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 on networks

under the hood of fast processors

- caching and (hidden) parallelism

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 on networks

under the hood of fast processors

- caching and (hidden) parallelism

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 libreadline.a foo.o foo-utility.o
$
```

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

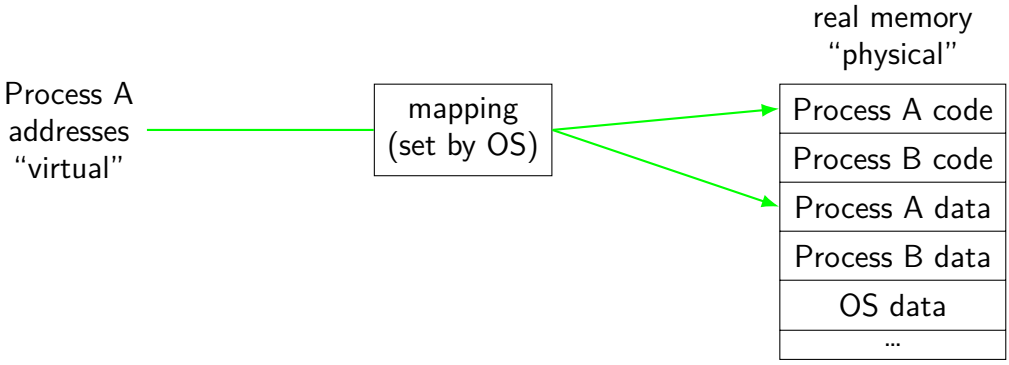
under the hood of sockets

- layered design on networks

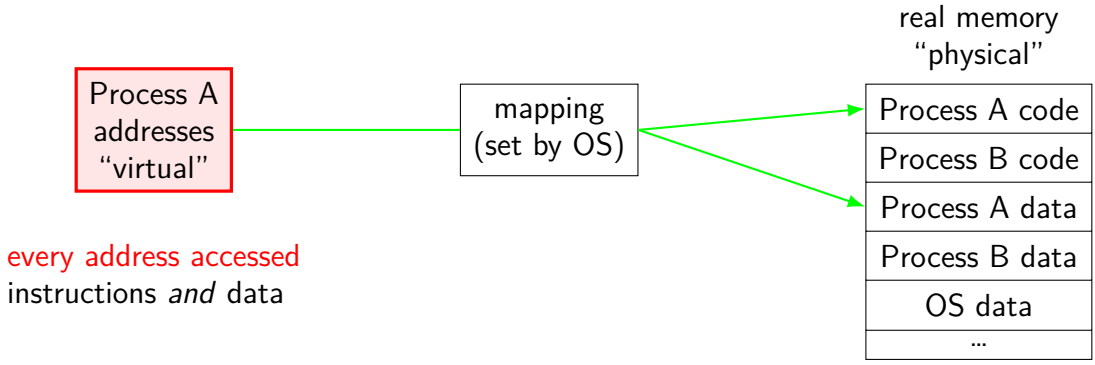
under the hood of fast processors

- caching and (hidden) parallelism

address translation

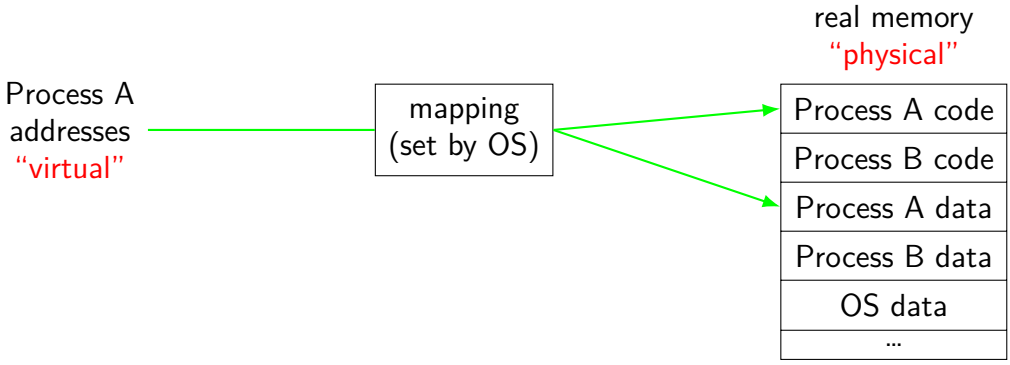


address translation



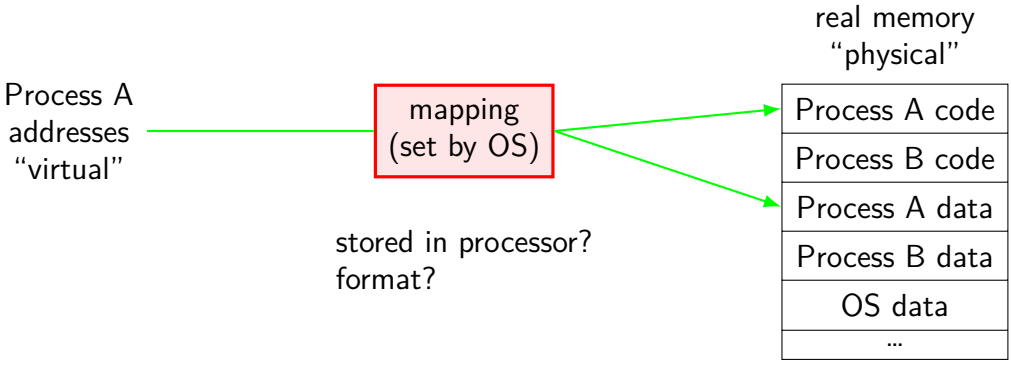
every address accessed
instructions *and* data

address translation



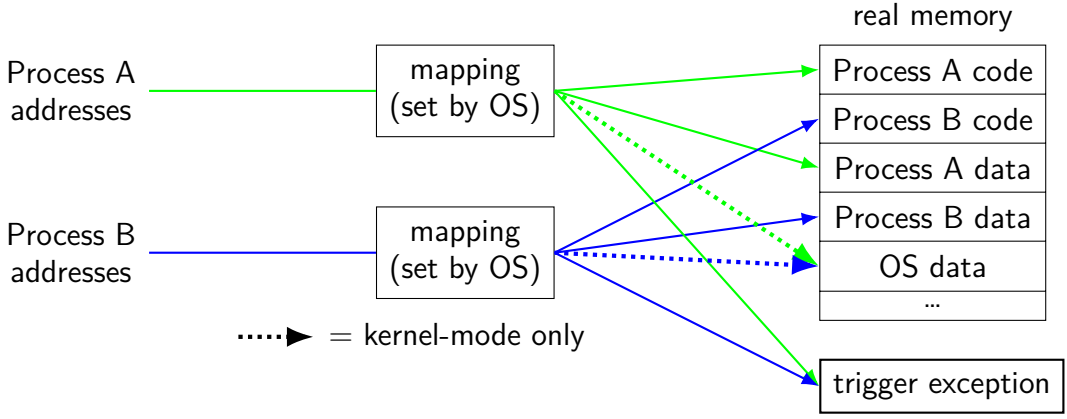
program addresses are 'virtual'
real addresses are 'physical'
can be different sizes!

address translation



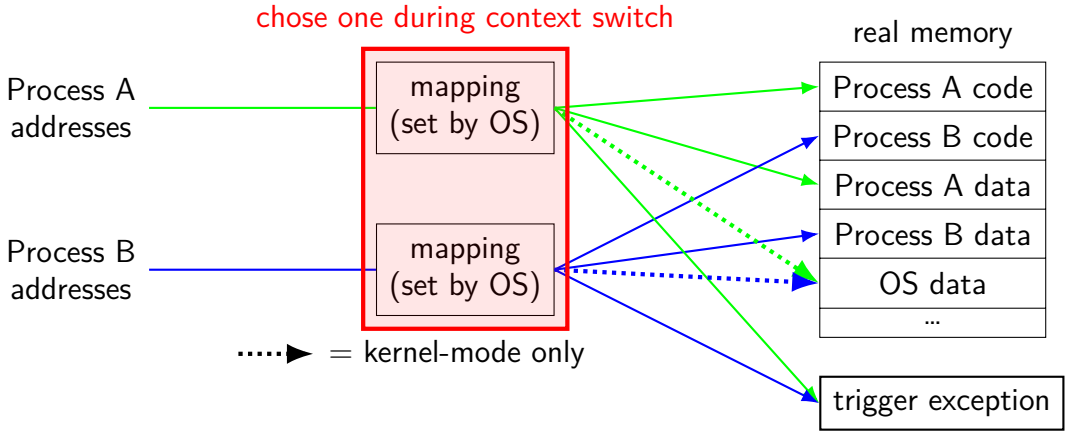
address spaces

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

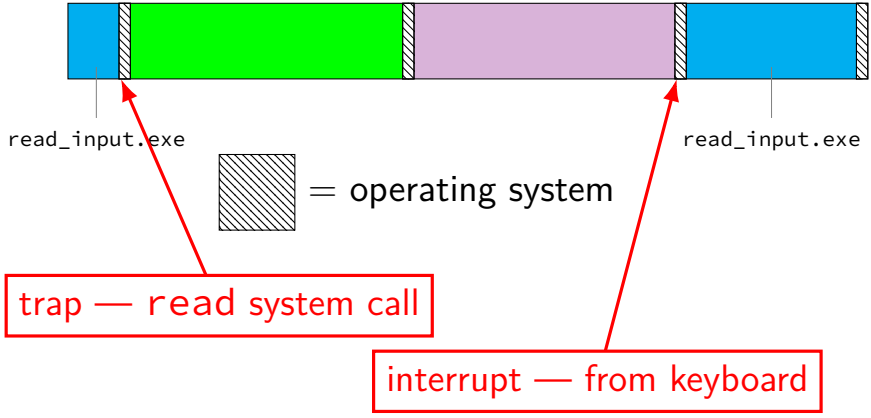
under the hood of sockets

- layered design on networks

under the hood of fast processors

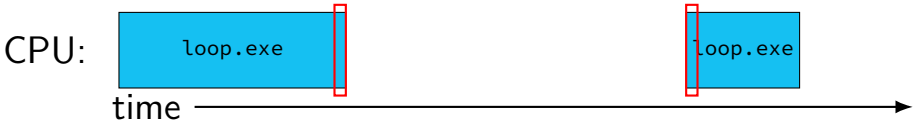
- caching and (hidden) parallelism

keyboard input timeline



time multiplexing

time multiplexing

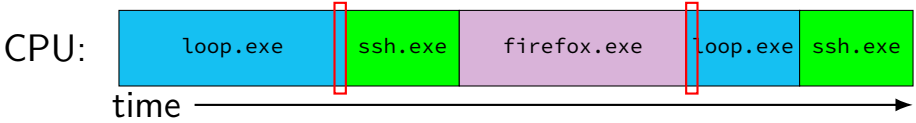



```
...
call get_time
    // whatever get_time does
movq %rax, %rbp
```

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
...
```

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 on networks

under the hood of fast processors

- caching and (hidden) parallelism

permissions

```
$ ls /u/other/secret
ls: cannot open directory '/u/other/secret': Permis
$ 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 on networks

under the hood of fast processors

- caching and (hidden) parallelism

layers

application	HTTP, SSH, SMTP, ...	application-defined meanings
transport	TCP, UDP, ...	reach correct program, reliablity/streams (sometimes)
network	IPv4, IPv6, ...	reach correct machine
link	Ethernet, Wi-Fi, ...	travel over wires/radio

more than four layers?

sometimes more layers above 'application'

e.g. HTTPS:
HTTP (app layer) on TLS (another app layer) on TCP (network) on ...

e.g. DNS over HTTPS:
DNS (app layer) on HTTP on on TLS on TCP on ...

e.g. SFTP:
SFTP (app layer??) on SSH (another app layer) on TCP on ...

e.g. HTTP over OpenVPN:
HTTP on TCP on IP on OpenVPN on UDP on different IP on ...

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`
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`
service name `https`	port number 443
service name `ssh`	port number 22

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 on networks

under the hood of fast processors

- caching and (hidden) parallelism

recall: data/instruction memory

model in CPU: one cycle per access

but earlier — had to talk to memory on different chip

can't do that in one cycle

solution: keep copies of part of memory ("cache")

- copy can be accessed quickly
- hope: almost always use copy?

2004 CPU

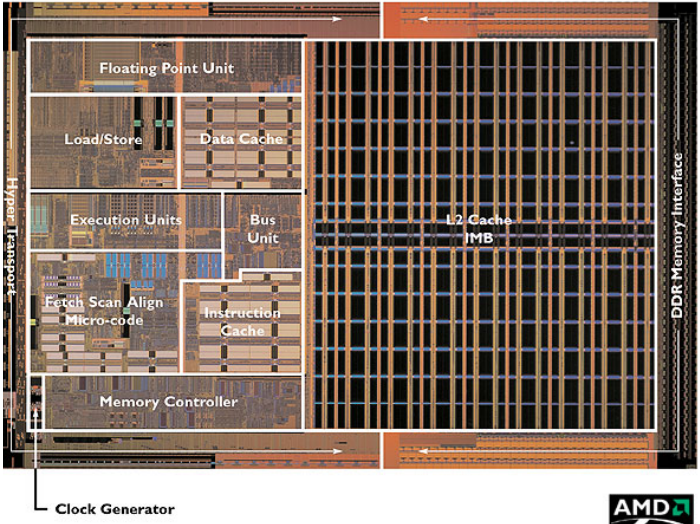
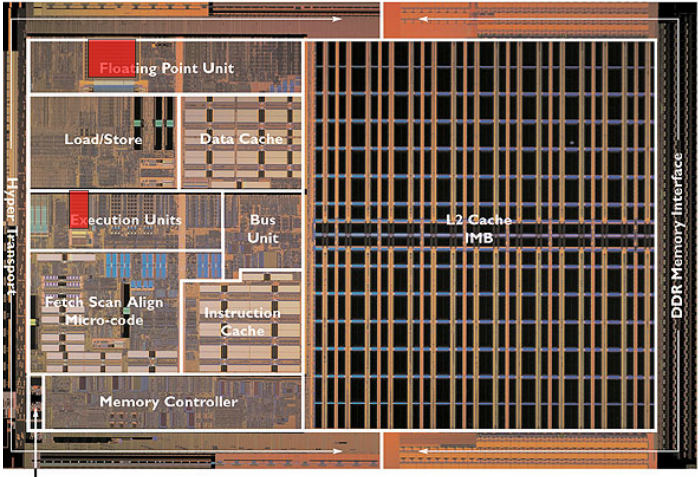


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



Registers

AMD 64 Opteron

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

2004 CPU

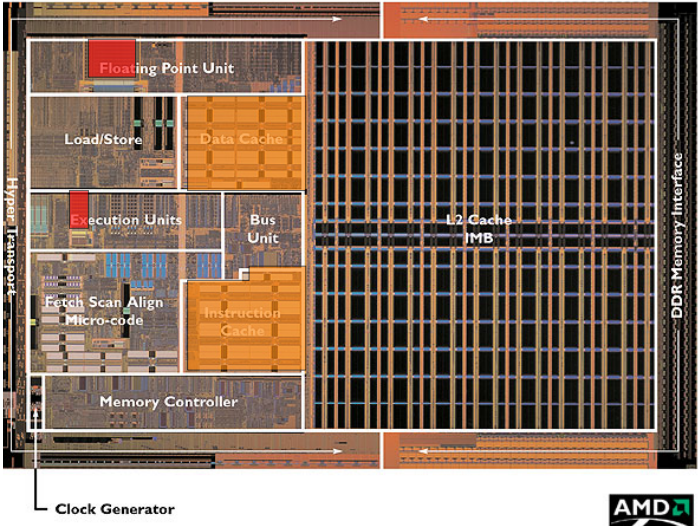
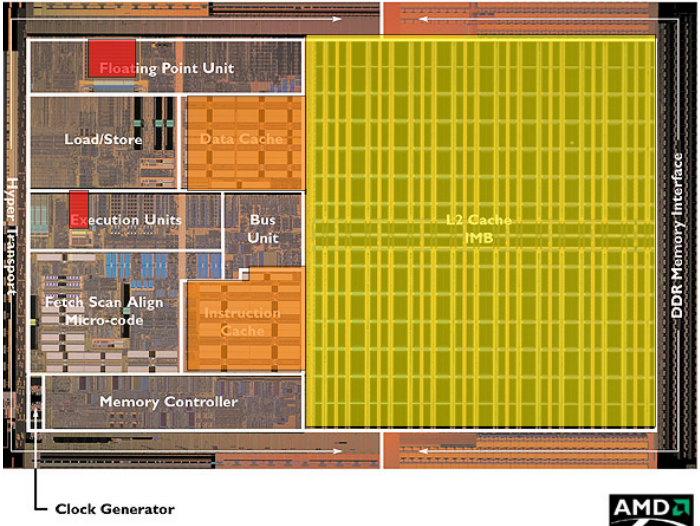


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



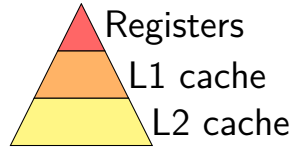
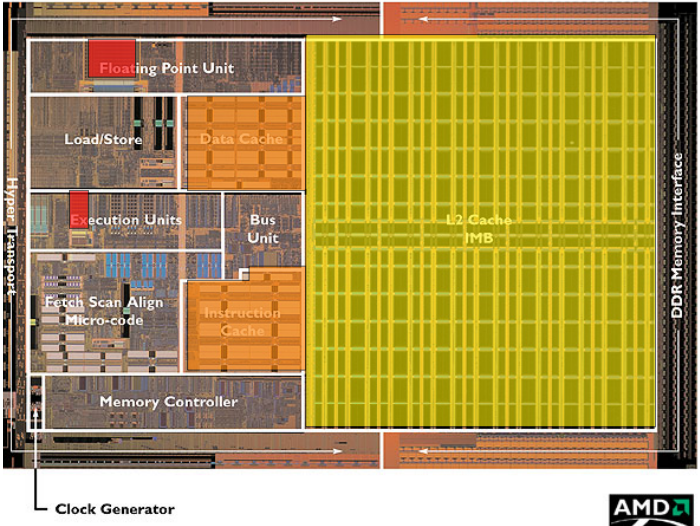


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



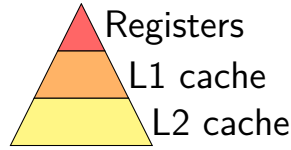
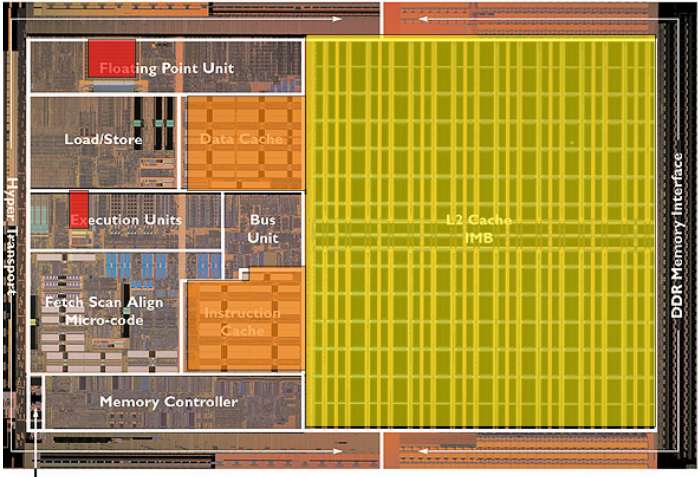


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



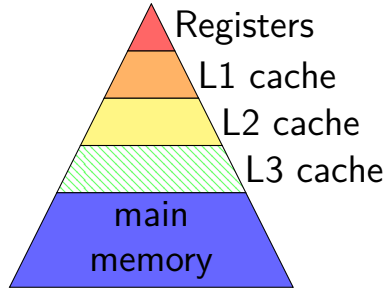
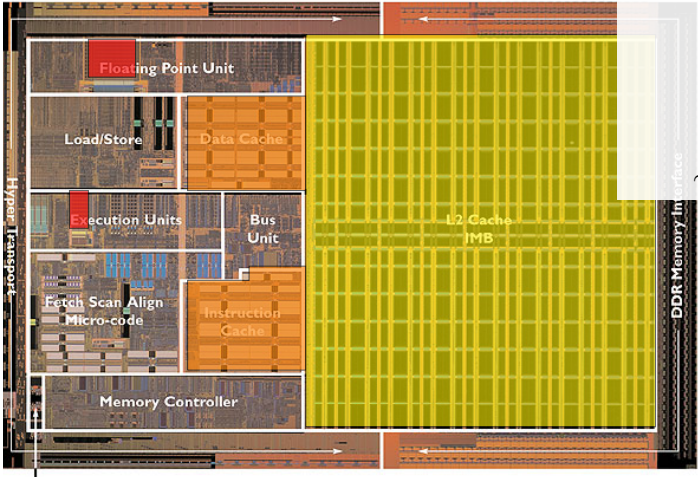


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



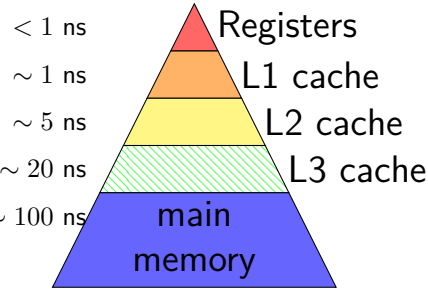
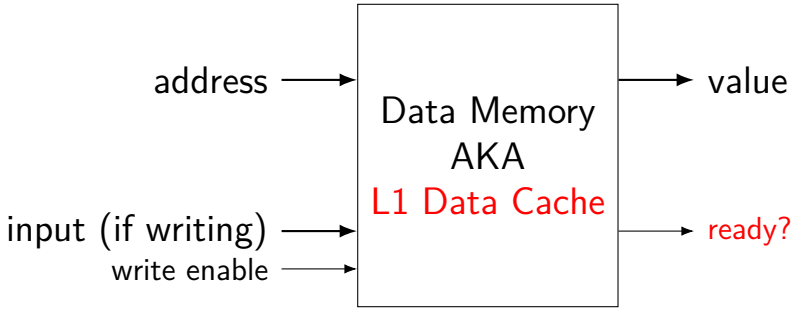
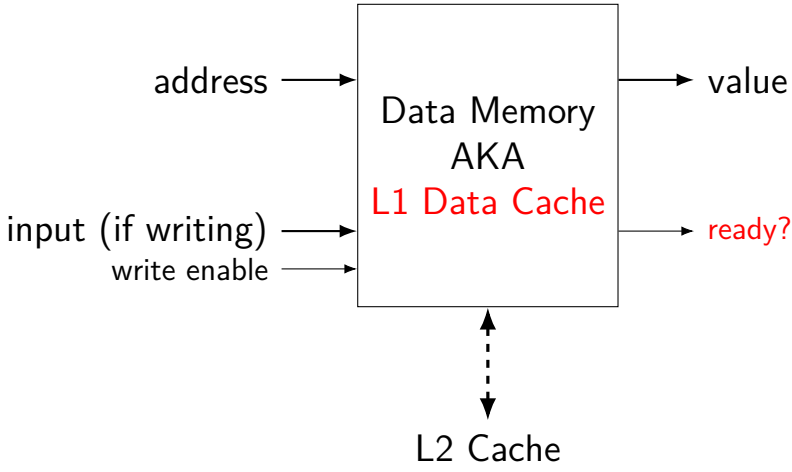


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

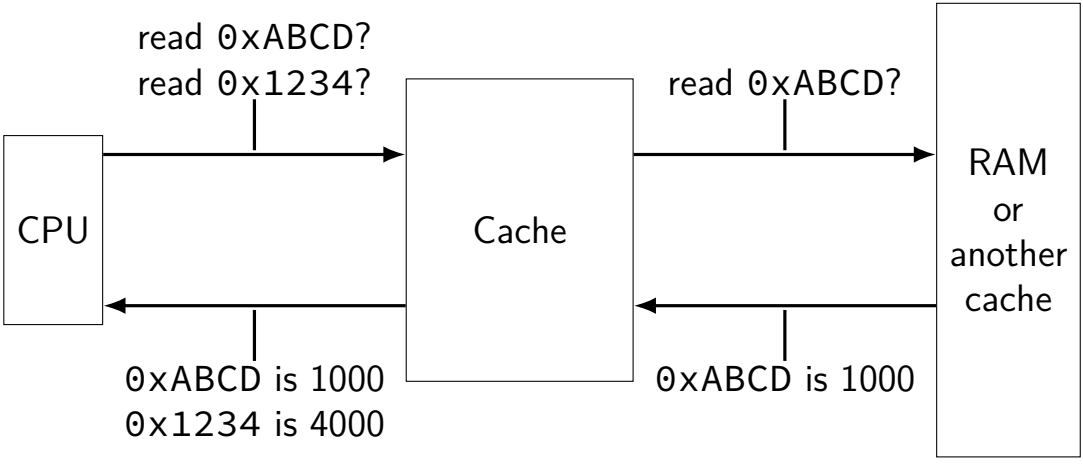
cache: real memory



cache: real memory



the place of cache



memory hierarchy goals

performance of the fastest (smallest) memory

- hide 100x latency difference? 99+% hit (= value found in cache) rate

capacity of the largest (slowest) 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 on networks

under the hood of fast processors

- caching and (hidden) parallelism

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 $10000000
loop2:
    addq %rbx, %rc
    addq %r8, %r9
    decq %rax
    jge loop2
    ret
```

about 40B instructio

my desktop: approx 2

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 $10000000
loop2:
    addq %rbx, %rc
    addq %r8, %r9
    decq %rax
    jge loop2
    ret
```

about 40B instructio

my desktop: approx 2

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 being complete/correct (can't tell how much time you actually spent)
- 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

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

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

4–6 questions

individual, open book, open notes, open Internet

- okay: looking up resources/tutorials/etc.
- not okay: asking Stack Overflow the quiz question
- not okay: IMing your friend the quiz question

asking about 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)

homeworks

several homework assignments

done individually

due before a week's first lab

exams

1 final exam

no midterms — instead:

- quizzes count a lot
- slightly more homework/lab than pilot

development enviroment

official: department machines via SSH or NX (remote desktop)

you can also use your own machines, but...

we will test your code on x86-64 Linux

I haven't checked assignments on a Windows or OS X machine

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)

late policy

no late quizzes

two quizzes dropped (unconditionally)

90% credit for 0–48 hours late homeworks

80% credit for 48–72 hours late homeworks

for labs that allow submission only, same policy as homeworks

- lab submission due time is 11:59pm

for other labs, policy on a lab-by-lab basis

excused lateness

special circumstances?

- illness, emergency, etc.

contact me, we'll figure something out

please don't attend lab/etc. sick!

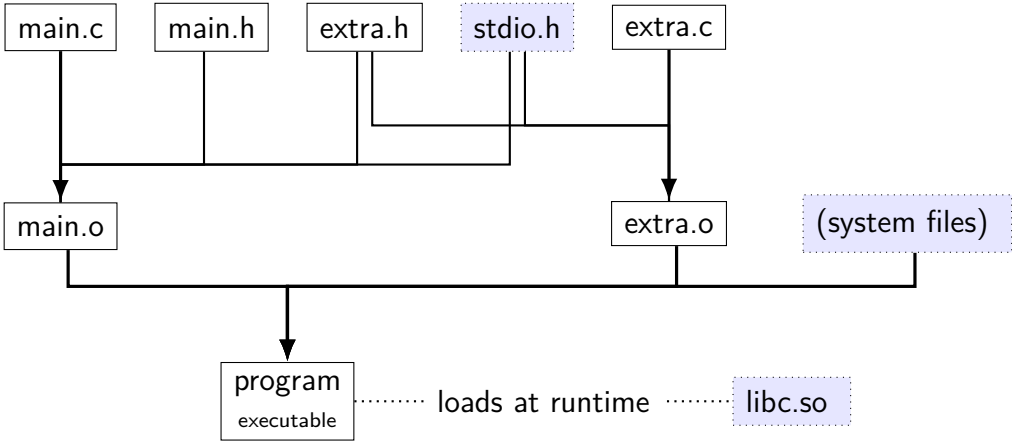
attendance

I won't take attendance in lecture

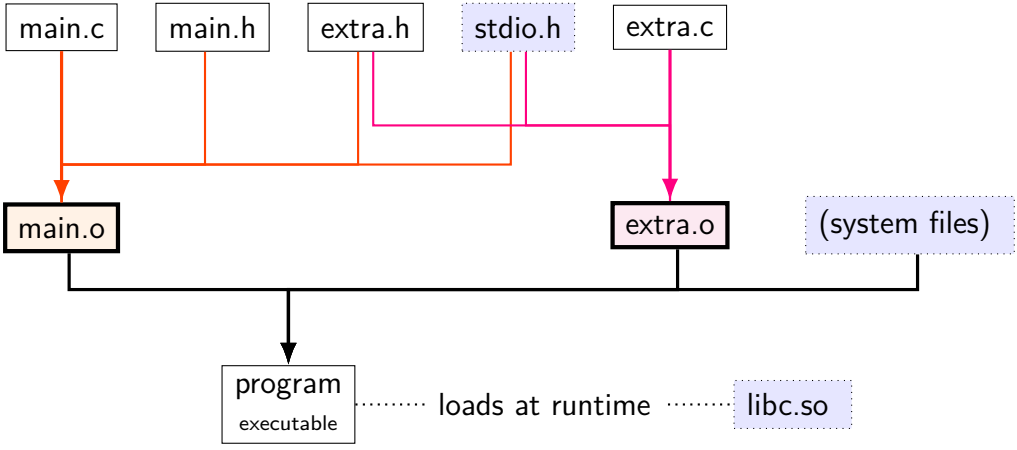
I will attempt to have lecture recordings

- sometimes there may be issues with the recording

files in building C programs [dynamic linking]

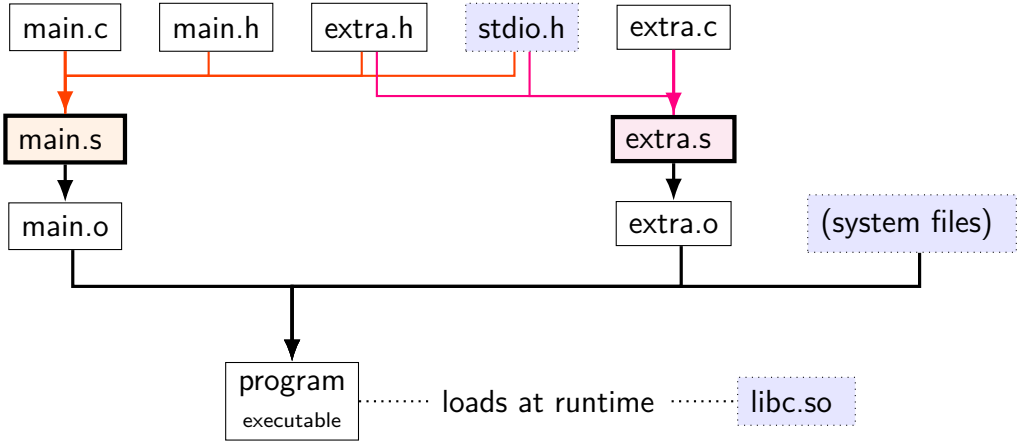


files in building C programs [dynamic linking]



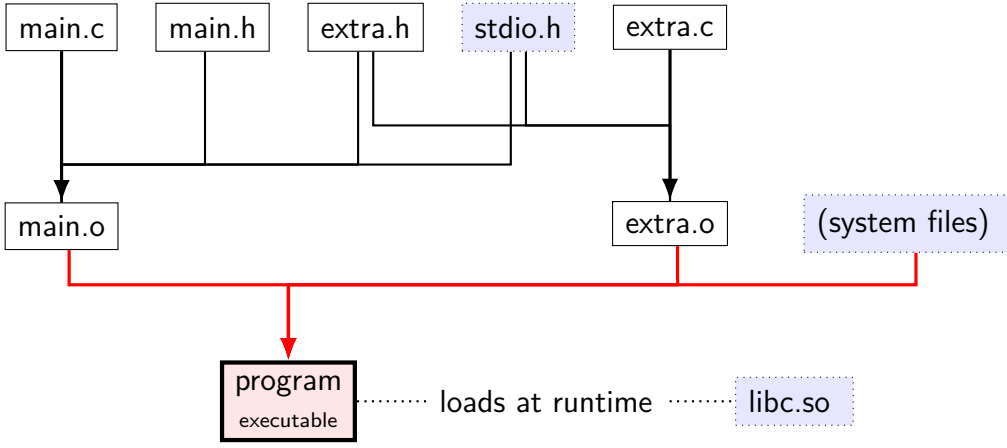

```
clang -c main.c
clang -c extra.c
```

files in building C programs [dynamic linking]



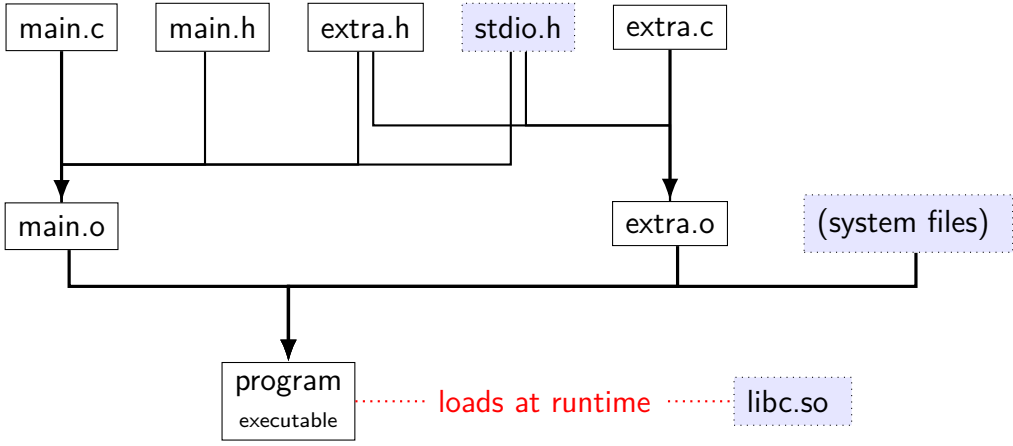

```
clang -S -c main.c
clang -S -c extra.c
```

files in building C programs [dynamic linking]



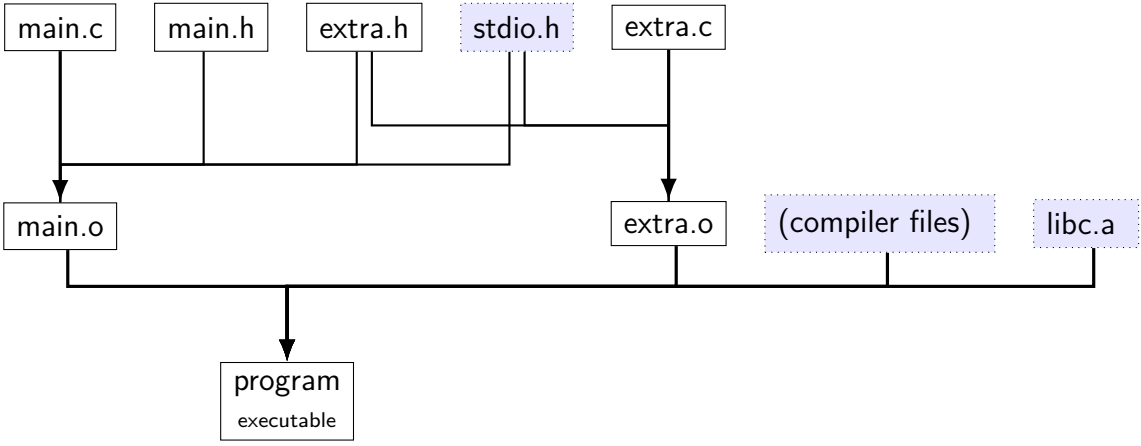

```
clang -o program main.o extra.o
```

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`)	dynamically linked library [‘shared object’]

exercise (incremental compilation)

program built from main.c + extra.c

main.c, extra.c both include extra.h, stdio.h

```
clang -c main.c                  # command 1
clang -c extra.c                 # command 2
clang -o program main.o extra.o # command 3
```

What commands need to be rerun if...

Question A: ...main.c changes?

Question B: ...extra.h changes?

make

`make` — Unix program for "making" things...

...by running commands based on what's changed

what commands? based on *rules* in *makefile*

make rules

```
main.o: main.c main.h extra.h
▶           clang -c main.c
```

before colon: target(s) (file(s) generated/updated)

after colon: prerequisite(s)

following lines prefixed by a tab character: command(s) to run

`make` will run the commands if any prerequisite is newer than the target

make rules

```
main.o: main.c main.h extra.h
▶           clang -c main.c
```

before colon: target(s) (file(s) generated/updated)

after colon: prerequisite(s)

following lines prefixed by a tab character: command(s) to run

make will run the commands if any prerequisite is newer than the target

make rules

```
main.o: main.c main.h extra.h
▶           clang -c main.c
```

before colon: target(s) (file(s) generated/updated)

after colon: prerequisite(s)

following lines prefixed by a tab character: command(s) to run

`make` will run the commands if any prerequisite is newer than the target

make rules

```
main.o: main.c main.h extra.h
▶           clang -c main.c
```

before colon: target(s) (file(s) generated/updated)

after colon: prerequisite(s)

following lines prefixed by a tab character: command(s) to run

make will run the commands if any prerequisite is newer than the target

make rules

```
main.o: main.c main.h extra.h
▶           clang -c main.c
```

before colon: target(s) (file(s) generated/updated)

after colon: prerequisite(s)

following lines prefixed by a tab character: command(s) to run

`make` will run the commands if any prerequisite is newer than the target

make rules

```
main.o: main.c main.h extra.h
▶           clang -c main.c
```

before colon: target(s) (file(s) generated/updated)

after colon: prerequisite(s)

following lines prefixed by a tab character: command(s) to run

make will run the commands if any prerequisite is newer than the target

make rules

```
main.o: main.c main.h extra.h
▶           clang -c main.c
```

before colon: target(s) (file(s) generated/updated)

after colon: prerequisite(s)

following lines prefixed by a tab character: command(s) to run

`make` will run the commands if any prerequisite is newer than the target

...after making sure prerequisites up to date

make rule chains

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

extra.o: extra.c extra.h
▶           clang -c extra.c
main.o: main.c main.h extra.h

▶           clang -c main.c
```

to *make* `program`, first...

update main.o and extra.o if they aren't

running make

"`make` *`target`*"

look in `Makefile` in current directory for rules
check if *`target`* is up-to-date
if not, rebuild it (and dependencies, if needed) so it is

"`make` *`target1 target2`*"

check if both target1 and target2 are up-to-date

"`make`"

if "*`firstTarget`*" is the first rule in Makefile,
same as '`make` *`firstTarget`*"

exercise: what will run?

```
W: X Y
►    buildW
X: Q
►    buildX
Y: X Z
►    buildY
```

W	modified 1 minute ago
X	modified 2 hours ago
Y	does not exist
Z	modified 1 hour ago
Q	modified 3 hours ago

exercise: "make W" will run what commands?

A. none
B. `buildY` only
C. `buildW` then `buildY`
D. `buildY` then `buildW`
E. `buildX` then `buildY` then `buildW`
F. `buildX` then `buildW`
G. something else

‘phony’ targets (1)

common to have Makefile targets that aren’t files

```
all: program1 program2 libfoo.a
```

“`make all`” effectively shorthand for “`make program1 program2 libfoo.a`”

no actual file called “all”

'phony' targets (2)

sometimes want targets that don't actually build file

example: "`make clean`" to remove generated files

```
clean:
▶	rm --force main.o extra.o
```

but what if I create...

```
clean:
▶	rm --force main.o extra.o

all: program1 program2 libfoo.a
```

Q: if I make a file called “all” and then “make all” what happens?

Q: same with “clean” and “make clean”?

marking phony targets

```
clean:
▶       rm --force main.o extra.o

all: program1 program2 libfoo.a

.PHONY: all clean
```

special .PHONY rule says " 'all' and 'clean' not real files"

(not required by POSIX, but in every make version I know)

conventional targets

common convention:

target name	purpose
(default), `all`	build everything
`install`	install to standard location
`test`	run tests
`clean`	remove generated files

redundancy (1)

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

extra.o: extra.c extra.h
▶        clang -o extra.o -c extra.c
main.o: main.c main.h extra.h

▶        clang -o main.o -c main.c
```

what if I want to run `clang` with `-Wall`?

what if I want to change to `gcc`?

variables (1)

```
CC = gcc
CFLAGS = -Wall
LDFLAGS = -Wall

program: main.o extra.o
▶       $(CC) $(LDFLAGS) -o program main.o extra.

extra.o: extra.c extra.h
▶       $(CC) $(CFLAGS) -o extra.o -c extra.c

main.o: main.c main.h extra.h
▶       $(CC) $(CFLAGS) -o main.o -c main.c
```

variables (2)

```
CC = gcc
CFLAGS = -Wall
LDFLAGS = -Wall

program: main.o extra.o
▶       $(CC) $(LDFLAGS) -o $@ $^

extra.o: extra.c extra.h
▶       $(CC) $(CFLAGS) -o $@ -c $<

main.o: main.c main.h extra.h
▶       $(CC) $(CFLAGS) -o $@ -c $<
```

aside: `$^` works on GNU make (usual on Linux), but not portable.

suffix rules

```
CC = gcc
CFLAGS = -Wall
LDFLAGS = -Wall

program: main.o extra.o
▶       $(CC) $(LDFLAGS) -o $@ $^

.c.o:
▶       $(CC) $(CFLAGS) -o $@ -c $<

extra.o: extra.c extra.h
main.o: main.c main.h extra.h
```

aside: `$^` works on GNU make (usual on Linux), but not portable.

pattern rules

```
CC = gcc
CFLAGS = -Wall
LDFLAGS = -Wall

program: main.o extra.o
▶       $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c
▶       $(CC) $(CFLAGS) -o $@ -c $<

extra.o: extra.c extra.h
main.o: main.c main.h extra.h
```

aside: these rules work on GNU make (usual on Linux), but less portable than suffix rules.

built-in rules

'make' has the 'make .o from .c' rule built-in already, so:

```
CC = gcc
CFLAGS = -Wall
LDFLAGS = -Wall

program: main.o extra.o
▶       $(CC) $(LDFLAGS) -o $@ $^

extra.o: extra.c extra.h
main.o: main.c main.h extra.h
```

(don't actually need to write supplied rule!)

writing Makefiles?

error-prone to automatically all .h dependencies

`-M` option to `gcc` or `clang`

- outputs Make rule
- ways of having make run this

Makefile generators

- other programs that write Makefiles

other build systems

alternatives to writing Makefiles:

other make-ish build systems

- ninja, scons, bazel, maven, xcodebuild, msbuild, ...

tools that generate inputs for make-ish build systems

- cmake, autotools, qmake, ...

backup slides