CSO2 (CS3130)

changelog [since lecture]

C exericse (2): correct return type of `mystery` to void

warmup: correct due date for this semester

waitlist: adjust to reflect lecture waitlist being current constraint

themes

automating building software

- libraries, taking advantage of incremental compilation

sharing machines

- multiple users/programs on one system

parallelism and concurrency

- doing two+ things at once

under the hood of sockets

- layered design of networks
- implementing secure communication

under the hood of fast processors

- caching, (hidden) parallelism, avoiding idle time

themes

automating building software

- libraries, taking advantage of incremental compilation

sharing machines

- multiple users/programs on one system

parallelism and concurrency

- doing two+ things at once

under the hood of sockets

- layered design of networks
- implementing secure communication

under the hood of fast processors

- caching, (hidden) parallelism, avoiding idle time

make

```
$ ./foo.exe
...
...
$ edit readline.c
$ make
clang -g -O -Wall -c readline.c -o readline.o
ar rcs terminal.o readline.o libreadline.a
clang -o foo.exe foo.o foo-utility.o -L. -lreadline
$
```

themes

automating building software

- libraries, taking advantage of incremental compilation

sharing machines

- multiple users/programs on one system

parallelism and concurrency

- doing two+ things at once

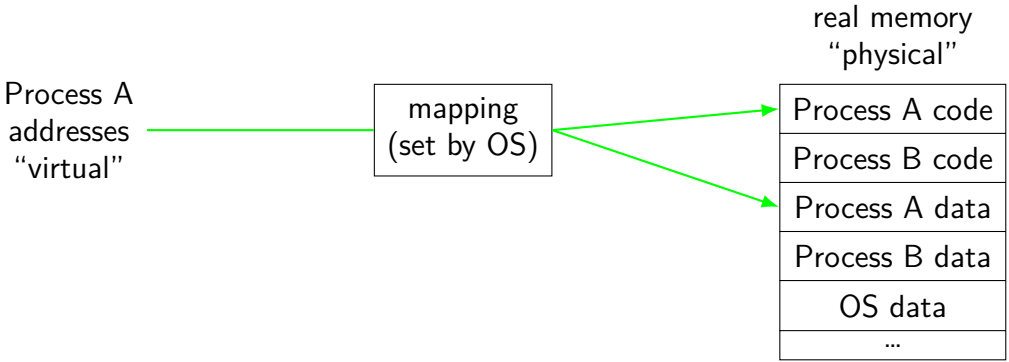
under the hood of sockets

- layered design of networks
- implementing secure communication

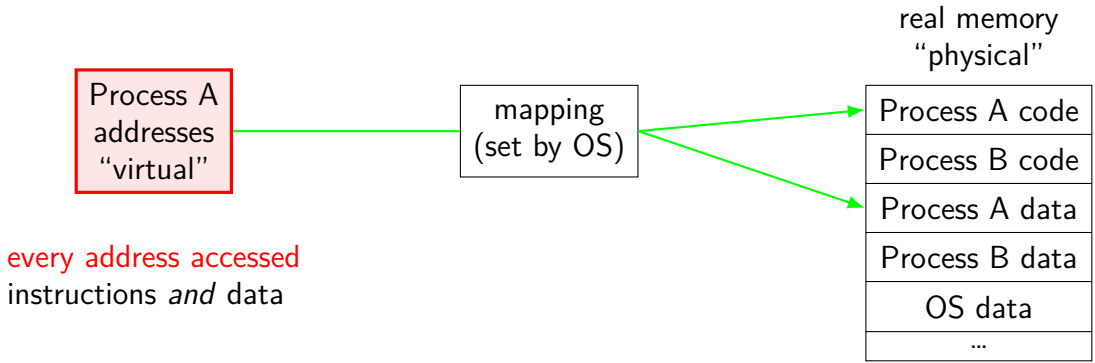
under the hood of fast processors

- caching, (hidden) parallelism, avoiding idle time

address translation

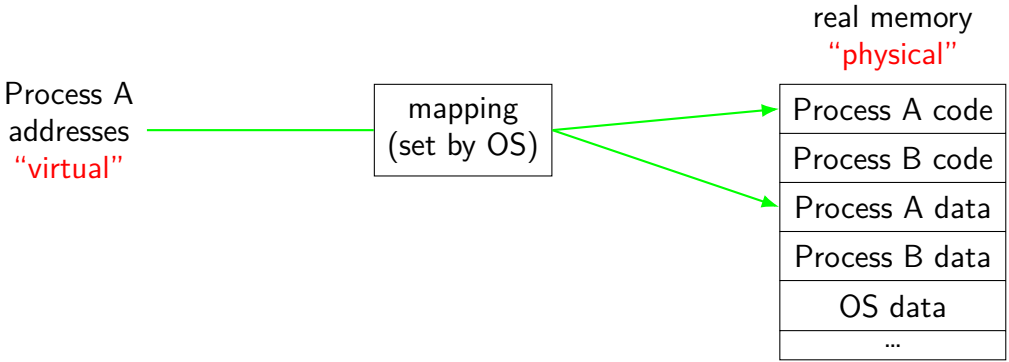


address translation



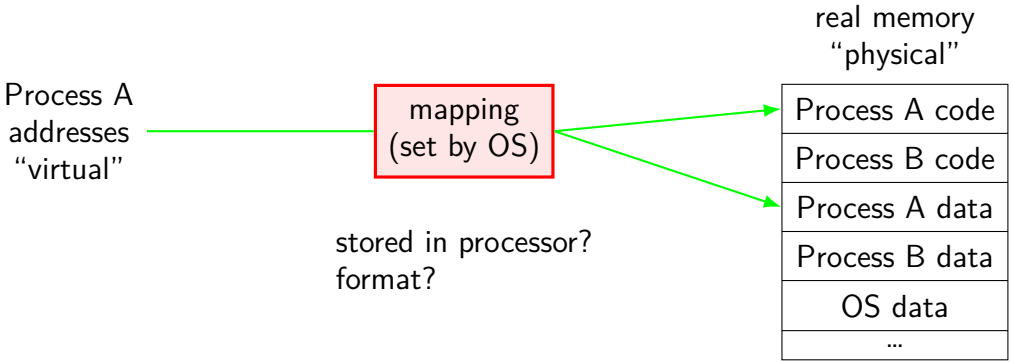
every address accessed
instructions *and* data

address translation



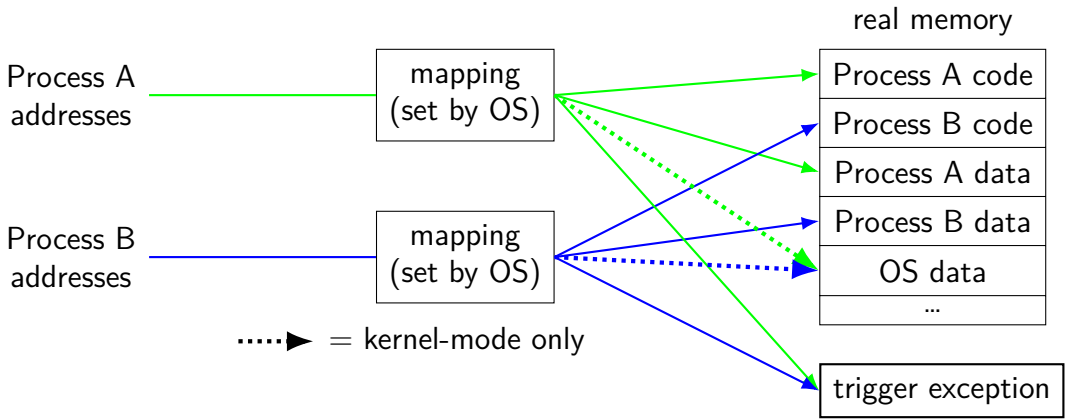
program addresses are 'virtual'
real addresses are 'physical'
can be different sizes!

address translation



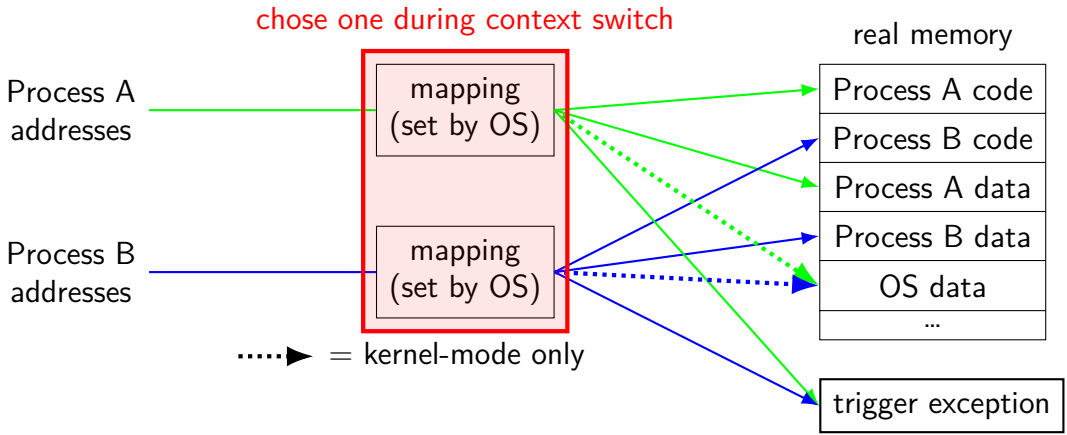
address spaces

illuision of dedicated memory



address spaces

illuision of dedicated memory



themes

automating building software

- libraries, taking advantage of incremental compilation

sharing machines

- multiple users/programs on one system

parallelism and concurrency

- doing two+ things at once

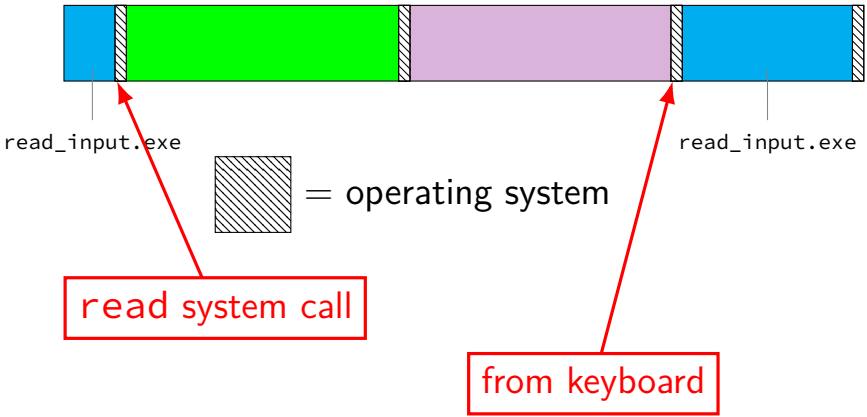
under the hood of sockets

- layered design of networks
- implementing secure communication

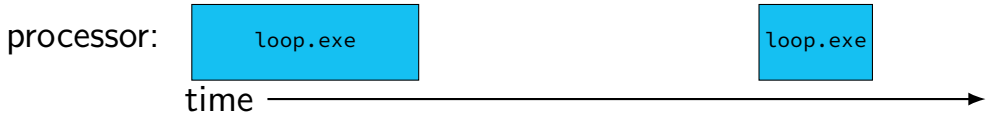
under the hood of fast processors

- caching, (hidden) parallelism, avoiding idle time

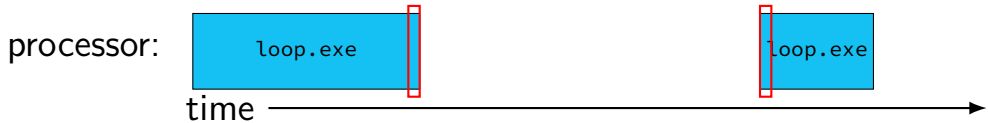
keyboard input timeline



time multiplexing



time multiplexing

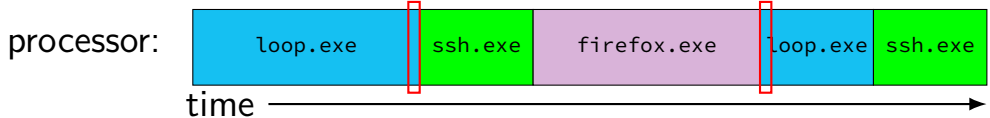



```
...
call get_time
    // whatever get_time does
movq %rax, %rbp
```

million cycle delay

```
call get_time
    // whatever get_time does
subq %rbp, %rax
...
```

time multiplexing

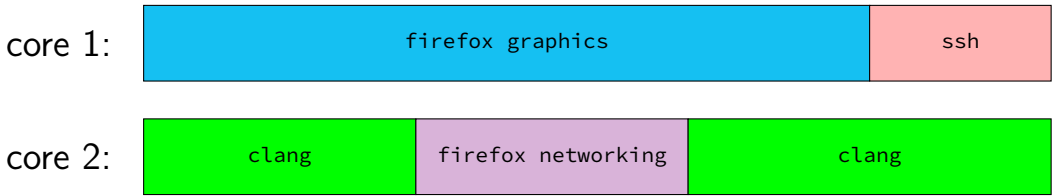



```
...
call get_time
    // whatever get_time does
movq %rax, %rbp
```

million cycle delay

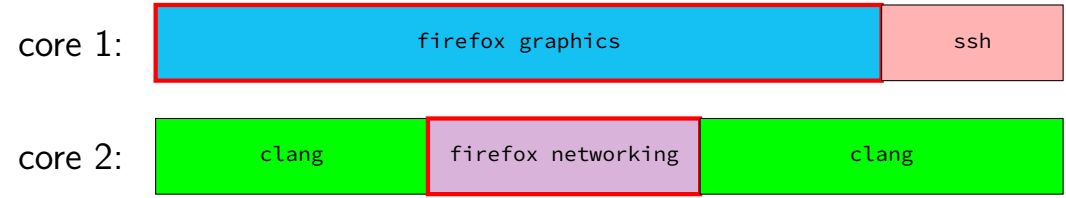
```
call get_time
    // whatever get_time does
subq %rbp, %rax
...
```

multiple cores+threads



multiple cores? each core still divided up

multiple cores+threads



one program with multiple *threads*

themes

automating building software

- libraries, taking advantage of incremental compilation

sharing machines

- multiple users/programs on one system

parallelism and concurrency

- doing two+ things at once

under the hood of sockets

- layered design of networks
- implementing secure communication

under the hood of fast processors

- caching, (hidden) parallelism, avoiding idle time

permissions

```
$ ls /u/other/secret
ls: cannot open directory '/u/other/secret': Permission denied
$ shutdown
shutdown: Permission denied
```

themes

automating building software

- libraries, taking advantage of incremental compilation

sharing machines

- multiple users/programs on one system

parallelism and concurrency

- doing two+ things at once

under the hood of sockets

- layered design of networks
- implementing secure communication

under the hood of fast processors

- caching, (hidden) parallelism, avoiding idle time

layers

application	HTTP, SSH, SMTP, ...	application-defined meanings
transport	TCP, UDP, ...	reach correct program, reliablity/streams
network	IPv4, IPv6, ...	reach correct machine (across networks)
link	Ethernet, Wi-Fi, ...	coordinate shared wire/radio
physical	...	encode bits for wire/radio

names and addresses

name	address
logical identifier	location/how to locate
variable `counter`	memory address `0x7FFF9430`
DNS name `www.virginia.edu`	IPv4 address `128.143.22.36`
DNS name `mail.google.com`	IPv4 address `216.58.217.69`
DNS name `mail.google.com`	IPv6 address `2607:f8b0:4004:80b::2005`
DNS name `reiss-t3620.cs.virginia.edu`	IPv4 address `128.143.67.91`
DNS name `reiss-t3620.cs.virginia.edu`	MAC address `18:66:da:2e:7f:da`
service name `https`	port number 443
service name `ssh`	port number 22

secure communication?

how do you know who your socket is to?

who can read what's on the socket?

what can you do to restrict this?

themes

automating building software

- libraries, taking advantage of incremental compilation

sharing machines

- multiple users/programs on one system

parallelism and concurrency

- doing two+ things at once

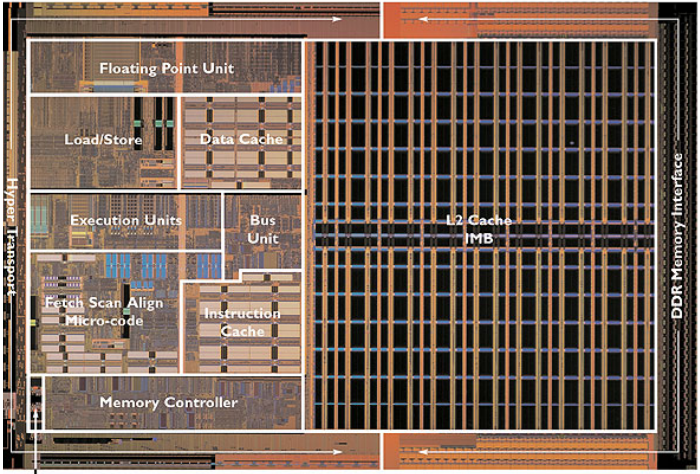
under the hood of sockets

- layered design of networks
- implementing secure communication

under the hood of fast processors

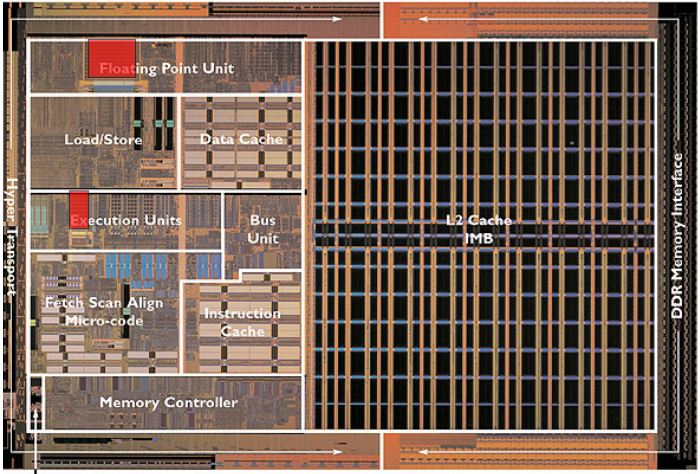
- caching, (hidden) parallelism, avoiding idle time

2004 CPU



ce: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU

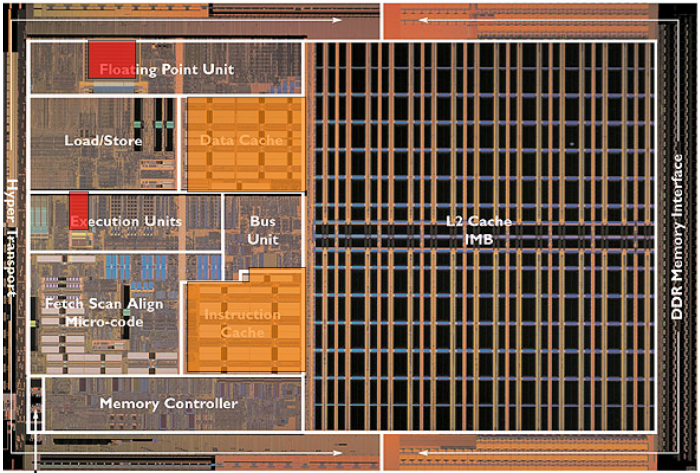


Registers

AMD

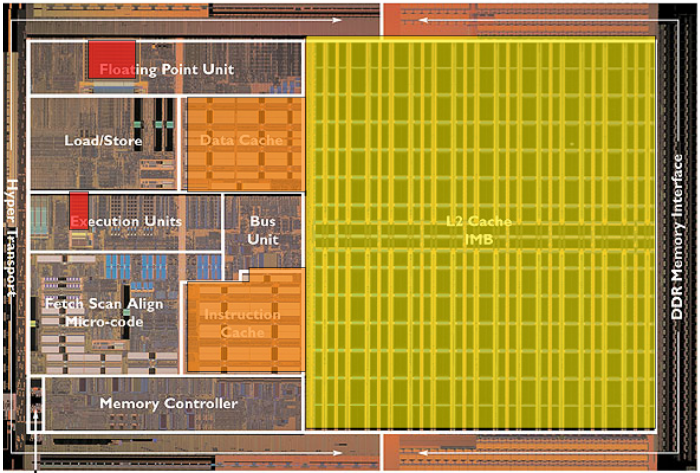
ce: approx 2004 AMD press image of Opteron die;
ox register location via chip-architect.org (Hans de Vries)

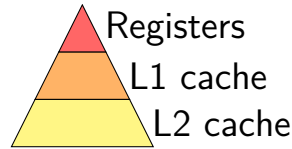
2004 CPU



ce: approx 2004 AMD press image of Opteron die;
prox register location via chip-architect.org (Hans de Vries)

2004 CPU

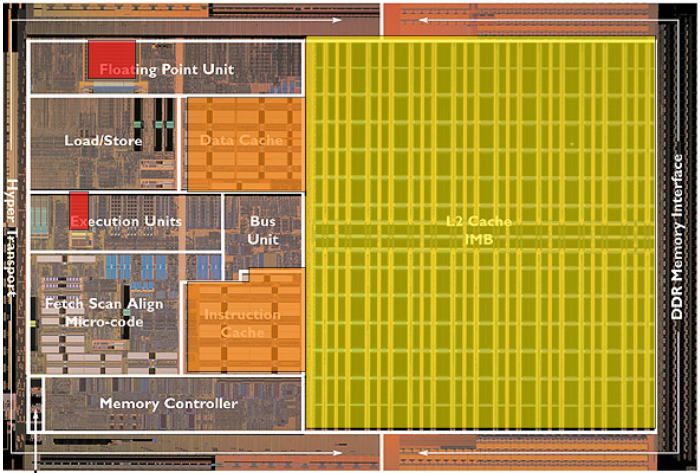


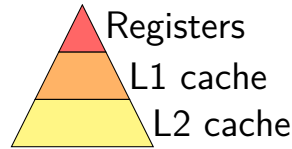


AMD

ce: approx 2004 AMD press image of Opteron die;
prox register location via chip-architect.org (Hans de Vries)

2004 CPU

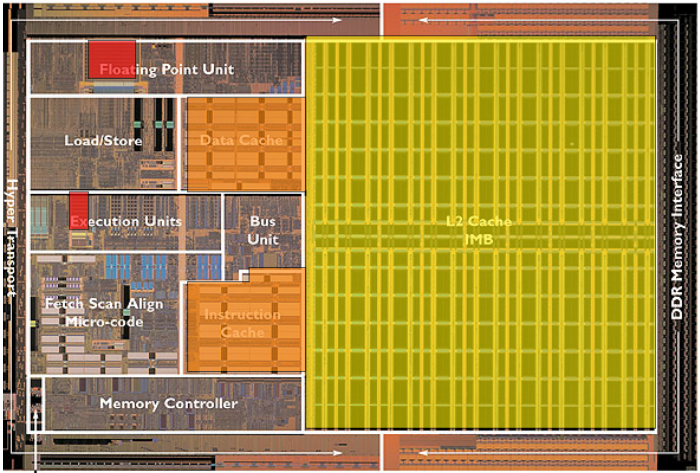


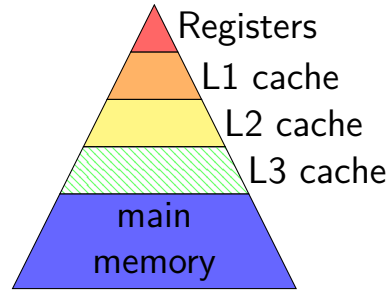


AMD

ce: approx 2004 AMD press image of Opteron die;
prox register location via chip-architect.org (Hans de Vries)

2004 CPU

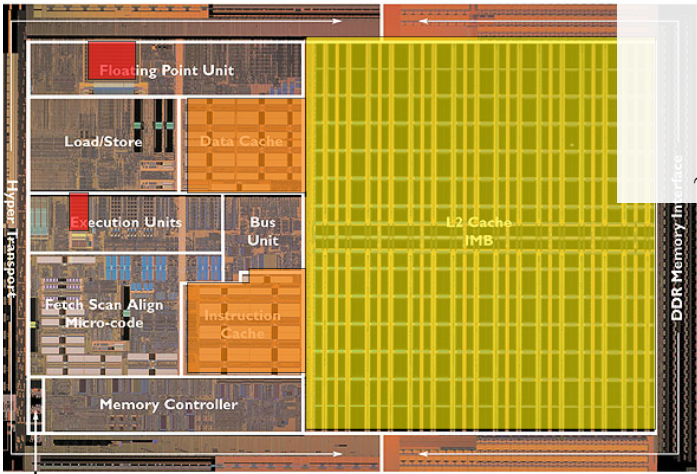


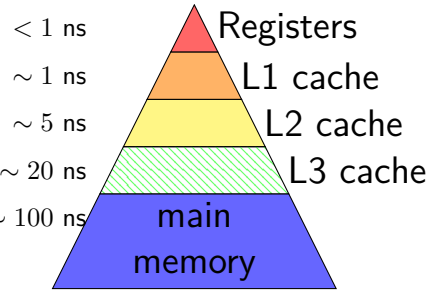


AMD

ce: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU





ce: approx 2004 AMD press image of Opteron die;
prox register location via chip-architect.org (Hans de Vries)

some performance examples

```
example1:
    movq $10000000000, %rax
loop1:
    addq %rbx, %rcx
    decq %rax
    jge loop1
    ret
```

about 30B instructions

my desktop: approx 2.65 sec

```
example2:
    movq $10000000000, %rax
loop2:
    addq %rbx, %rcx
    addq %r8, %r9
    decq %rax
    jge loop2
    ret
```

about 40B instructions

my desktop: approx 2.65 sec

some performance examples

```
example1:
    movq $10000000000, %rax
loop1:
    addq %rbx, %rcx
    decq %rax
    jge loop1
    ret
```

about 30B instructions

my desktop: approx 2.65 sec

```
example2:
    movq $10000000000, %rax
loop2:
    addq %rbx, %rcx
    addq %r8, %r9
    decq %rax
    jge loop2
    ret
```

about 40B instructions

my desktop: approx 2.65 sec

C exercise

```
int array[4] = {10,20,30,40};
int *p;
p = &array[0];
p += 2;
p[1] += 1;
```

array =

A.	compile or runtime error	B.	{10,20,30,41}
C.	{10,20,32,41}	D.	{10,21,30,40}
E.	{12,21,30,40}	F.	none of these

C exercise (2)

```
int *array2[4]; int array1[4] = {10,20,30,40};
void mystery(int **p) {
    *p = &array1[2];
}
int main() {
    int **q;
    q = array2;
    mystery(q);
    array1[1] = *q;
    ...
}
```

array1 =

A.	compile or runtime error	B.	{10,10,30,40}
C.	{10,30,30,40}	D.	{10,10,20,30}
E.	{10,20,10,20}	F.	none of these

some avenues for review

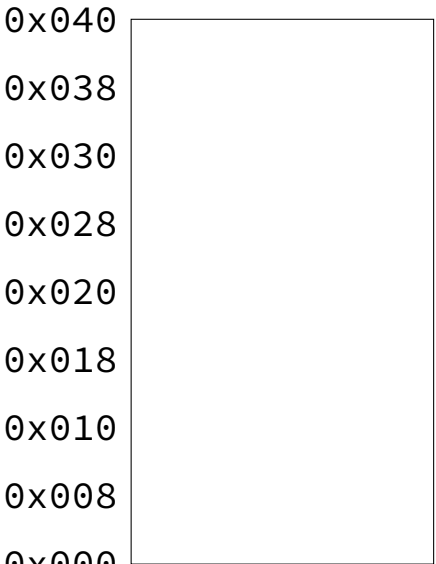
review CSO1 stuff

- labs 9–12 (of last Fall)
  `https://researcher111.github.io/uva-cso1-F23-DG/`

exercises we've used in the past:

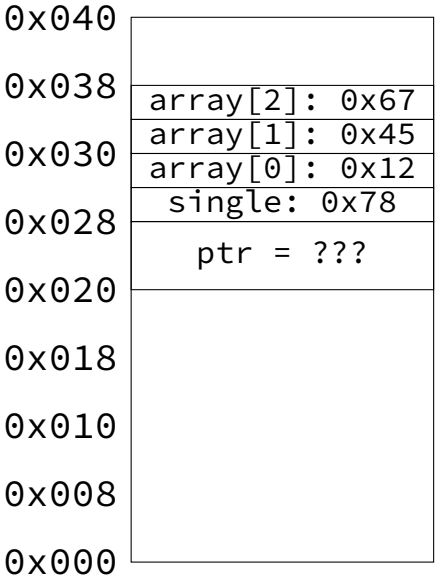
- implement strsep library function
- implement conversion from dynamic array to linked list

some pointer stuff



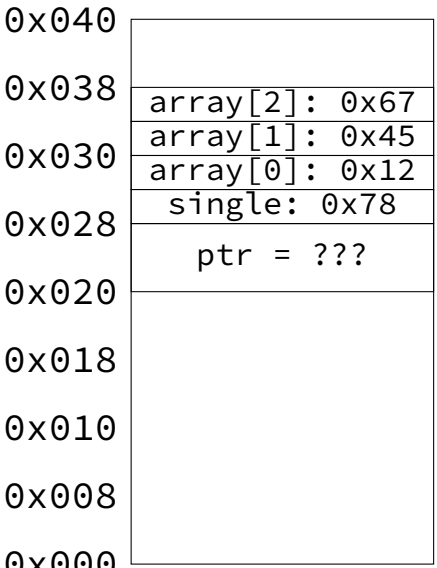

```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;
```

some pointer stuff




```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;
```

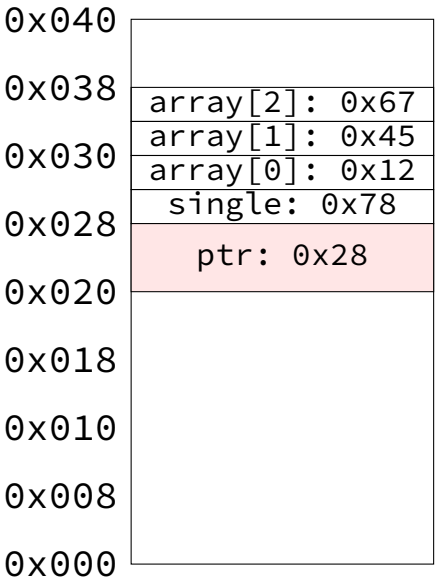
some pointer stuff




```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;
```

~~`*ptr = 0xAB;`~~ compile error

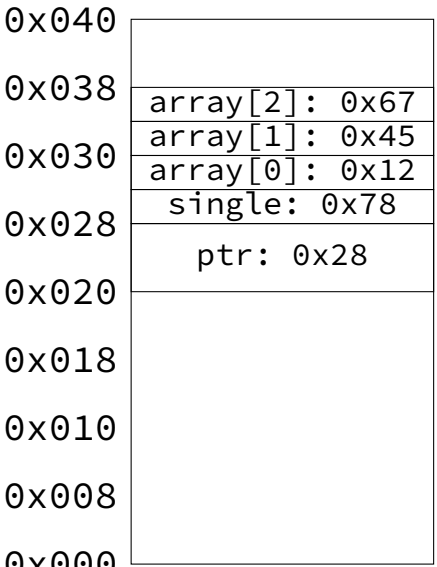
some pointer stuff




```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;

ptr = &single;
ptr = (int*) 0x28;
```

addr. of single

some pointer stuff



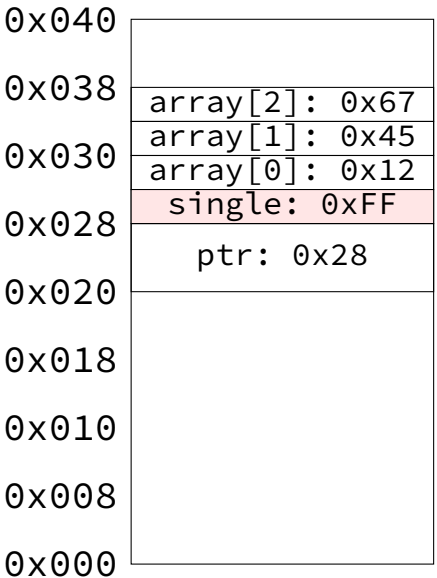

```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;

ptr = &single;
ptr = (int*) 0x28;
```
addr. of single

~~`ptr = 0x28;`~~ compile error

~~`ptr = (int*) single;`~~

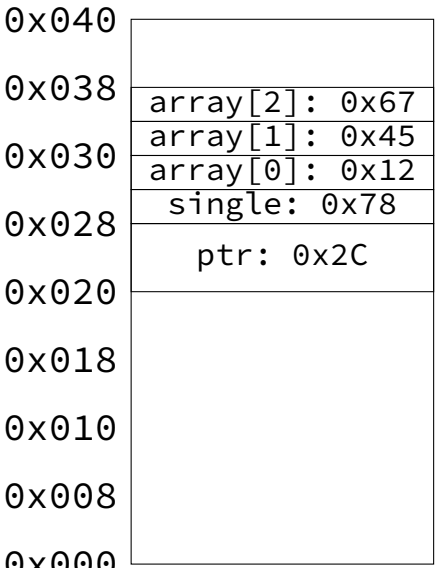
pointer to unknown place

some pointer stuff



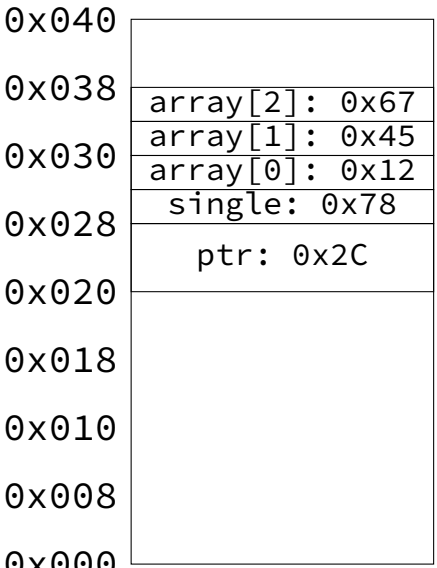

```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;
ptr = &single;

*ptr = 0xFF;
```

some pointer stuff




```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;

ptr = array;
ptr = &array[0];
ptr = (int*) 0x2C;
```

some pointer stuff



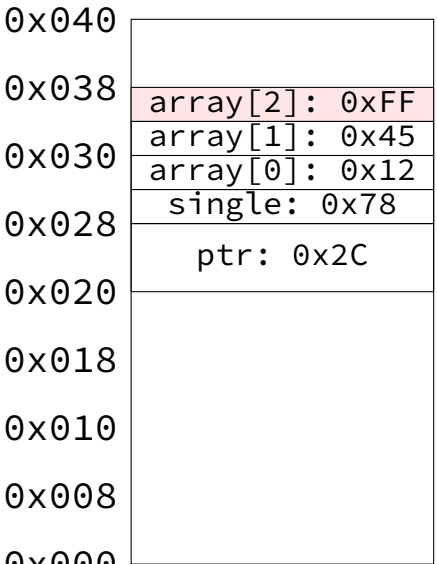

```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;

ptr = array;
ptr = &array[0];
ptr = (int*) 0x2C;
```

~~`ptr = array[0];`~~ compile error

~~`ptr = (int*) array[0];`~~

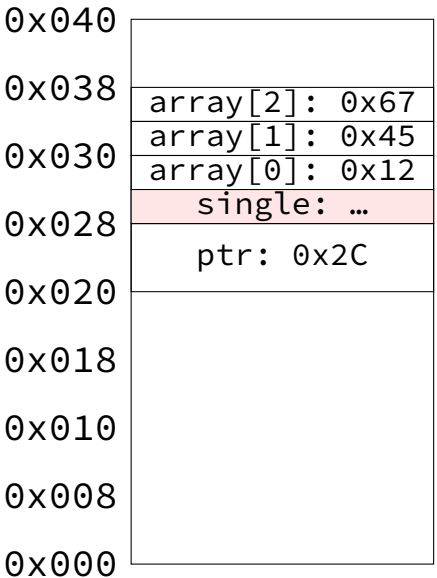
pointer to unknown place

some pointer stuff




```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;
ptr = &array[0];

ptr[2] = 0xFF;
*(ptr + 2) = 0xFF;

int *temp1; temp1 = ptr + 2;
*temp1 = 0xFF;

int *temp2; temp2 = &ptr[2];
*temp2 = 0xFF;
```

some pointer stuff




```
int array[3]={0x12,0x45,0x67};
int single = 0x78;
int *ptr;

void change_arg(int *x) {
    *x = compute_some_value();
}
…
change_arg(&single);
```

waitlists

2p, 3:30p heavily limited by room capacity!

if you are on that waitlist, suggest changing to 5p/6:30p

(changed after lecture:)

will increase course capacity soon

- as of Thursday lecture, limited by lecture capacity

make sure you don't have non-capacity restriction

- e.g. credit hour limit, time conflict

labs

attend lab in person and get checked off by TA, *or*

(most labs) submit something to submission site and we'll grade it

- submit to submission site? don't care if you attend the lab
- more strict about submissions without checkoffs
- in-person lab checkoff of incomplete lab at least 50% credit

some labs will basically require attendance

- or contact me for other arrangements if you can't (sick, etc.)
- logistically won't work otherwise — e.g. code review

if can't make lab in-person (example: sick)

- let me know, can arrange late/alternate checkoff

lab collaboration and submissions

please collaborate on labs!

when working with others on lab and submitting code files

please indicate who you worked with in those files

- via comment or similar

lab space

if labs are full, might kick out students from ‘wrong’ lab section

homeworks

several homework assignments

done individually

generally due on Fridays

(tentative dates on schedule)

homework/lab automatic testing

some homeworks/labs have automatic testing

with some delay after you submit

- usually 10s of minutes
- depending on assignment, number of submissions in queue
- if you submit very early, testing program might not be setup yet

when testing program doesn't understand/can't test something, left for manual grading ("not yet graded")

intention is that testing results are not surprises
if you did some manual testing (no hidden requirements, etc.)

if you think testing program made a mistake,
please submit regrade request

warmup assignment

first homework

due Friday 2 Feb

write C function to split a string into array of strings

- with dynamic memory allocation

write C program to call function using input/command-line arguments

write Makefile for it (next topic, next week's lab)

quizzes

released evening after Thursday lecture

- starting *next* week

due 15 minutes before lecture on Tuesdays

about lecture and/or lab from the prior week

5–6ish questions

individual, open book, open notes, open Internet

quizzes and work/comments

quizzes will have place for comments/work

will be used to do grading

delay: about 1 week after quiz is due

please use so we can give partial credit

if you find possible error in quiz question
please make your best guess about was meant
and explain what you did in the comemnts

on help on quiz questions

I and the TAs won't answer quiz questions...

but we will answer questions about the lecture material, etc.

(and TAs (not you) are responsible for knowing
what they can't answer
but we'd prefer you don't try to test those limits)

going over past quizzes

have in past gone over quiz Qs in lecture

- either when a lot missed it or
- on request in lecture

also fine office hour/Piazza question

readings

in lieu of textbook, have readings

mostly written by Prof Tychnoveich (now at UIUC) with edits by me

on website; should be indicated with corresponding lecture

readings often link to alternative/supplemental readings on topic

lecture + assignment sync

generally:
quiz after lecture and/or lab coverage
labs after lecture coverage
homework after lab coverage

means homework (and sometimes quiz)
may be relatively delayed from lecture coverage

exams

1 final exam

- likely in-person
- see official exam schedule

no midterms — instead:

- quizzes count a lot

development enviroment

we will test via something like SSH into portal

- officially supported environment

no restrictions re: IDEs

- but make sure you test/know how to run from command line

many students had success with VSCode + its SSH support

some notes on VSCode

I don't use VSCode (I use vim via SSH+tmux...)

but many of our TAs do; their advice:...

use SSH support to run on portal (dept machine)

- tutorial in last semester's CS 2130 lab (linked off main course website)

install Microsoft's C/C++ extension

- set C standard in settings as 'gnu17' or similar

install Microsoft's Makefile Tools extension

getting help

office hours — calendar will be posted on website

- mix of in-person and remote, indicated on calendar
- remote OH will use Discord + online queue
- in-person OH may or may not — indicated on whiteboard, probably

Piazza

- use private questions if homework code, etc.

emailing me (preferably with '3130' in subject)

collaboration (1)

labs — you can/should work with other students

- everyone should understand the work submitted
- we may ask questions/etc. to check on occassion

homeworks — individual

- write your own code / do not share your code
- can ask/look up *conceptual* questions of others
- others includes other students, Q&A sites, code generation tools, etc.
- **cite** any sources you use (comments in code)

collaboration (2)

quizzes — individual

but open book+notes+etc.

can/should have help reviewing lecture/readings/etc.

- legitimate questions for office hours

don't ask other students, stack overflow, gen AI tools, etc. the quiz questions

don't try to find exactly the quiz question on stack overflow

feedback

anonymous feedback on Canvas

would appreciate feedback (esp. when I can do something)

(but not a good way to ask for regrades, etc.)

late policy

no late quizzes

one quiz dropped (unconditionally)

90% credit for 0–72 hours late homeworks

for labs that allow submission only

- lab submission due time is 11:59am the next day
- 90% credit for 0–24 hours late

no late lab checkoffs except by special arrangement

excused lateness

special circumstances?

- illness, emergency, etc.

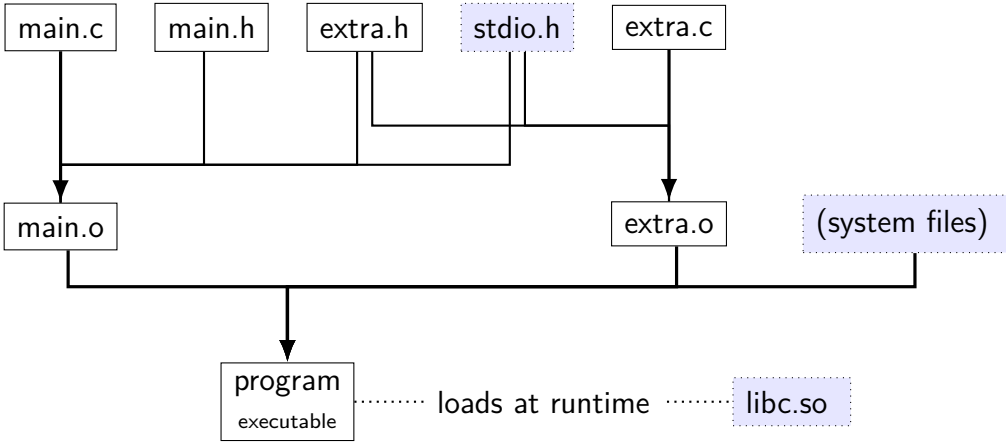
contact me, we'll figure something out

please don't attend lab/etc. sick!

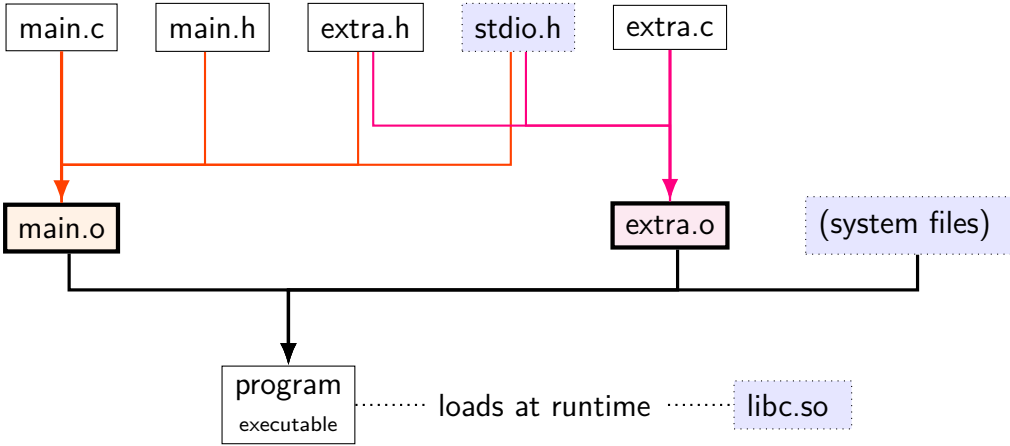
attendance

- I won't take attendance in lecture
- I will attempt to have lecture recordings
  - sometimes there may be issues with the recording

files in building C programs [dynamic linking]

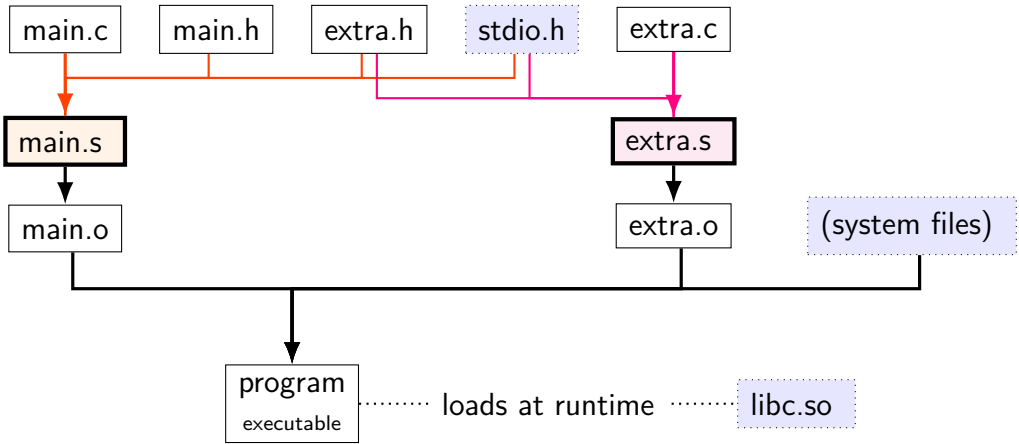


files in building C programs [dynamic linking]



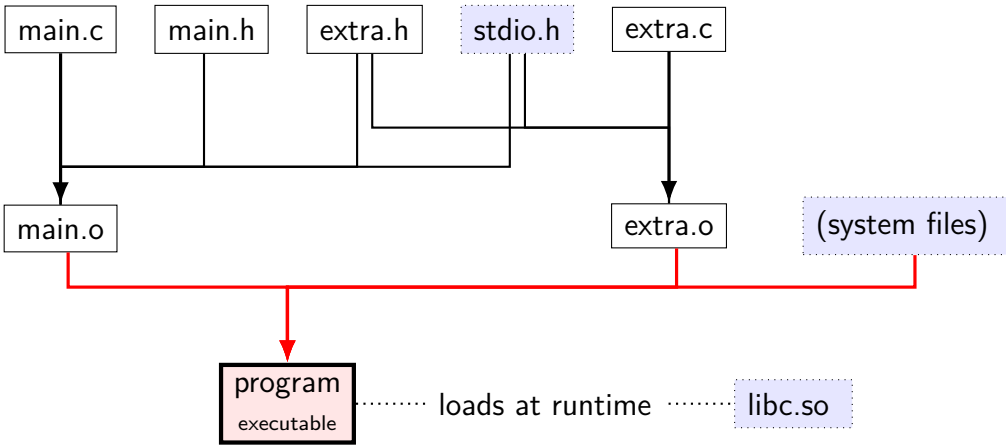

```
clang -c main.c
clang -c extra.c
```

files in building C programs [dynamic linking]



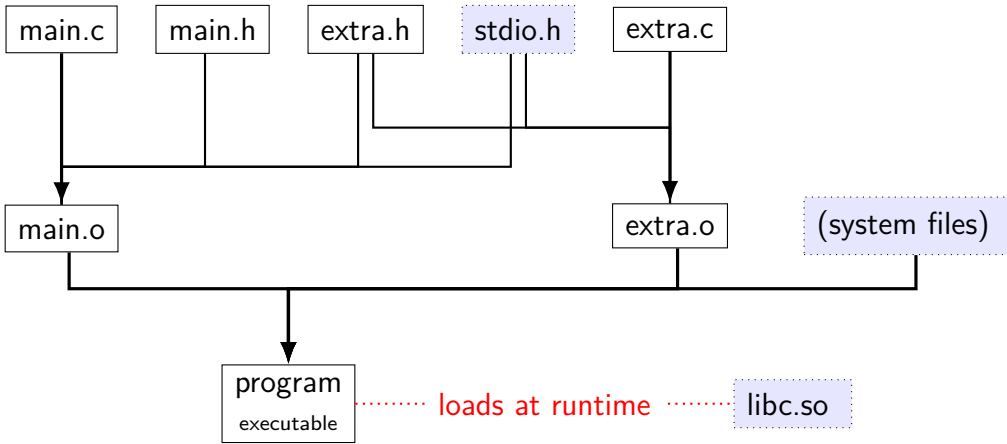

```
clang -S -c main.c
clang -S -c extra.c
```

files in building C programs [dynamic linking]



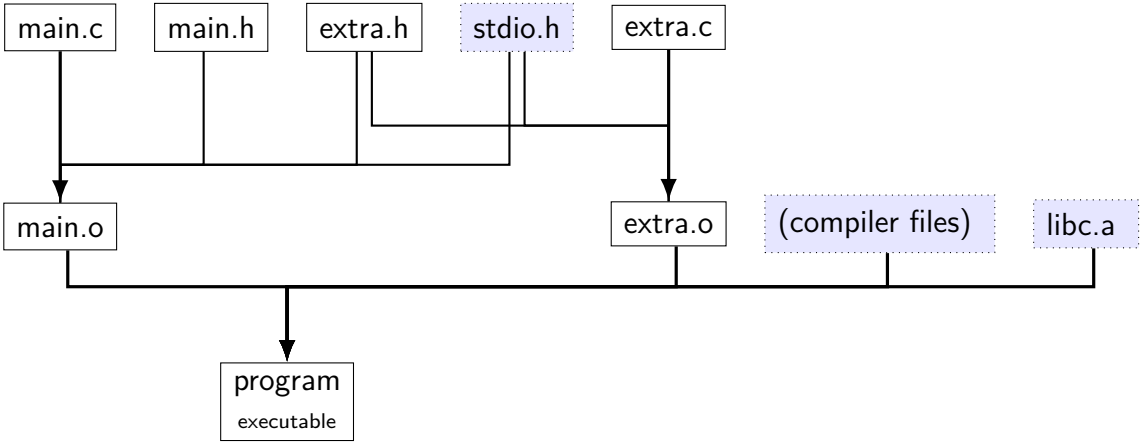

```
clang -o program main.o extra.o
```

files in building C programs [dynamic linking]




```
./program ...
```

files in building C programs [static linking]



file extensions

name		
`.c`		C source code
`.h`		C header file
`.s`	(or `.asm`)	assembly file
`.o`	(or `.obj`)	object file (binary of assembly)
(none)	(or `.exe`)	executable file
.a	(or `.lib`)	statically linked library [collection of .o files]
.so	(or `.dll` or `.dylib`)	dynamically linked library ['shared object']