

# assembly

JULIA EVANS  
@b0rk

We hear computers "think in binary". But what does that MEAN??

your computer's memory  
(RAM)

0100100100010100010

0100



Some of this  
is cat pictures



Some of  
this is  
\* programs \*

programs are binary

0100101101110101100



ooo what does  
THAT mean

those are  
INSTRUCTIONS

every CPU has an  
≡ instruction set ≡  
(x86 or ARM)  
usually

some instructions

jmp

mov

add

xor

push

inc

instructions are  
numbers

the **inc** instruction  
("increment")

on x86 is  
1000000 or 0x40

assemblers translate "human  
readable" assembly code  
into binary

assembly code  $\xrightarrow{\text{assembler}}$  binary

mov \$1, %rax      --01001001  
mov \$1, %rdi      0....  
xor %rdi, %rdi

↑  
register

the **instruction register**  
contains an address in  
RAM.

CPU  $\xrightarrow{\text{IR}}$  0x5884

I will go  
look in RAM there  
and run the code  
I find!

"C" stands for linearizable

# the CAP theorem

Julia Evans @b0rk

from Martin Kleppmann's "A critique of the CAP theorem"

in distributed systems, network partitions happen

???

hello?

computer

someone unplugged a cable!

garbage collector! too much network traffic!

if you want to be consistent you can't always be available

panda

elephant

you're gonna have to wait for an answer

"CP systems"

consul

zookeeper

etcd

chubby

when they reply, you can believe them, but they don't always give you answers

"AP systems" available + partition tolerant

this doesn't mean very much.

always return "lol"

very carefully considered weaker consistency model

You can call both of these "AP"

CAP is a very simple theorem

I read the whole proof! It took 10 minutes + there's no math

CAP won't help you reason about most systems

I have a replicated database what can you tell me?

nothing! 😊

CAP

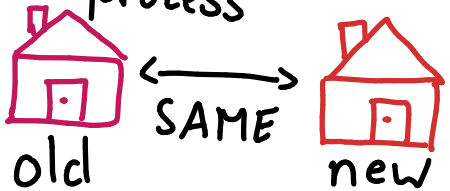
JULIA EVANS  
@b0rk

# copy on write

drawings.jvns.ca

every time you start a new process on Linux, it does a

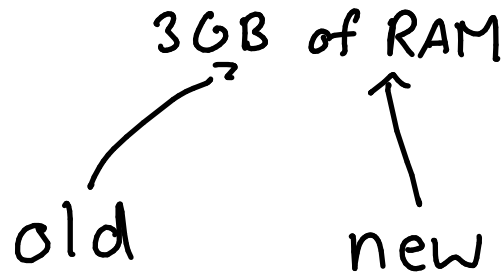
`fork()` "clone" or "clone" which copies the parent process



old ← SAME → new

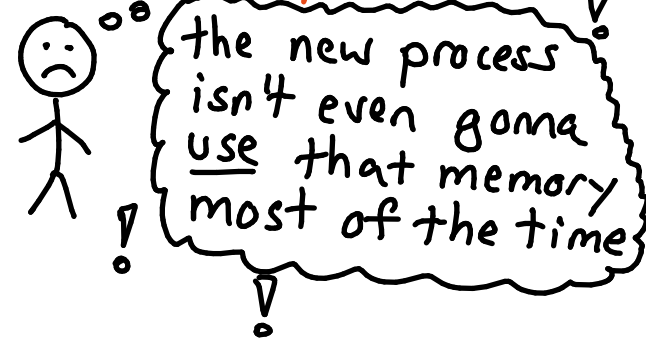
the cloned process has EXACTLY the same memory

3GB of RAM




old → 3GB of RAM ← new

copying all the memory every time we fork would be **slow** and a **waste of space**.



the new process isn't even gonna use that memory most of the time!

so Linux lets them share RAM instead of copying




Oh no! won't the processes pollute each others' memory?

how do we make this work?!

Linux marks all the memory for both processes as **read-only** (in the page table)

①



I'm going to write to the shared memory!

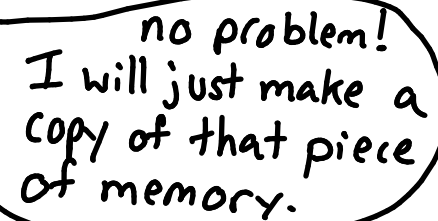
②

CPU

UHHH that is not allowed!  
Linux! PAGE FAULT!

③

LINUX



no problem!  
I will just make a copy of that piece of memory.

④

everyone is happy ♥

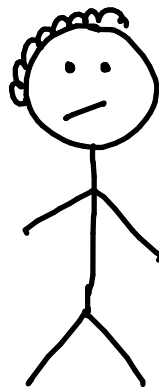
# directories + symlinks

@b0rk  
Julia Evans

drawings.jvns.ca

What's a directory?

<u>filename</u>	<u>inode number</u>
awesome.jpg	279932
blah.txt	13227
cumberbatch	233333



I made a directory  
with 2,000,000  
files

It's so  
SLOW

listing  
your directory  
is gonna be  
REAL SLOW

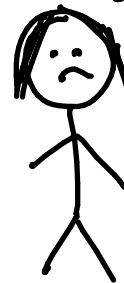
(a few seconds at least)

what's a symlink?

it's just a file with the  
name of another file in it! ▽

\$ **readlink** my-cool-link

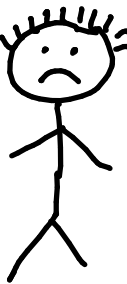
/home/julia/long-complicated-  
file-name



OLD

on ext 2 even opening  
files in big directories  
is slow ☹☹

that's right! ext 2  
directories have no index  
so you have to SEARCH  
THE WHOLE THING ☹



ext 2 is OLD though. ext 3 is  
OK.

# how does DNS work?

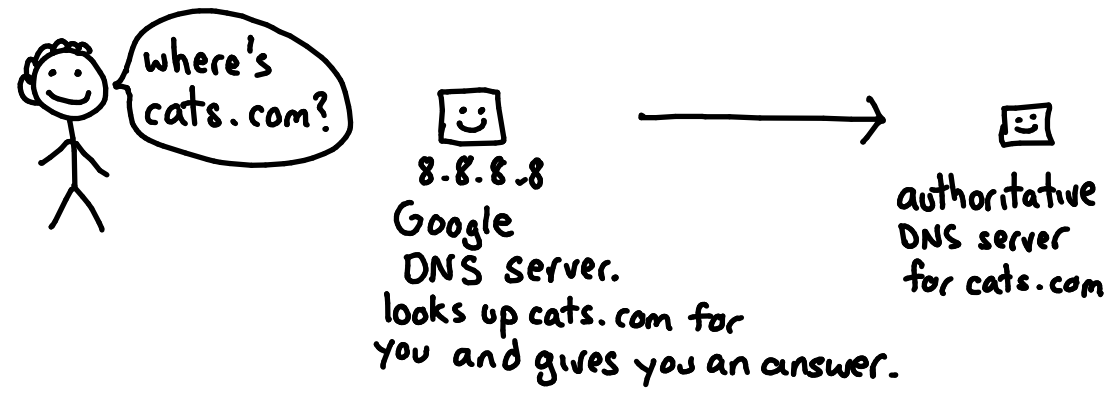
JULIA  
EVANS  
@bork

more of these at [drawings.jvns.ca](http://drawings.jvns.ca)

DNS servers translate names to IP addresses



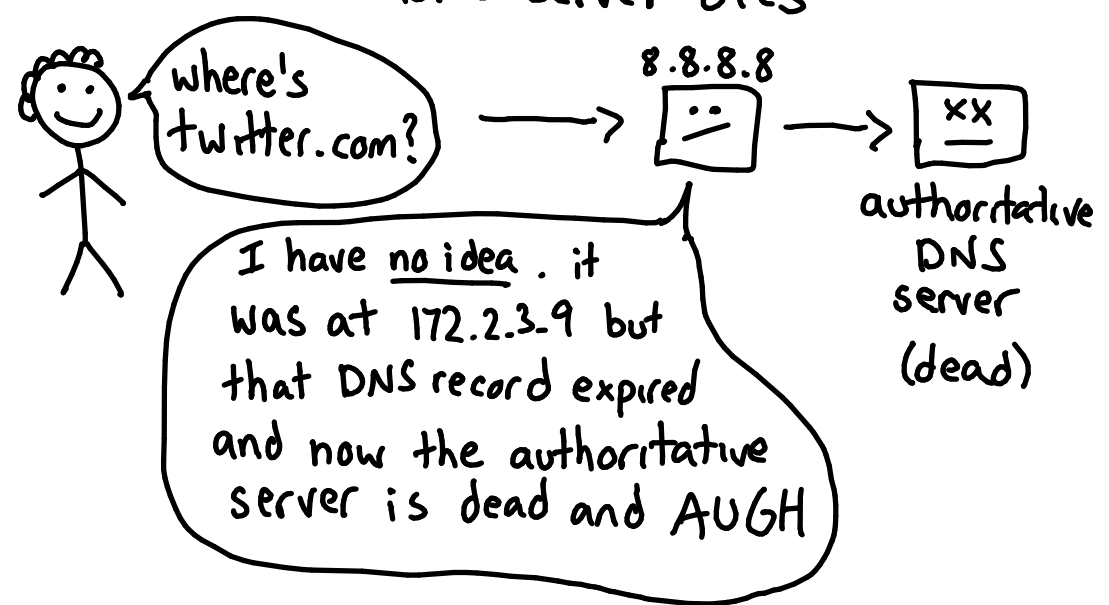
Most DNS queries get cached



sometimes they tell you it's an alias (CNAME record)



When an important DNS server dies



# floating point

JULIA EVANS  
@bork

more at: [drawings.jvns.ca](http://drawings.jvns.ca)

a double is 64 bits.

that means there are  $2^{64}$   
different doubles

going up to  
 $1.8 \times 10^{308}$

some double arithmetic

$$2^{52} + 0.2 = 2^{52} \quad (\text{the next number after } 2^{52} \text{ is } 2^{52} + 1)$$

$$1 + \frac{1}{2^{53}} = 1 \quad (\text{the next number after } 1 \text{ is } 1 + \frac{1}{2^{52}})$$

$3 \times 10^{100} = \text{infinity}$  ← infinity is a double  
 $\text{infinity} - \text{infinity} = \text{nan}$  (not a number)

there are  $2^{52}$  numbers  
between 1 and 2

$$1 + \frac{1}{2^{52}}, 1 + \frac{2}{2^{52}}, \dots$$

$2^{51}$  numbers between 2+4

$2^{50}$  between 4 and 8

etcetera.

Javascript only  
has doubles (and Lua?)



that means  
after  $2^{53}$   
you don't  
have every  
integer!

printing doubles  
is nontrivial

the shortest version  
of  $25.64853898042e8$

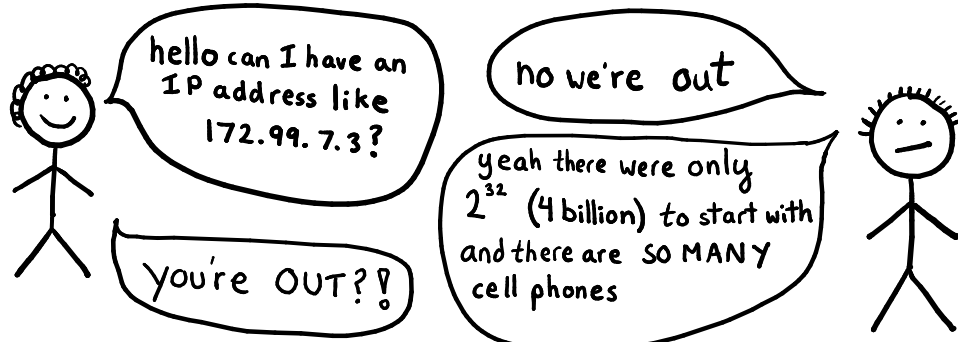
is  $2.564854e9$

↑  
calculating the  
shortest representation  
takes time!

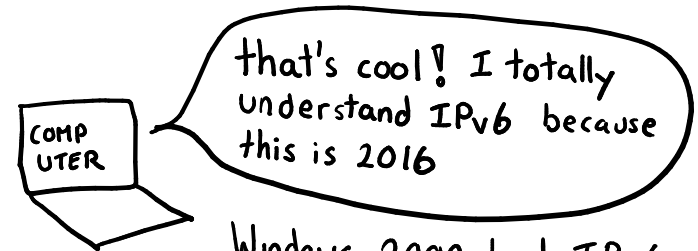
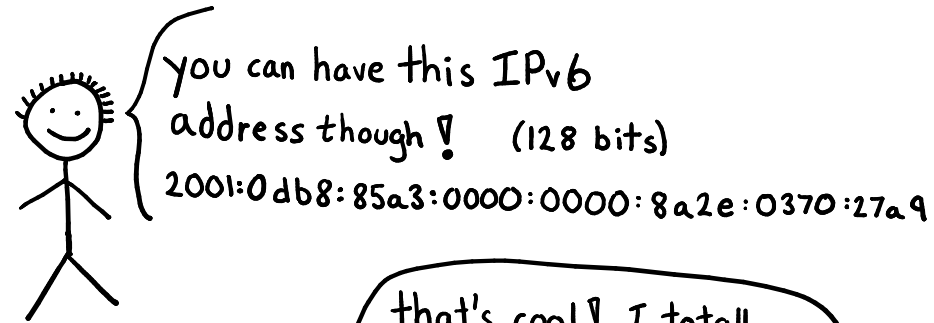
# IPv6

JULIA EVANS  
@b0rk

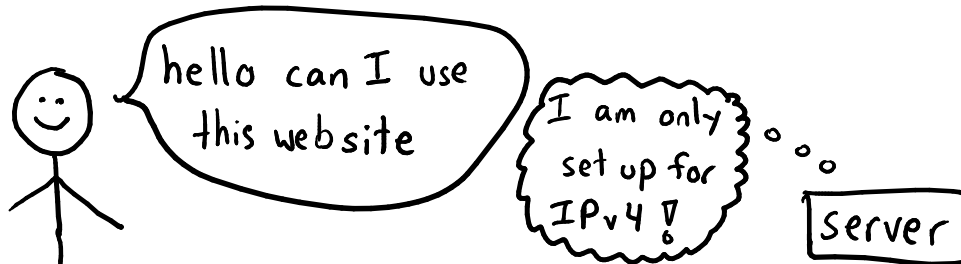
drawings.jvns.ca



What will we DO?! IPv4 addresses are 32 bits



Windows 2000 had IPv6 support. operating systems: SO READY



IPv6 user

sometimes people put translators in the middle to turn IPv6 packets into IPv4 packets

adoption  
it's happening!!!

Google says 30% of American traffic they see is using IPv6

people were putting it off but we're REALLY RUNNING OUT of IPv4 addresses so now they have no choice

# the "OSI model" for networking

JULIA EVANS  
@b0rk



I don't always find it useful but it's good to know what "layer 4" means



what does "this is an L4 proxy" mean?

## LAYERS

- 1: electrical engineering stuff, wires, frequencies, wifi
- 2: Ethernet protocol + others
- 3: IP (IP addresses)
- 4: TCP + UDP (ports)
- 5+6: nobody ever talks about these
- 7: HTTP and friends

If a load balancer is labelled "L7" it usually means it looks at the Host: header inside your HTTP packets.

layer 3  
networking  
tool

↑  
ignores layer  
4 and above

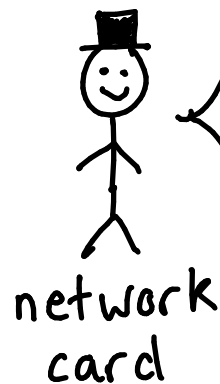
I only know about IP addresses!  
I don't even know what a port is let alone what the packet says



# what's a MAC address?

more at: [drawings.jvns.ca!](http://drawings.jvns.ca!)

every computer on the internet has a **network card**



hello! you can call me  
0a:58:ff:ea:05:97

↑  
**MAC address**

when you make HTTP requests with Ethernet/Wifi, every packet gets sent to a MAC address



here is a cat for 0a:58:...



0a:58

wait, how do I know someone **else** on the same network isn't reading all my packets?

you don't! that's one reason we use HTTPS + secure Wifi networks



your router has a table that maps IP addresses to MAC addresses



a message for 192.0.2.77?  
I will send that to  
0a:58:ff:ea:05:97!

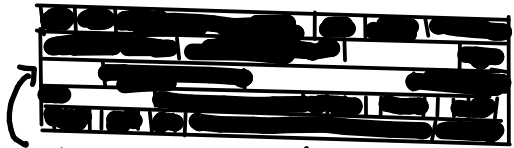
(read about ARP for more)

# memory allocation

Julia Evans  
@b0rk

at any given time  
your program has  
a fixed amount of  
memory

■ used  
587 MB □ free



this was used but then  
got freed

and it can ask  
the OS for more  
memory



google  
chrome

now I have  
1.8 GB of  
memory! yay!

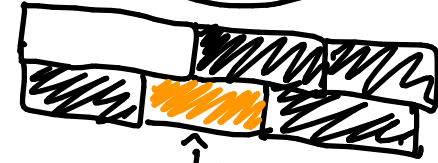
your allocator tries to  
fill in unused pieces when  
you ask for memory



can I have  
512 bytes of  
memory?

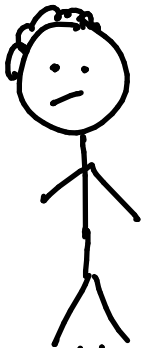
YES

malloc



↑ your new memory

you can invent your  
own strategy to allocate  
memory



glibc malloc's algorithm  
is dumb I'm going to  
do my own thing

especially  
if you understand  
your memory  
access  
patterns  
well

this is sort of normal to do  
if you care a LOT about performance

alternatives to libc malloc

jemalloc

Facebook

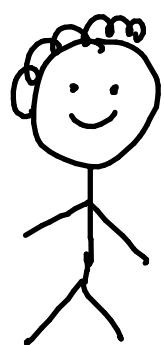
tcmalloc

Google

# man pages = awesome

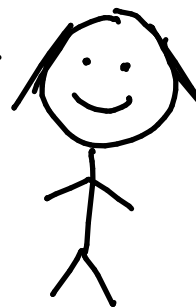
(sometimes. Quality may vary 😊)

JULIA EVANS  
@bark



I found out I can get documentation for programs (like grep) with **man grep**!

but that's not all!! lots of other things have man pages too!



GREAT

man pages are split up into 8 sections

- ① ② ③ ④ ⑤ ⑥ ⑦ ⑧

/usr/share/man/man5

has section 5 on my machine.

① programs

\$man grep  
\$man ls

③ C functions

\$man 3 printf  
\$man fopen

⑤ file formats

\$man sudoers  
for /etc/sudoers  
→ \$man proc

⑦ miscellaneous

\$man 7 pipe  
\$man 7 symlink  
(these are cool!)

② system calls

\$man sendfile

④ devices

\$man null  
for /dev/null docs

⑥ games

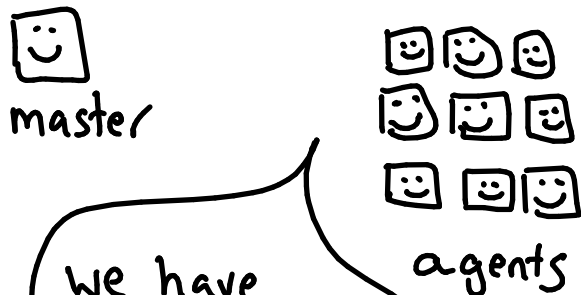
(not very useful)  
man sl is good if you have sl though

⑧ sysadmin programs

\$man apt  
\$man chroot

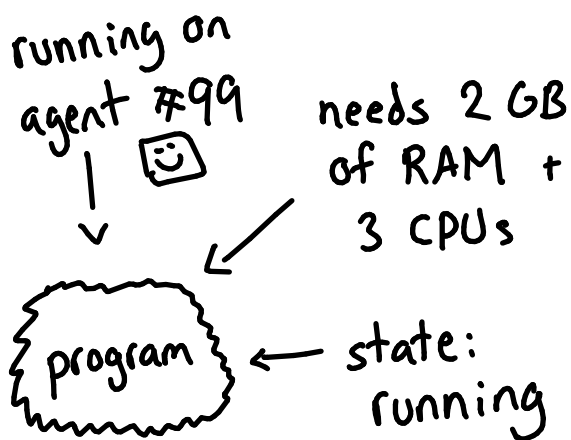
# mesos

mesos manages resources

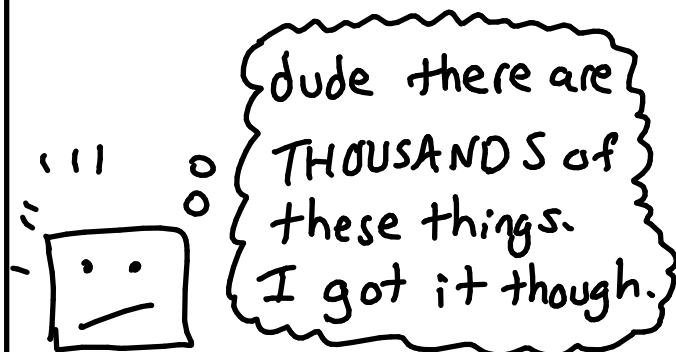


We have 200 CPUs + 800 GB of RAM. what should we do?

agents run "tasks"



the Mesos master keeps track of EVERY running task



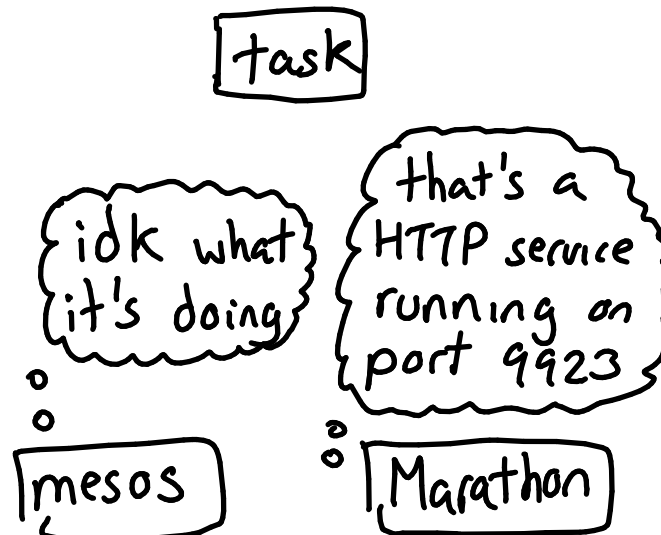
**frameworks** ask the Mesos master to run tasks there are LOTS.

- Marathon (HTTP services)
- Chronos (cron-like jobs)
- Jenkins
- Spark
- Hadoop
- Elastic Search
- Cassandra

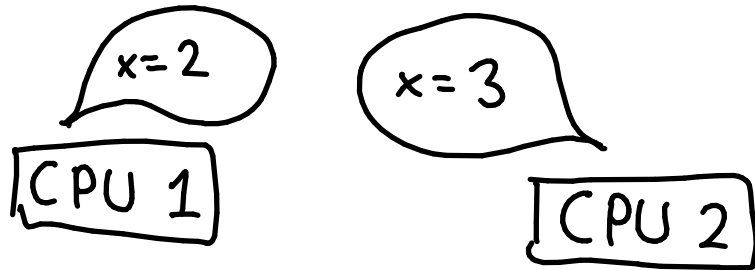
you can split your Mesos cluster between several frameworks



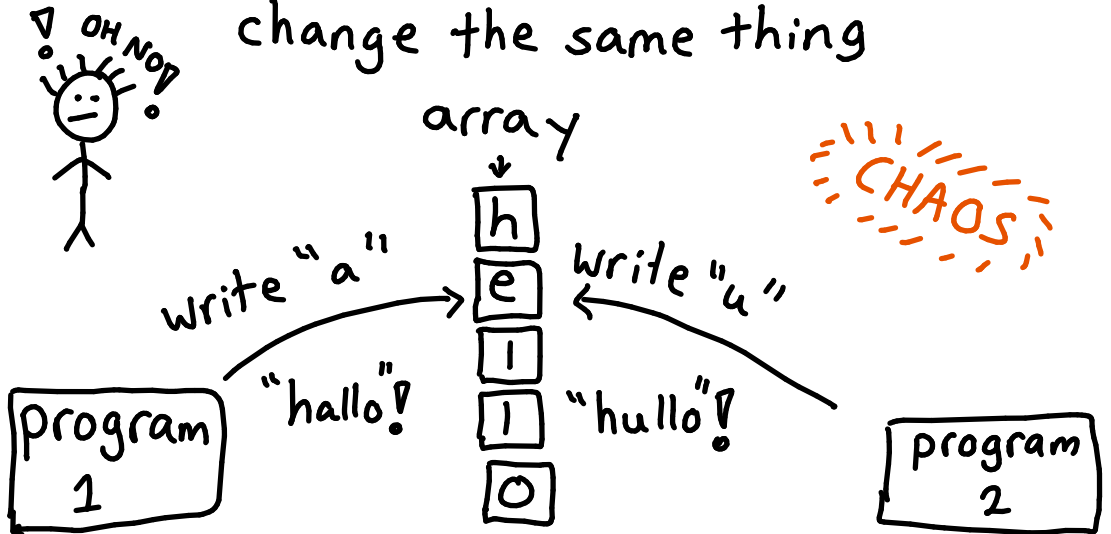
Mesos doesn't know much about tasks



Sometimes you're running code on 2 CPUs at the same time

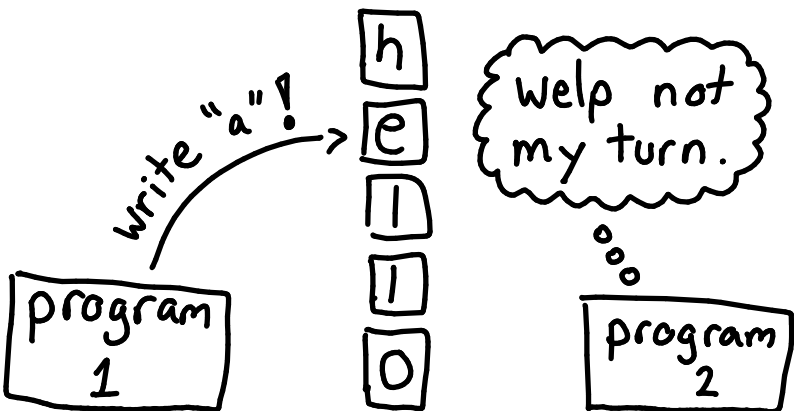


Sometimes 2 threads want to change the same thing



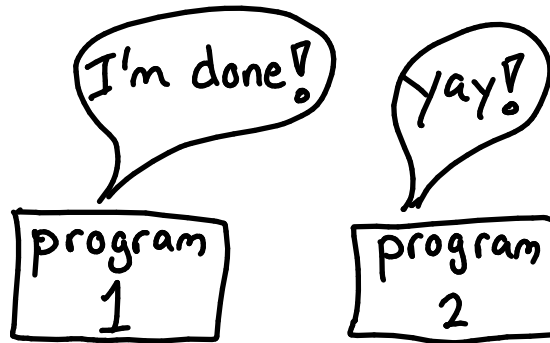
a **mutex** keeps track of whether something is in use

♥ program 1's turn ♥  
♥ mutex



when you're done, you tell the mutex it's available

♥ available ♥  
mutex

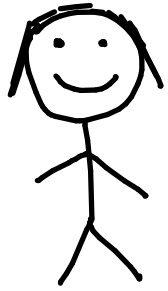


there's lots more but we're outta space

- semaphores
- mutex
- compare and swap
- atomic instructions

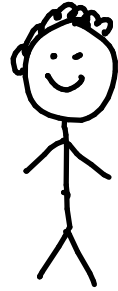
# networking concepts

JULIA EVANS  
@b0rk



hey I want to understand all the networking stuff that happens when I go to google.com!

YES that is awesome. there are a lot of concepts but you can totally learn them all!



(knows many networking concepts now)

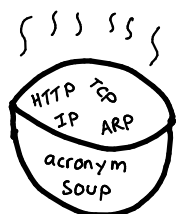
## protocols

- DNS
- SSL/TLS
- IP
- TCP
- UDP
- ICMP (ping)
- ARP
- BGP
- ethernet

## other concepts

- socket
- packet
- port
- IP address
- nameserver
- NAT
- router
- TTL
- checksum
- +some more

it's a lot to learn but it's totally possible to learn how it all fits together to get you pictures of cats ♡



JULIA EVANS  
@b0rk

# anatomy of a packet

more at  
[drawings.jvns.ca](http://drawings.jvns.ca)

When you get a webpage like Facebook, it comes into your computer in many small **packets**

Let's see what those look like!

Packets are split into a few sections (or "headers")

ethernet/wifi  
82:53:ac:99:2f:33 ← MAC address

← "physical layer". this gets changed constantly as your packet moves between computers.

IP ("internet protocol")  
FROM: 172.96.2.3 TO: 123.9.2.32

← in charge of getting your packet to the right server (like an address on an envelope)

TCP (or UDP)  
sequence number: 877392 ← counts bytes sent so far  
checksum: 8447  
↑ detect corrupted data  
from: port 9979 to: port 80

← in charge of preventing data corruption and helping you retry lost packets

video streaming uses UDP instead. UDP does not try to be reliable.

HTTP (or whatever)

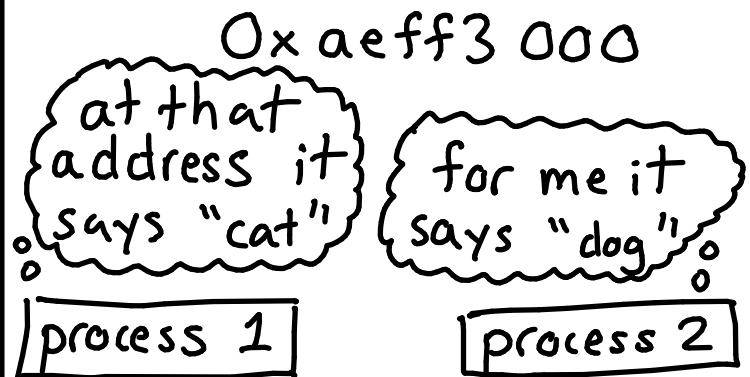
GET / HTTP / 1.1  
Host: google.com  
Accept-Language: en-US

← the actual data you're trying to send!

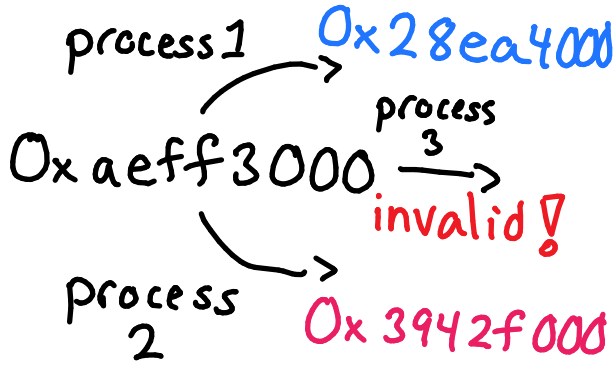
JULIA EVANS  
@b0rk

# page table (in 32 bit memory)

every process has its own memory space



each address maps to a 'real' address in physical RAM



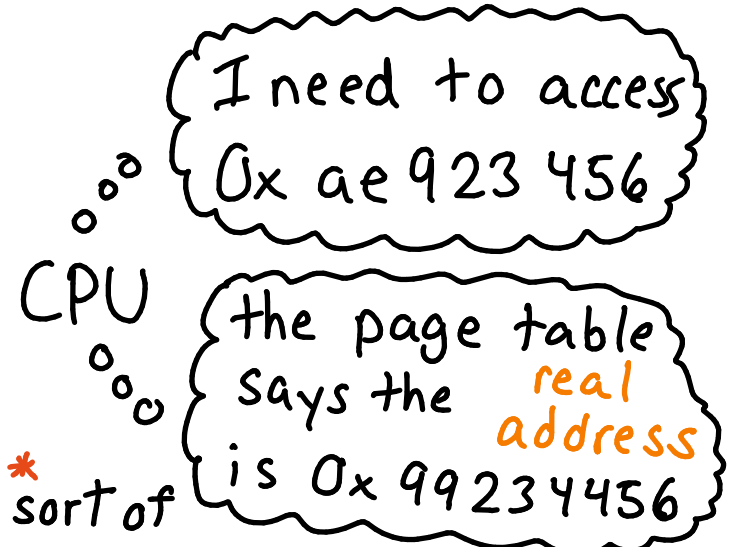
processes have a "page table" in RAM that stores all their mappings

0x12345000 → 0xae925...

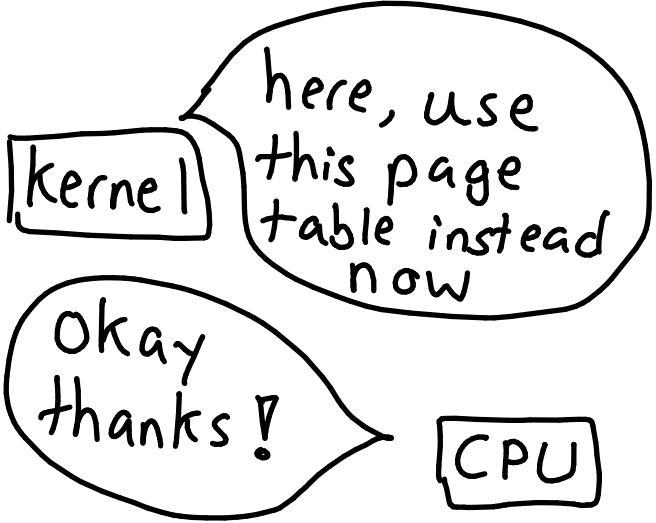
0x23f49000 → 0x12345...

the mappings are usually 4kB blocks (4kB is the normal size of a "page")

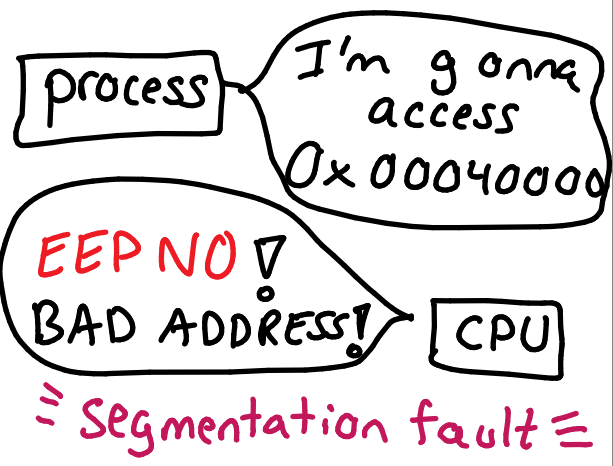
every\* memory access uses the page table



when you switch processes...



some pages don't map to a physical RAM address





# unix permissions

JULIA EVANS  
@bork

3 kinds of things  
you can do to a file

↓  
read    ↓    write    ↓    execute

```
$ ls -l awesome.png
-rw-r--r-- bork staff
```

↑                    ↑                    ↖  
bork can            staff can            ANYONE  
do this            do this            can do  
(user)            (group)            this

```
$ ls -l /bin/ping
-rwsr-xr-x root root
```

↑  
setuid flag

This means ping always  
runs as root (who owns  
it), no matter who  
started ping

what's this  
755 business?

7	means	rx
6	→	r-
5	→	-x
4	→	--

it's binary?  
5 → 101 → r-x

755 means  
rx r-x r-x

more weird  
permissions  
things

setgid

sticky bit

but I ran out  
of space

# pipes

JULIA EVANS  
@b0rk

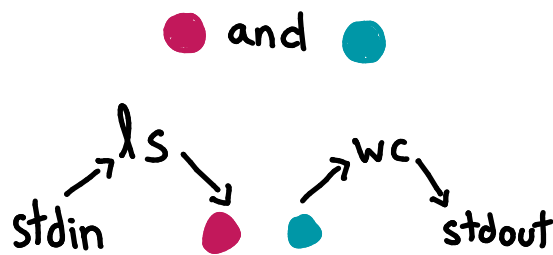
drawings.jvns.ca

Sometimes you want to send the output of one process to the input of another

```
$ ls | wc -l
```

53  
↖ 53 files!

a pipe is a pair of 2 magical file descriptors



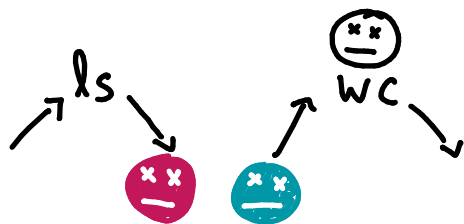
When `ls` does `write(●, "hi")`  
`wc` can read it!  
`read(●)`  
→ "hi"

## pipe buffers

ls: I'm gonna write a bajillion bytes to ●

uh no if my buffer is full you have to wait ●

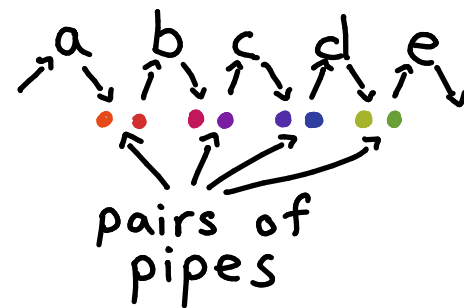
what if your target process dies?



ls gets sent SIGPIPE if ● gets closed (ls usually dies)

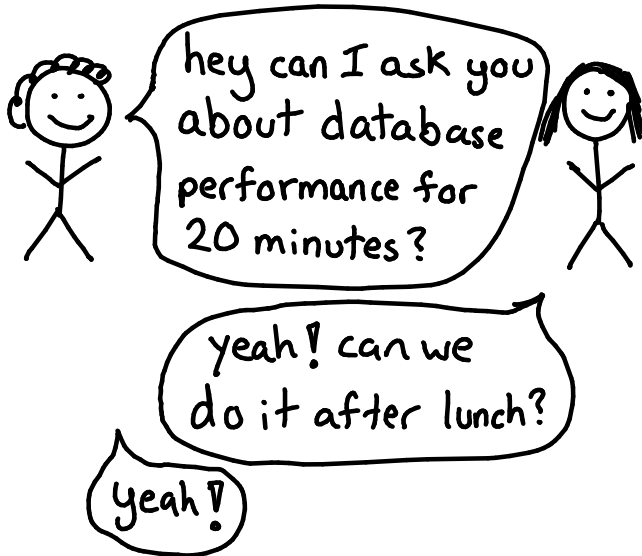
you can pipe SO MANY things together

```
$ a | b | c | d | e
```



# asking good questions

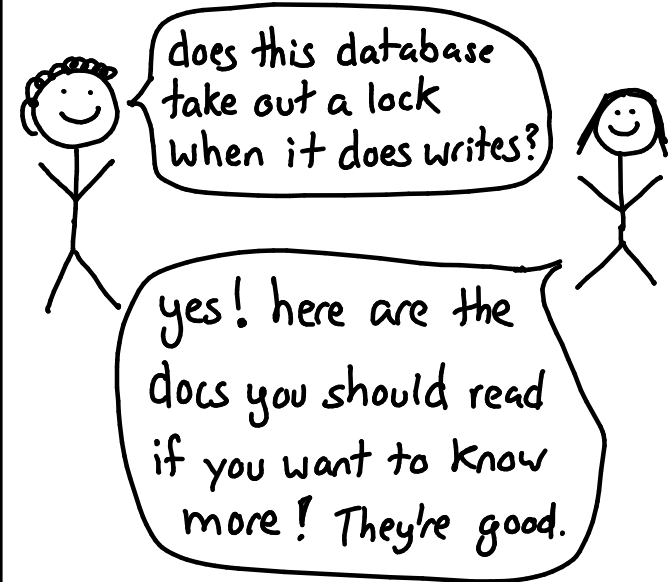
## find a good time



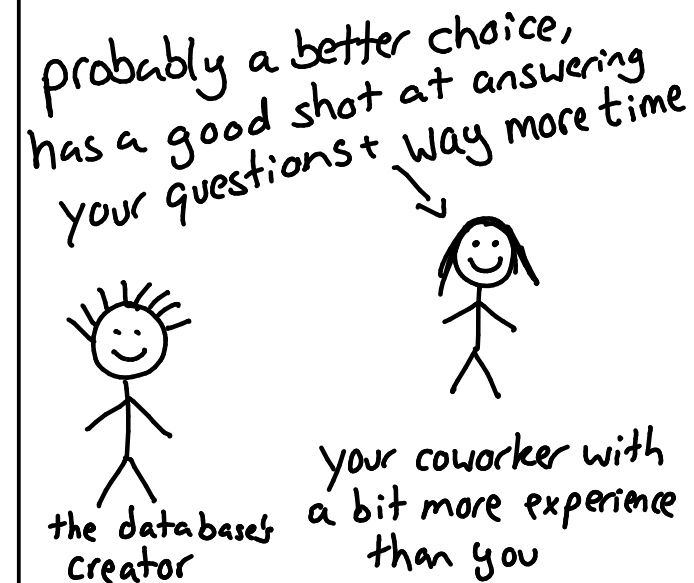
## state what you know



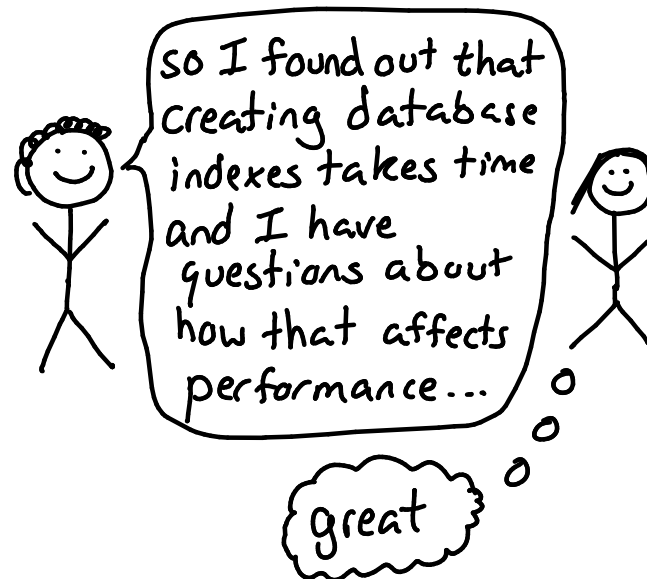
## ask factual questions



## choose who to ask



## do some research



## profit



once upon a time...

I want to run a program!

Sorry I can only do 1 program at a time

COMPUTARR

from the 60s or so

your computer today

I want to run!

no, me!

ME

I have stuff to do too you know!

program

operating system

every CPU core can only run 1 program at a time

ok time to stop it's Jimmy's turn to use the CPU now

operating system

every program gets a few ms at a time

steps when we switch the running process

"context switch"

- save:
  - registers
  - stack pointer
  - which CPU instruction to start at next time
- set up memory for new process
- load new registers and stuff

all this takes time (2 microseconds?).

- ↳ It's ok to do but
- ↳ you don't want to be switching processes constantly

you don't use the CPU when you're waiting

hey I'm waiting for a network response

cool! I'll run other stuff until that comes back.

OS

more at  
drawings:jvns.ca

# ★ the stack ★ (in a C program)

JULIA EVANS  
@bork

your program has

-> local variables

```
int x = 2;
```

-> a function to return to

```
void parent() {  
    do_thing();  
}
```

-> function arguments

```
make_cat(name, fluffiness)
```

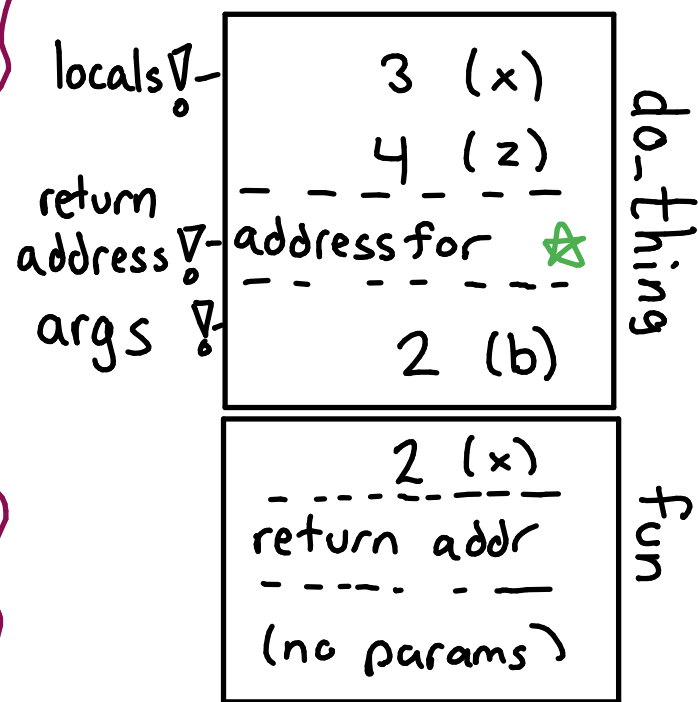
these all live in a part  
of memory called  
**the stack**

example program

```
int fun () {  
    int x = 2;  
    do_thing(2);  
    int y = 4; ★  
}
```

```
void do_thing(b) {  
    int x = b + 1;  
    int z = 4; ←←  
    return; ←←
```

the stack at ←←



there's a limit  
to how big  
your stack can  
get! Exceed it  
and you get a  
**STACK OVERFLOW**

# What's a thread?

JULIA EVANS  
@b0rk

drawings.jvns.ca

a process can have lots of threads

process id	thread id
1888	1888
1888	1892
1888	1893
1888	2007

threads share memory

thread 1: I'm going to write "3" to address 2977886

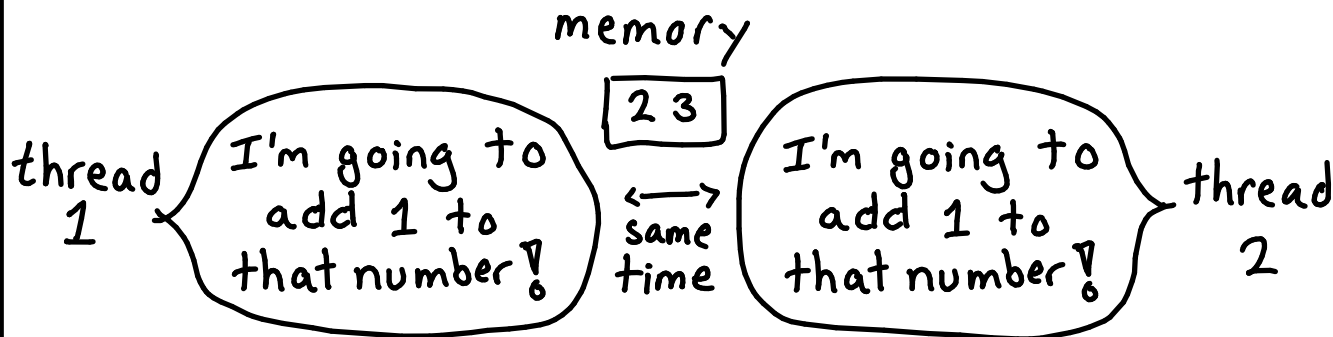
thread 2: I can overwrite that if I want!

but they can run different code

thread 1: I'm doing a calculation!

thread 2: I'm waiting for a network request to finish!

sharing memory can cause problems  
"race conditions"



RESULT: 24 ← WRONG (should be 25)

if you have 8 CPU cores, you can run code for 8 threads at the

SAME TIME

1 5  
2 6  
3 7  
4 8  
CPU cores

SO BUSY ☺☺

# user space vs. kernel space

JULIA EVANS  
@b0rk

drawings.jvns.ca

the Linux kernel has **millions** of lines of code

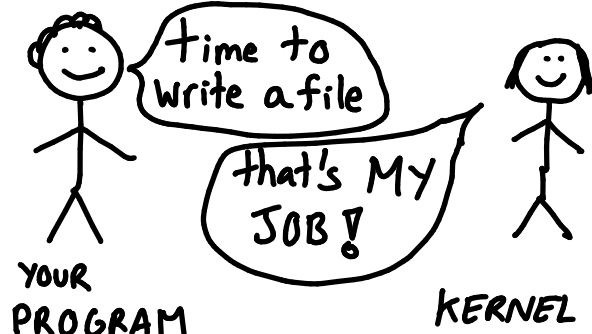
- ★ read+write files
- ★ decide which programs get to use the CPU
- ★ make the keyboard work

When Linux kernel code runs, that's called

**kernel space**

When your program runs, that's

**user space**



time for a **context switch** to kernel space

your program switches back and forth

```
str = "my string"
```

```
x = x + 2
```

```
file.write(str) ← ★ switch to kernel space ★
```

```
y = x + 4  
str = str * y ← ★ and we're back to user space! ★
```

timing your process

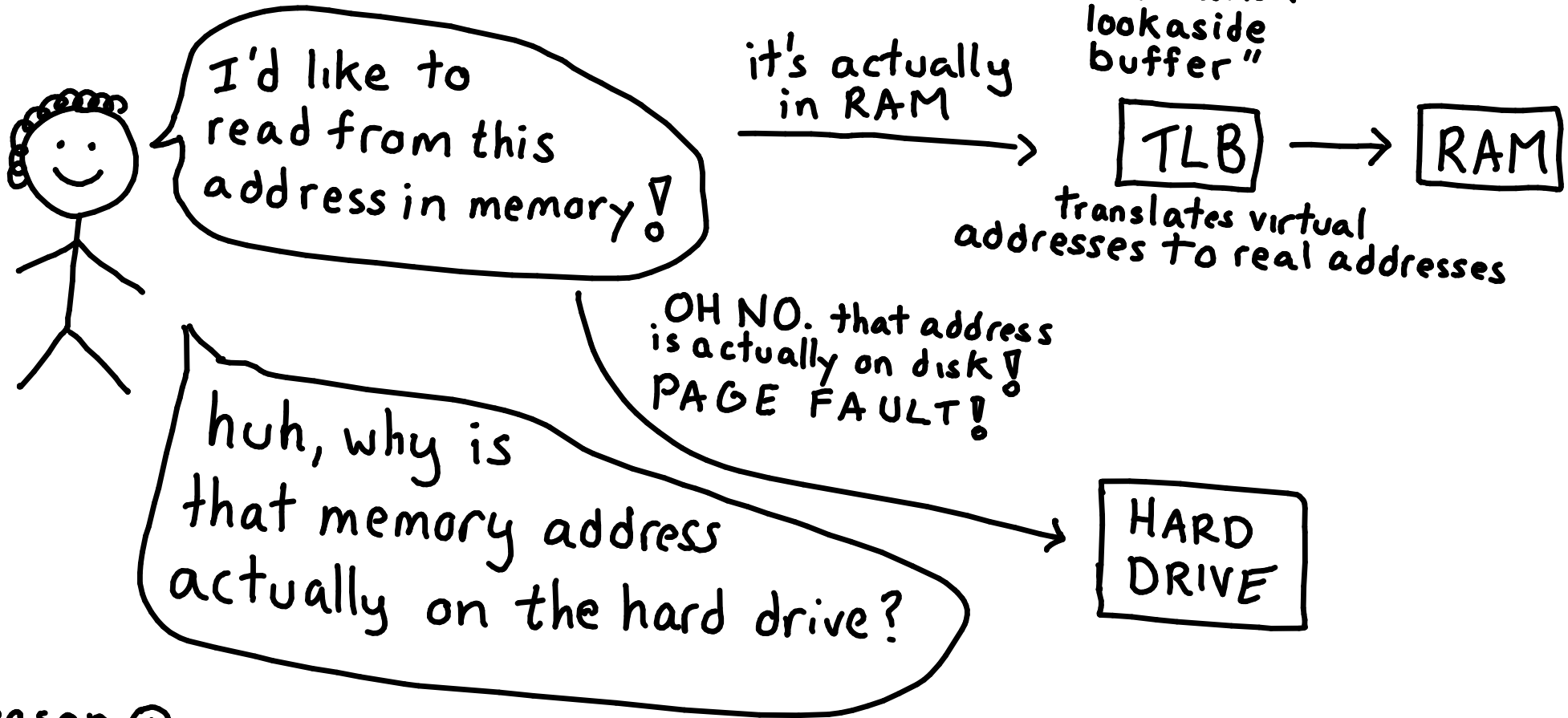
\$ time find /home

0.15 user      0.73 system

↑  
time spent in  
your process

↑  
time spent by  
the kernel doing  
work for your  
process

# ≡ VIRTUAL MEMORY ≡



reason ①

mmap

lets you map a bunch of stuff on disk into memory. None of it will actually get read from disk until you access the memory.

reason ②

swap

if you run out of memory, it gets saved to disk and your computer gets SUPER SLOW !!