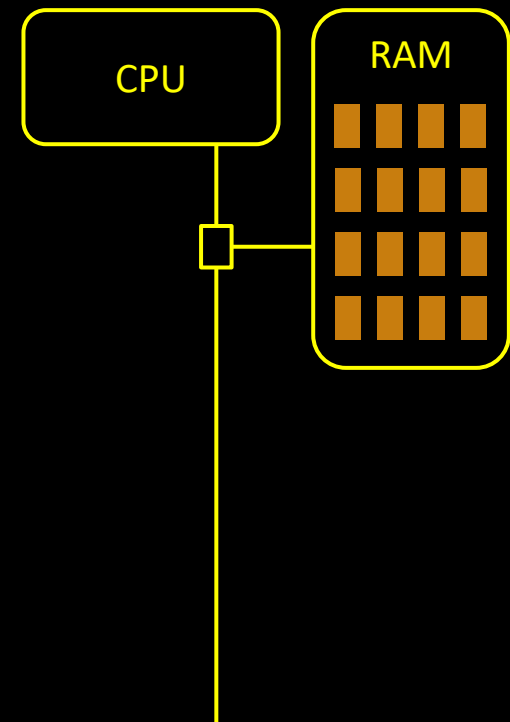
Belady's anomaly with FIFO

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3

Page replacement algorithms

- So we'd better stick to stack-based algorithms such as
 - LFU (Least Frequently Used)
 - LRU (Least Recently Used)
- How to count the number of accesses to each page?
 - Do we really want to use NFU?
- How to store the last access time of each page?




Page replacement algorithms

- Only one hardware component can do it: the MMU
- When a virtual page is accessed, the MMU sets the corresponding accessed bit in the page table
 - It is up to the kernel to periodically test and reset this bit...

	Phys. Page	Valid	R	W	X	Accessed
0	2	1				0
1	4	1				0
2	5	1				0
3		0				
4	0	1				0
5		0				
6		0				
7		0				
8		0				
9		0				
10		0				
11	10	1				0
12		0				
13		0				
14		0				

Page replacement algorithms

- Only one hardware component can do it: the MMU
- When a virtual page is accessed, the MMU sets the corresponding accessed bit in the page table
 - It is up to the kernel to periodically test and reset this bit...



	Phys. Page	Valid	R	W	X	Accessed
0	2	1				0
1	4	1				0
2	5	1				1
3		0				
4	0	1				0
5		0				
6		0				
7		0				
8		0				
9		0				
10		0				
11	10	1				0
12		0				
13		0				
14		0				

Page replacement algorithms

- Only one hardware component can do it: the MMU
- When a virtual page is accessed, the MMU sets the corresponding accessed bit in the page table
 - It is up to the kernel to periodically test and reset this bit...

	Phys. Page	Valid	R	W	X	Accessed
0	2	1				0
1	4	1				0
2	5	1				1
3		0				
4	0	1				0
5		0				
6		0				
7		0				
8		0				
9		0				
10		0				
→ 11	10	1				1
12		0				
13		0				
14		0				

Page replacement algorithms

- Periodically, the kernel can walk through the page table and read (& reset) the accessed bit for each page
 - Sampling
 - Information = “was this page accessed during the last period”?

	Phys. Page	Valid	R	W	X	Accessed
0	2	1				0
1	4	1				0
2	5	1				1
3		0				
4	0	1				0
5		0				
6		0				
7		0				
8		0				
9		0				
10		0				
11	10	1				1
12		0				
13		0				
14		0				

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[xyz]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[xyz]
4		[xyz]
3		[xyz]
2		[xyz]
1		[xyz]
0		[xyz]

Page replacement algorithms

- Periodically, the kernel can walk through the page table and read (& reset) the accessed bit for each page

- Sampling

- Information = “was this page accessed during the last period”?

- Aging

- Kernel can maintain an approximation of last access time per physical page

	Phys. Page	Valid	R	W	X	Accessed		Physical addresses	Approx. last access time
0	2	1				0	15		[xyz]
1	4	1				0	14		[xyz]
2	5	1				1	13		[xyz]
3		0					12		[xyz]
4	0	1				0	11		[xyz]
5		0					10		[1xy]
6		0					9		[xyz]
7		0					8		[xyz]
8		0					7		[xyz]
9		0					6		[xyz]
10		0					5		[1xy]
11	10	1				1	4		[0xy]
12		0					3		[xyz]
13		0					2		[0xy]
14		0					1		[xyz]
							0		[0xy]

Page replacement algorithms

- Periodically, the kernel can walk through the page table and read (& reset) the accessed bit for each page

- Sampling

- Information = “was this page accessed during the last period”?

- Aging

- Kernel can maintain an approximation of last access time per physical page

	Phys. Page	Valid	R	W	X	Accessed
0	2	1				0
1	4	1				0
2	5	1				0
3		0				
4	0	1				0
5		0				
6		0				
7		0				
8		0				
9		0				
10		0				
11	10	1				0
12		0				
13		0				
14		0				

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[1xy]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[1xy]
4		[0xy]
3		[xyz]
2		[0xy]
1		[xyz]
0		[0xy]

Page replacement algorithms

- Periodically, the kernel can walk through the page table and read (& reset) the accessed bit for each page

- Sampling

- Information = “was this page accessed during the last period”?

- Aging

- Kernel can maintain an approximation of last access time per physical page

	Phys. Page	Valid	R	W	X	Accessed
0	2	1				0
1	4	1				0
2	5	1				0
3		0				
4	0	1				1
5		0				
6		0				
7		0				
8		0				
9		0				
10		0				
11	10	1				1
12		0				
13		0				
14		0				

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[1xy]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[1xy]
4		[0xy]
3		[xyz]
2		[0xy]
1		[xyz]
0		[0xy]

Page replacement algorithms

- Periodically, the kernel can walk through the page table and read (& reset) the accessed bit for each page

- Sampling

- Information = “was this page accessed during the last period”?

- Aging

- Kernel can maintain an approximation of last access time per physical page

	Phys. Page	Valid	R	W	X	Accessed		Physical addresses	Approx. last access time
0	2	1				0	15		[xyz]
1	4	1				0	14		[xyz]
2	5	1				0	13		[xyz]
3		0					12		[xyz]
4	0	1				1	11		[xyz]
5		0					10		[11x]
6		0					9		[xyz]
7		0					8		[xyz]
8		0					7		[xyz]
9		0					6		[xyz]
10		0					5		[01x]
11	10	1				1	4		[00x]
12		0					3		[xyz]
13		0					2		[00x]
14		0					1		[xyz]
							0		[10x]

Allez sur **wooclap.com** et utilisez le code **SEFOREVER**



CoWs



1

Le CoW permet au noyau de sauvegarder une copie d'une page juste avant une écriture, pour éventuellement restaurer la page si l'écriture n'était pas autorisée

35%

23

2

Un CoW résulte souvent du fait qu'un processus farfelu n'appelle pas exec après fork

29%

19



3

Le CoW est un mécanisme grâce auquel le CPU peut annuler les effets d'une exécution spéculative en restaurant les valeurs qu'ils a sauvées

20%

13

4

Vous savez que c'est une question piège n'est-ce pas?
Alors foutu pour foutu pourquoi ne pas cliquer ici pour faire genre "j'ai répondu" ?

17%

11

wooclap



100 %



66 / 923



Page replacement algorithms

- Now we have an “approximate last access time” for each physical pages
 - Can be used to implement a pseudo-LRU policy
 - 11x is more “important” than 10x, which is more important than 01x, etc.
 - E.g. Physical pages 2 and 4 (stamped “00x”) are a good candidates for being evicted from RAM

	Phys. Page	Valid	R	W	X	Accessed
0	2	1				0
1	4	1				0
2	5	1				0
3		0				
4	0	1				0
5		0				
6		0				
7		0				
8		0				
9		0				
10		0				
11	10	1				0
12		0				
13		0				
14		0				

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[11x]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[01x]
4		[00x]
3		[xyz]
2		[00x]
1		[xyz]
0		[10x]

Page replacement algorithms

- Should our algorithm be local or global?

- Local
 - Search victim among pages from current process
- Global
 - Inspect all pages before deciding

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[11x]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[01x]
4		[00x]
3		[xyz]
2		[00x]
1		[xyz]
0		[10x]

Page replacement algorithms

- Should our algorithm be local or global?
 - Local
 - Search victim among pages from current process
 - Would tend to keep the number of resident pages constant

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[11x]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[01x]
4		[00x]
3		[xyz]
2		[00x]
1		[xyz]
0		[10x]

48

Page replacement algorithms

- Should our algorithm be local or global?
 - Global
 - Inspect all pages before deciding
 - Treat all processes equally?
 - Or consider evicting pages from the biggest?

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[11x]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[01x]
4		[00x]
3		[xyz]
2		[00x]
1		[xyz]
0		[10x]

Page replacement algorithms

- Should our algorithm be local or global?
 - Global
 - Inspect all pages before deciding
 - Treat all processes equally?
 - Or consider evicting pages from the biggest?
 - Take from the rich and give to the poor, uh?



Errol Flynn, *The Adventures of Robin Hood*, 1938.

Page replacement algorithms

- Not so obvious kids...
- Some process may have a lavish lifestyle
 - What we call “*Train de vie luxueux*” in France
- They have “just enough money” to be happy

Page replacement algorithms

- Not so obvious kids...
- Some process may have a lavish lifestyle
 - What we call “*Train de vie luxueux*” in France
- They have “just enough money” to be happy

Présidentielle: [redacted] affirme qu'il n'arrive pas à mettre d'argent de côté

Page replacement algorithms

- Not so obvious kids...
- Some process may have a lavish lifestyle
 - What we call “*Train de vie luxueux*” in France
- They have “just enough money” to be happy

Présidentielle: [redacted] affirme qu'il n'arrive pas à mettre d'argent de côté

Pour [redacted], les députés ont besoin de "9.000 euros net par mois, plus 3.000 euros de frais" -

Page replacement algorithms

- Not so obvious kids...
- Some process may have a lavish lifestyle
 - What we call “*Train de vie luxueux*” in France
- They have “just enough money” to be happy

Présidentielle: [redacted] affirme qu'il n'arrive pas à mettre d'argent de côté

Pour [redacted], les députés ont besoin de "9.000 euros net par mois, plus 3.000 euros de frais" -

[redacted]: "Un député obligé de justifier ses frais n'est plus libre"

Page replacement algorithms

- Not so obvious kids...
- Some process may have a lavish lifestyle
 - What we call “*Train de vie luxueux*” in France
- They have “just enough money” to be happy

Présidentielle: [redacted] affirme qu'il n'arrive pas à mettre d'argent de côté

Pour [redacted], les députés ont besoin de "9.000 euros net par mois, plus 3.000 euros de frais" -

[redacted]: "Un député obligé de justifier ses frais n'est plus libre"

Le député [redacted] justifie l'aide pour « ménages modestes » reçue pour rénover son logement

Page replacement algorithms

- Not so obvious kids...
- Some process may have a lavish lifestyle
 - What we call “*Train de vie luxueux*” in France
- They have “just enough money” to be happy

Présidentielle: [redacted] affirme qu'il n'arrive pas à mettre d'argent de côté

Pour [redacted], les députés ont besoin de "9.000 euros net par mois, plus 3.000 euros de frais" -

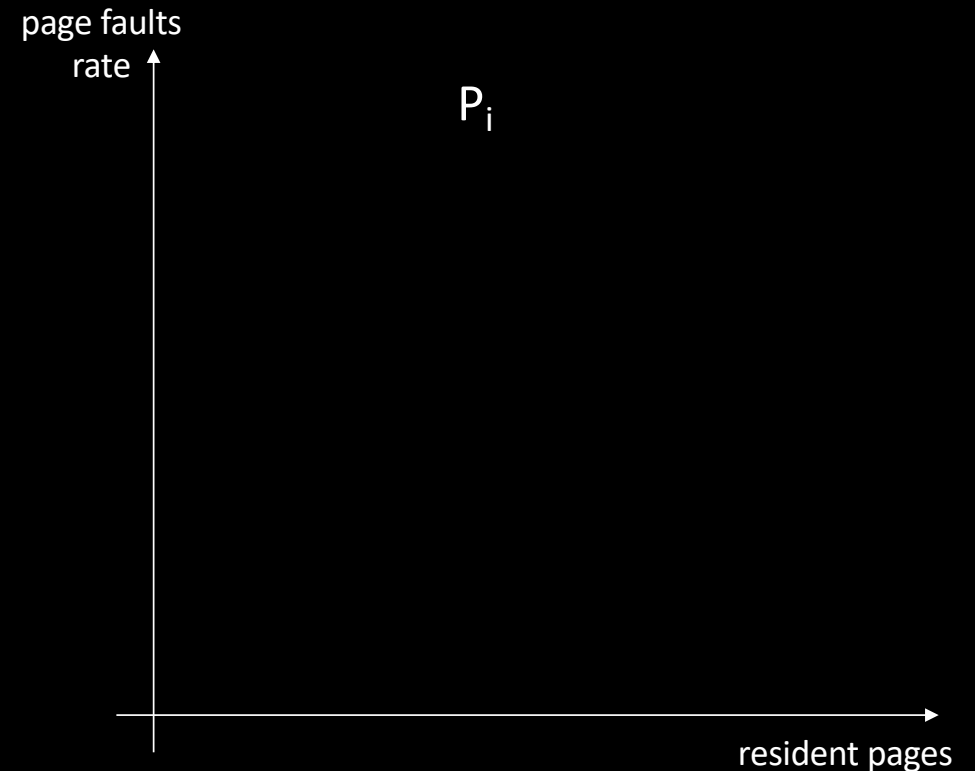
[redacted]: "Un député obligé de justifier ses frais n'est plus libre"

Le député [redacted] justifie l'aide pour « ménages modestes » reçue pour rénover son logement

[redacted] s'est fait offrir pour près de 200 000 € de costumes en cinq ans

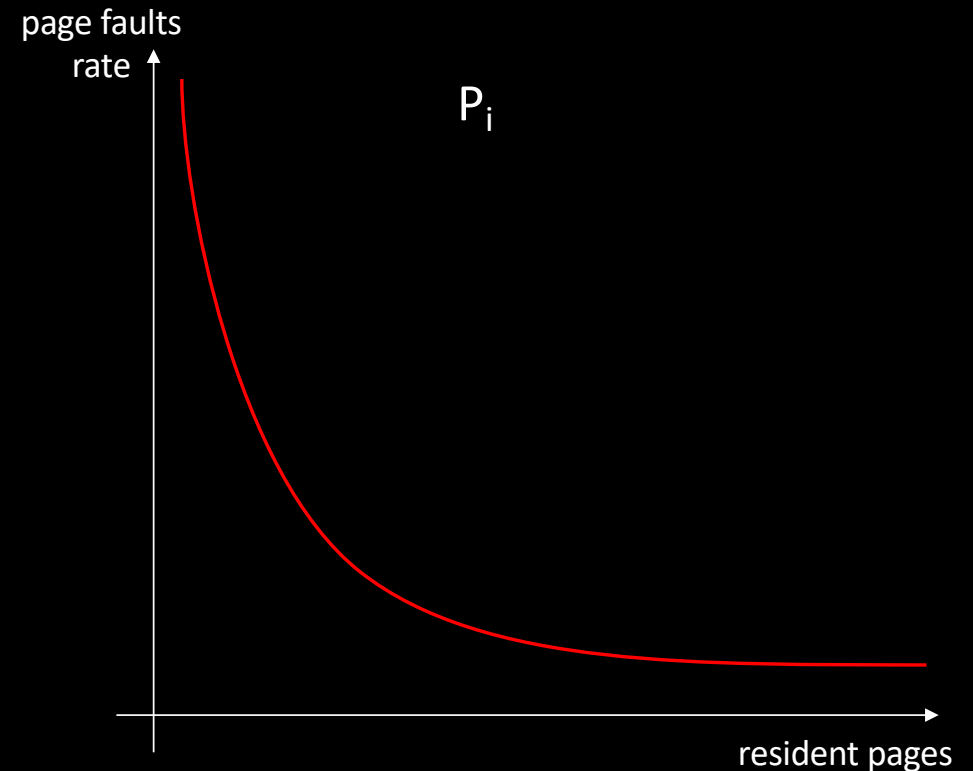
The notion of Working Set

- Can we determine the number of resident pages which will make a process happy?
 - The OS can track the number of page faults per second when
 - 1 resident page is granted
 - 2 resident pages are granted
 - 3 resident pages are granted
 - Etc.



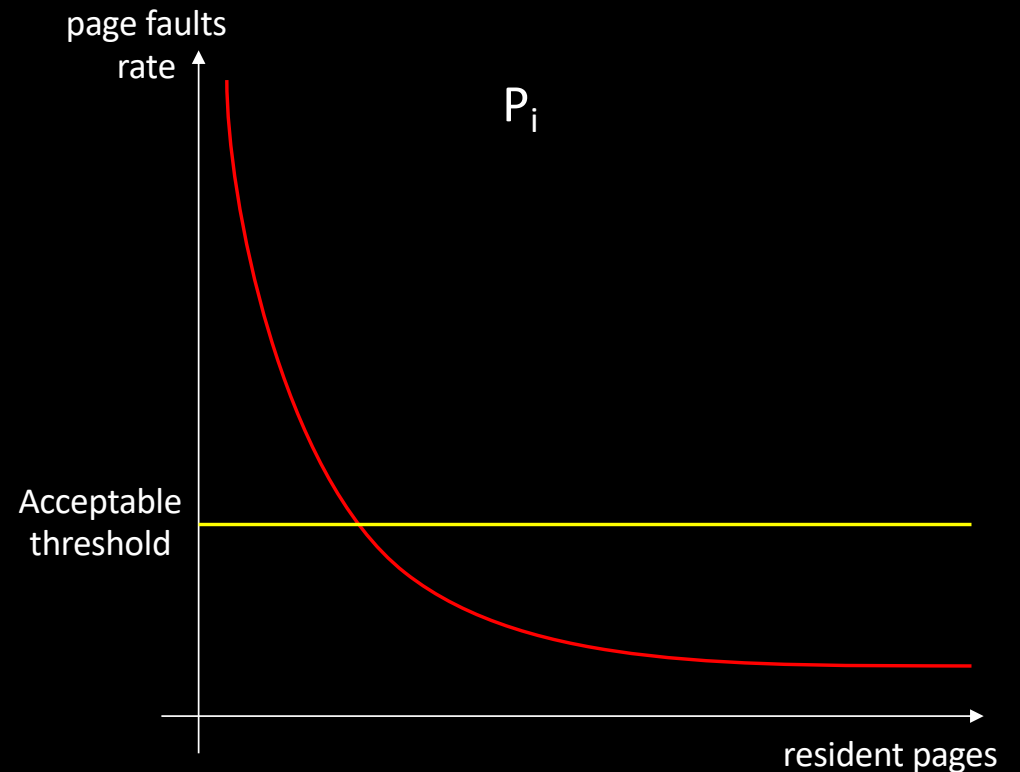
The notion of Working Set

- Can we determine the number of resident pages which will make a process happy?
 - The OS can track the number of page faults per second when
 - 1 resident page is granted
 - 2 resident pages are granted
 - 3 resident pages are granted
 - Etc.



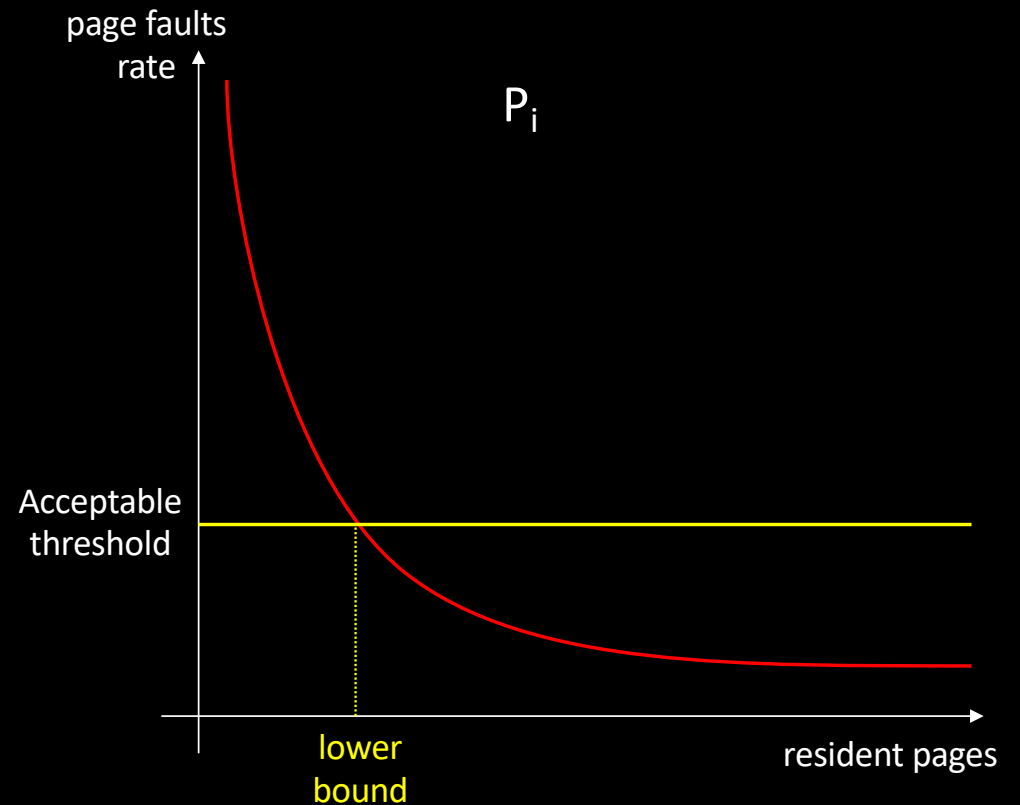
The notion of Working Set

- Can we determine the number of resident pages which will make a process happy?
 - Beyond an “acceptable threshold”, the page fault rate incurs a severe slowdown



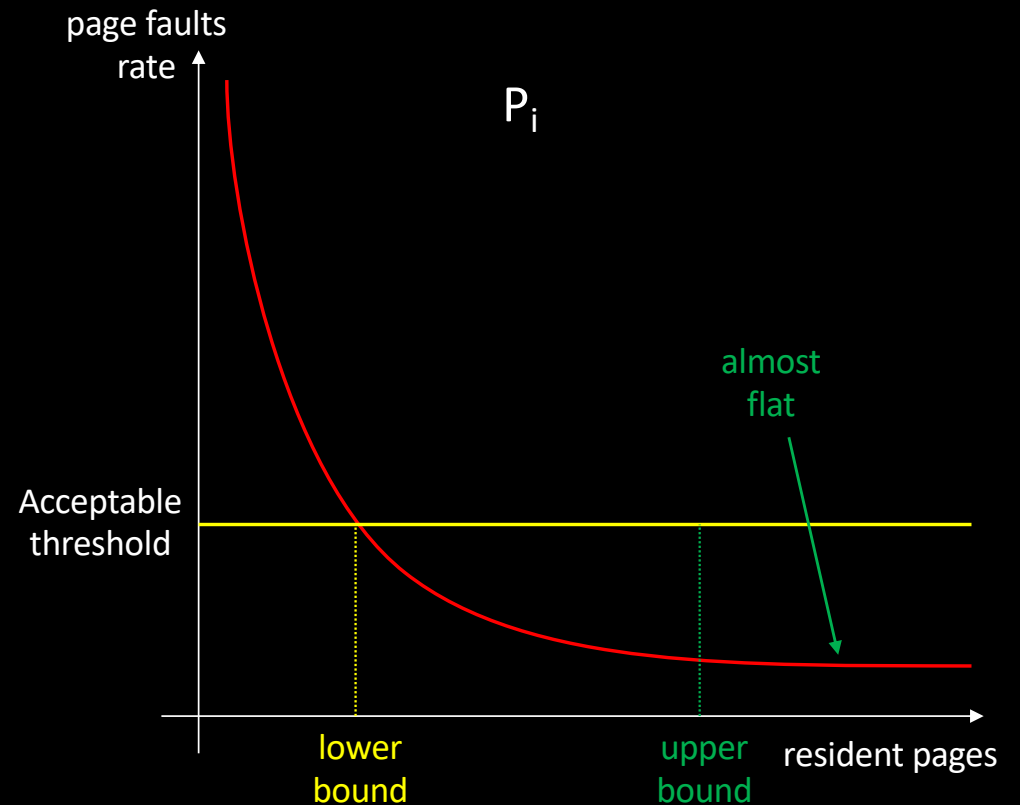
The notion of Working Set

- Can we determine the number of resident pages which will make a process happy?
 - Beyond an “acceptable threshold”, the page fault rate incurs a severe slowdown
 - This gives a lower bound on the number of pages which should stay in RAM for P_i



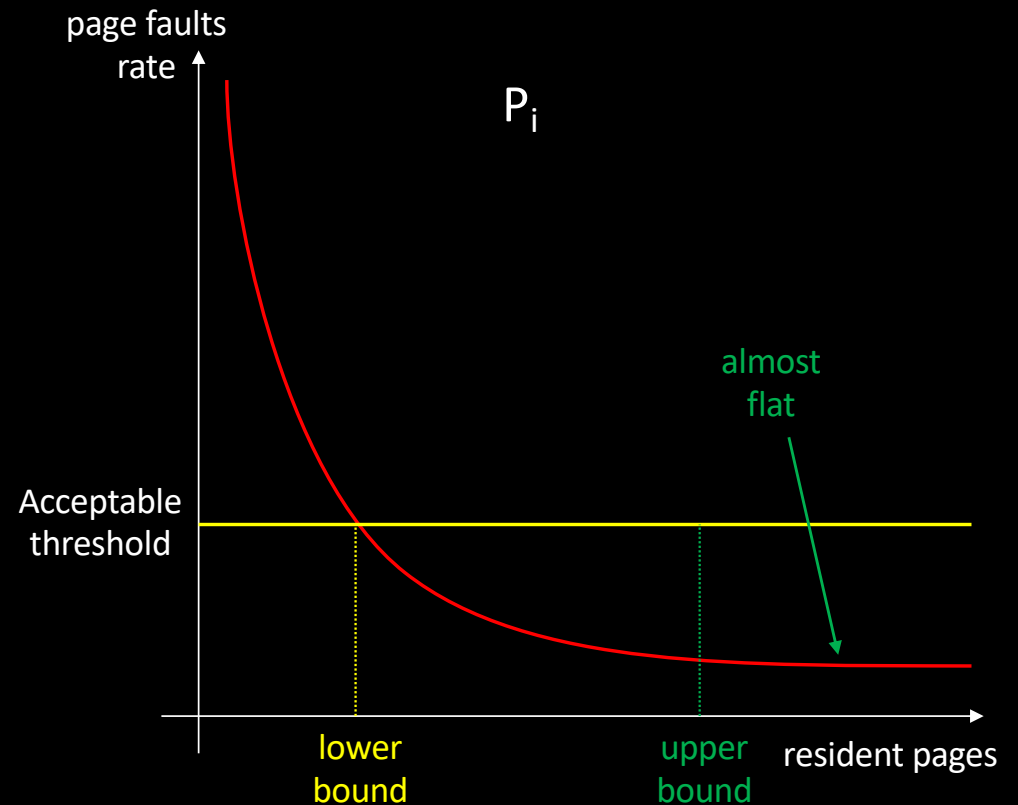
The notion of Working Set

- Can we determine the number of resident pages which will make a process happy?
 - Beyond an upper bound, the page fault rate doesn't decrease any more



The notion of Working Set

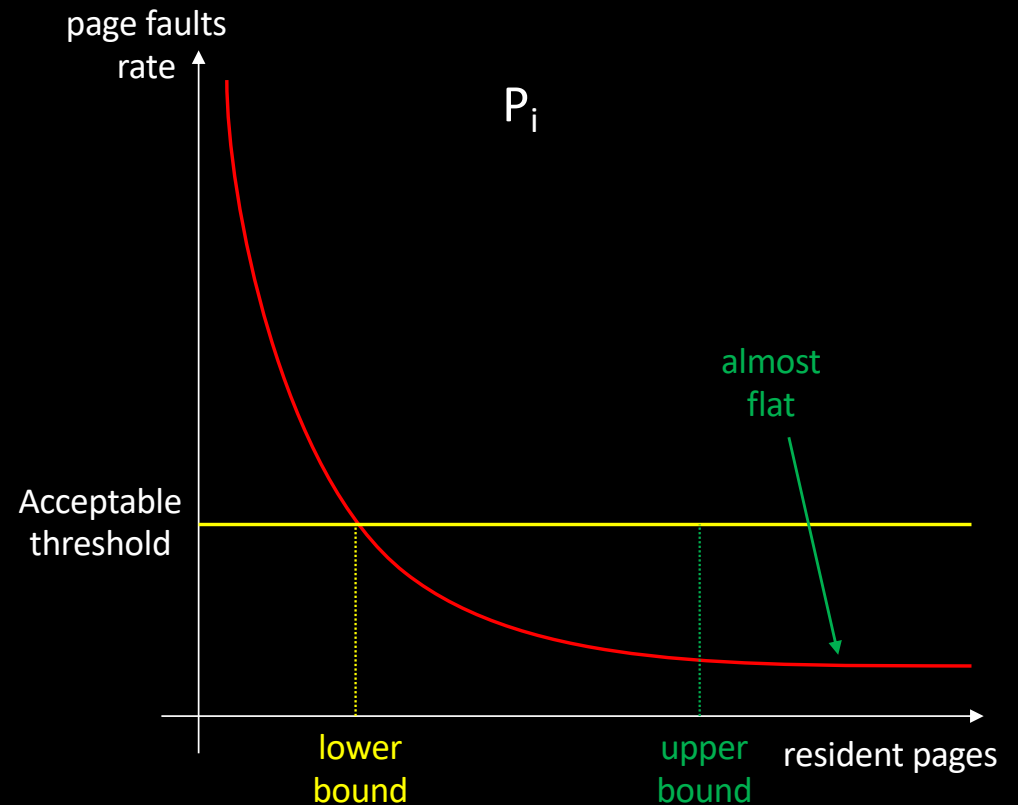
- Keeping the number of resident pages $\in [\text{lower}..\text{upper}]$ is ideal
- When searching for a victim
 - We should consider processes which live far beyond their lower bound
- In practice, modern OSes approximate the WS to
 - “pages which have been accessed during a predefined period”



The notion of Working Set

- Keeping the number of resident pages $\in [\text{lower}..\text{upper}]$ is ideal

- Note that it may be impossible
 - Too many ready processes
=> Trashing
- Avoiding trashing
 - Gang scheduling
 - Process swapping



Page replacement algorithms

- Recap

- When RAM is full
 - A victim page must be evicted to satisfy a new allocation
- The kernel is able to compute an approximation of the last access time for each physical page
 - A page which is not accessed for a long period (taking the virtual time of processes into account) does not belong to a working set
 - Good candidate
 - [see WSClock algorithm]

	Physical addresses	Approx. last access time
15		[xyz]
14		[xyz]
13		[xyz]
12		[xyz]
11		[xyz]
10		[11x]
9		[xyz]
8		[xyz]
7		[xyz]
6		[xyz]
5		[01x]
4		[00x]
3		[xyz]
2		[00x]
1		[xyz]
0		[10x]

Page replacement algorithms

- Side notes

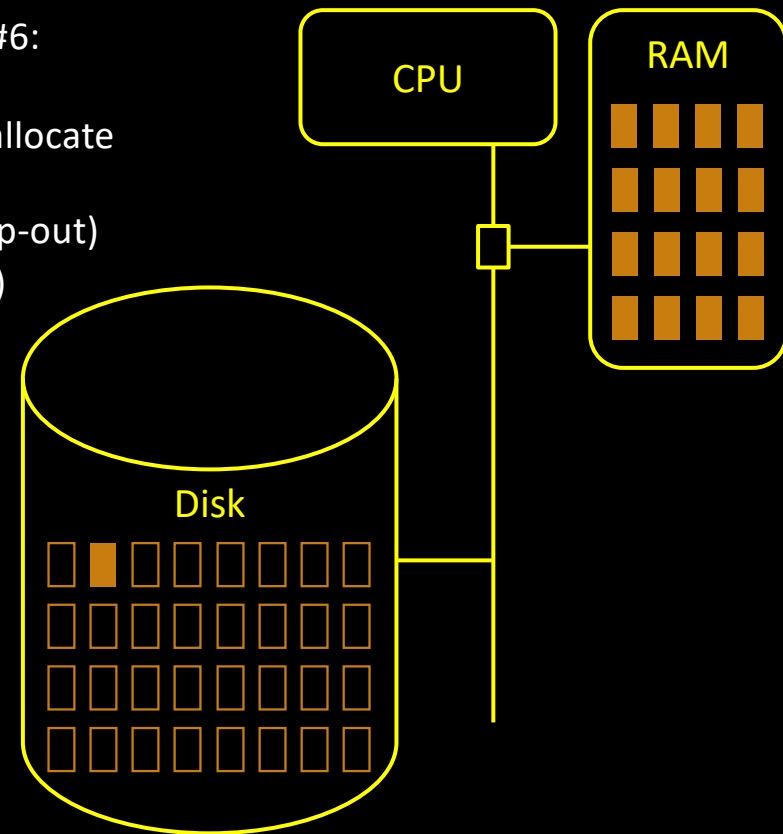
- How do we find the PTE (*Page Table Entry*) pointing on a physical page?
 - Reverse mapping stored in a structure associated to each physical page
- How to swap-out shared pages?
 - What if RAM mostly contains shared pages?

Bringing pages back to RAM

	Phys. Page	Valid	R	W	X
0	2	1	1	0	1
1	4	1	1	0	1
2	5	1	1	0	1
3		0			
4	0	1	1	1	0
5		0			
6	1	0			
7		0			
8		0			
9		0			
10		0			
11	11	1	1	1	0
12		0			
13		0			
14		0			

If process accesses virtual page #6:

1. Page fault raised
2. `get_free_page()` is called to allocate a new physical frame (note: this may trigger a swap-out)
3. Read page from disk (slot #1) and write to new frame
4. Fix page table entry to reference new frame

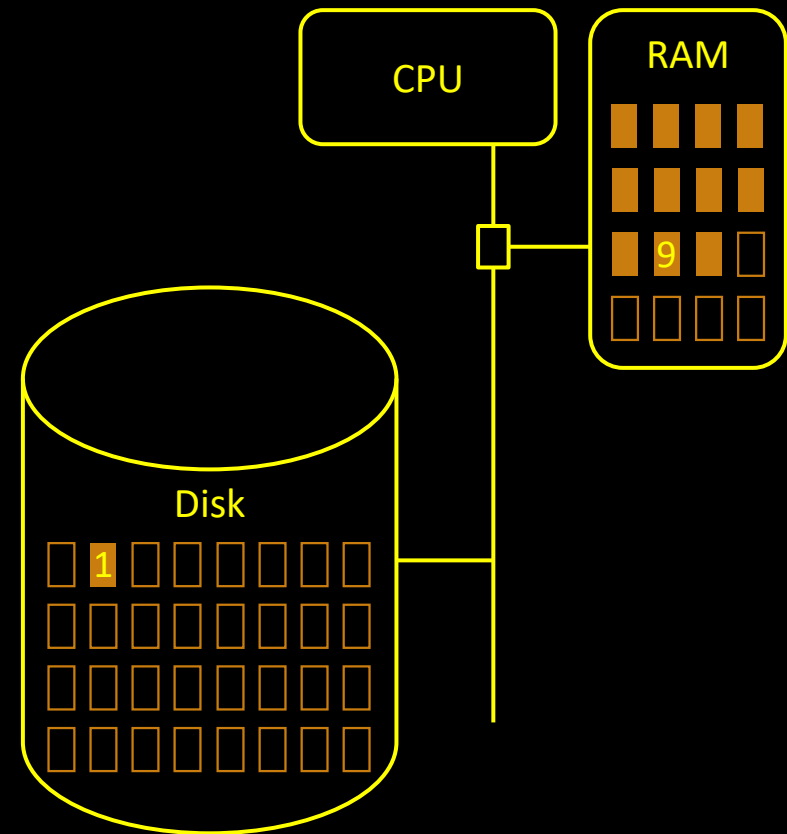


Bringing pages back to RAM

- In the worst case, two I/O operations are involved before the process can resume its execution
 - Too slow!
- Several optimizations can be used
 - Swap-outs can be anticipated
 - Some swap-outs can be avoided

Avoiding Swap-outs

- When a page has already been swapped-out, and is now back to memory
 - The swap slot can be kept as is
 - If the page is not modified until it must be evicted again
 - i.e., the page is “clean”
 - Then no disk write is need
 - If the page was modified in the meantime
 - i.e., the page is “dirty”
 - Then it must be swapped-out again



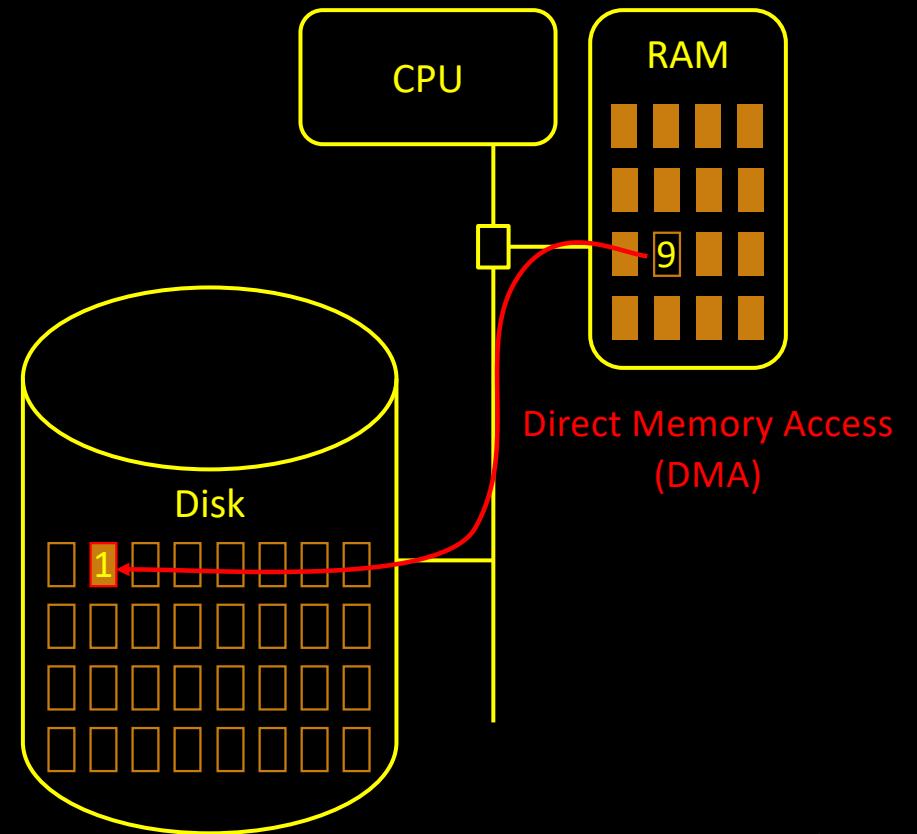
Avoiding Swap-outs

- Detecting if the page was modified could be done by removing the 'w' access mode
 - Too expensive
- Page table entries feature a dirty bit
 - Set by MMU on each write access
 - Cleared by OS when page is installed in RAM

	Phys. Page	Valid	R	W	X	Accessed	Dirty
0	2	1				0	0
1	4	1				1	0
2	5	1				1	1
3		0					
4	0	1				0	0
5		0					
6		0					
7		0					
8		0					
9		0					
10		0					
11	10	1				1	0
12		0					
13		0					
14		0					

Anticipating swap-outs

- A kernel DAEMON thread (kswapd) keeps maintaining a threshold of free pages in RAM
 - When the reserve of free pages is too small, the thread frees pages “in the background”
- Most of the time, `get_free_page()` finds a free frame immediately!



Additional resources
available on
<http://gforgeron.gitlab.io/se/>