welcome!

to cs161 extended time discussion :)

slides
bit.ly/cs161-disc

feedback
bit.ly/extended-feedback

about me — abhi

about me — abhi

- abhi (he/him/his)

about me — abhi

- abhi (he/him/his)
- from st. louis, missouri

about me — abhi

- abhi (he/him/his)
- from st. louis, missouri
- love writing and film photography (recently)

about me — abhi

- abhi (he/him/his)
- from st. louis, missouri
- love writing and film photography (recently)
- i’m here to be your point of contact!
  - 1-hr disc: M/W 5-6pm Wheeler 20~~0~~2
  - abhiganesh@berkeley.edu

about you

- name, pronouns, major, year, anything
- where are you from?
- thoughts on cs61c/coding/CS
- misc (choose as many as you want)
    - favorite place to travel
    - songs you know every word to
    - favorite food
    - best places to visit in berkeley
    - hobbies

hack of the day

- mailchimp compromised via social engineering attacks on employees
  - mimicked the Okta authentication pages of the respective organizations
  - compromised “133 users' names, store URLs, addresses, and email addresses but not their payment data, passwords, or other sensitive information”

how?

general questions, concerns, etc.

security principles

1. know your threat model

the threat model

- who your attacker is
- what resources they have

the threat model

- who your attacker is
- what resources they have

---

common assumptions

- can interact with systems without notice
- knows operating systems, vulnerabilities in software, usually patterns of activity, etc.

common assumptions

common assumptions

- can interact with systems without notice

common assumptions

- can interact with systems without notice
- knows operating systems, vulnerabilities in software, usually patterns of activity, etc.

common assumptions

- can interact with systems without notice
- knows operating systems, vulnerabilities in software, usually patterns of activity, etc.
- has the resources required to mount the attack

common assumptions

- can interact with systems without notice
- knows operating systems, vulnerabilities in software, usually patterns of activity, etc.
- has the resources required to mount the attack
- can and will obtain privileges if possible

trusted computing base (TCB)

trusted computing base (TCB)

- the components that security relies upon

trusted computing base (TCB)

- the components that security relies upon
- properties:

trusted computing base (TCB)

- the components that security relies upon
- properties:
    - correctness

trusted computing base (TCB)

- the components that security relies upon
- properties:
    - correctness
    - completeness (can’t be bypassed)

trusted computing base (TCB)

- the components that security relies upon
- properties:
  - correctness
  - completeness (can’t be bypassed)
  - security (can’t be tampered with)

trusted computing base (TCB)

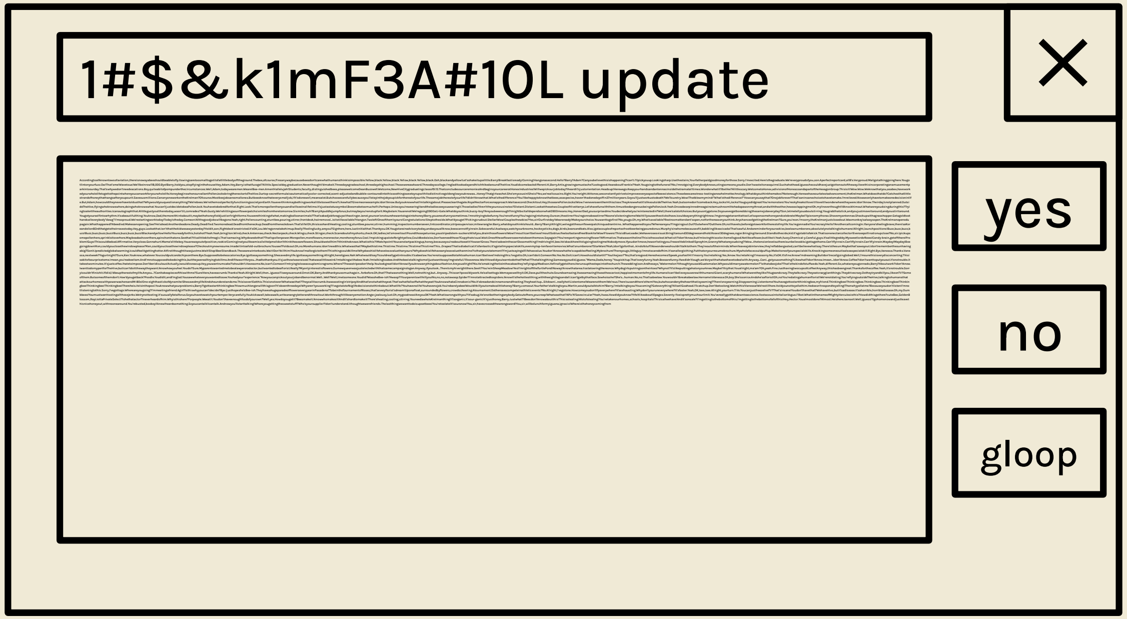
- the components that security relies upon
- properties:
  - correctness
  - completeness (can’t be bypassed)
  - security (can’t be tampered with)
- generally as small as possible (KISS)

security principles

2. consider human factors

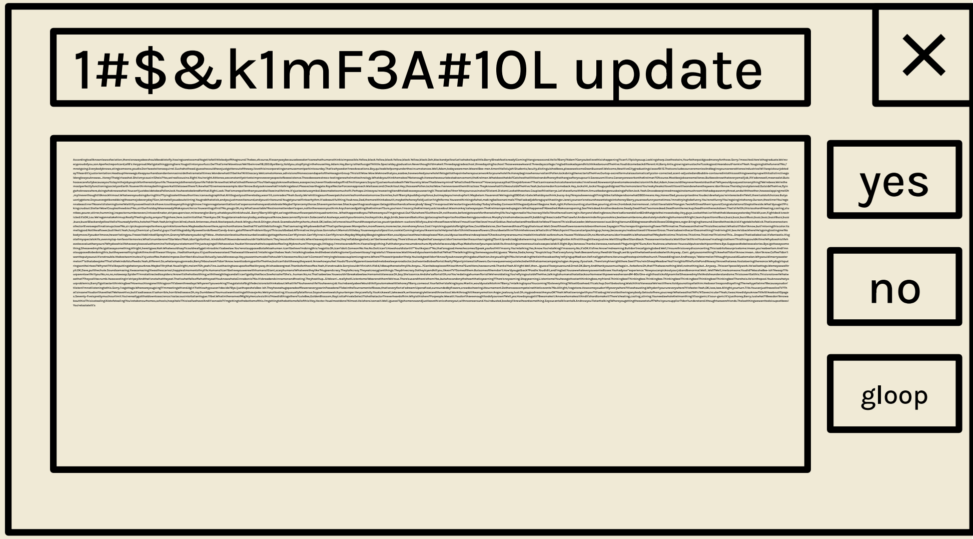
consider human factors

- you’ve designed the world’s best security system. here’s the dialog



consider human factors

- your security system should be *intuitive*
- ensure security is being used (as intended)
- user friendliness
  - prevent social engineering attacks



security principles

3. security is economics

security is economics

security is economics

- security is like a chain: as strong as the weakest link

security is economics

- security is like a chain: as strong as the weakest link
    - focus on the weakest link

security is economics

- security is like a chain: as strong as the weakest link
  - focus on the weakest link
- balance security vs resources

security principles

4-11: the rest cause too many slides

security principles

security principles

4. detect if you can’t prevent

security principles

4. detect if you can’t prevent
5. defense in depth (castle walls, a moat, etc.)

security principles

4. detect if you can’t prevent
5. defense in depth (castle walls, a moat, etc.)
6. least privilege (do i need to edit files?)

security principles

4. detect if you can’t prevent
5. defense in depth (castle walls, a moat, etc.)
6. least privilege (do i need to edit files?)
7. separation of responsibility (two people to nuke)

security principles

4. detect if you can’t prevent
5. defense in depth (castle walls, a moat, etc.)
6. least privilege (do i need to edit files?)
7. separation of responsibility (two people to nuke)
8. complete mediation (check ALL accesses)

security principles

4. detect if you can’t prevent
5. defense in depth (castle walls, a moat, etc.)
6. least privilege (do i need to edit files?)
7. separation of responsibility (two people to nuke)
8. complete mediation (check ALL accesses)
9. shannon’s maxim (security through obscurity 👎)

security principles

4. detect if you can’t prevent
5. defense in depth (castle walls, a moat, etc.)
6. least privilege (do i need to edit files?)
7. separation of responsibility (two people to nuke)
8. complete mediation (check ALL accesses)
9. shannon’s maxim (security through obscurity 👎)
10. use fail-safe defaults (doors lock when they fail)

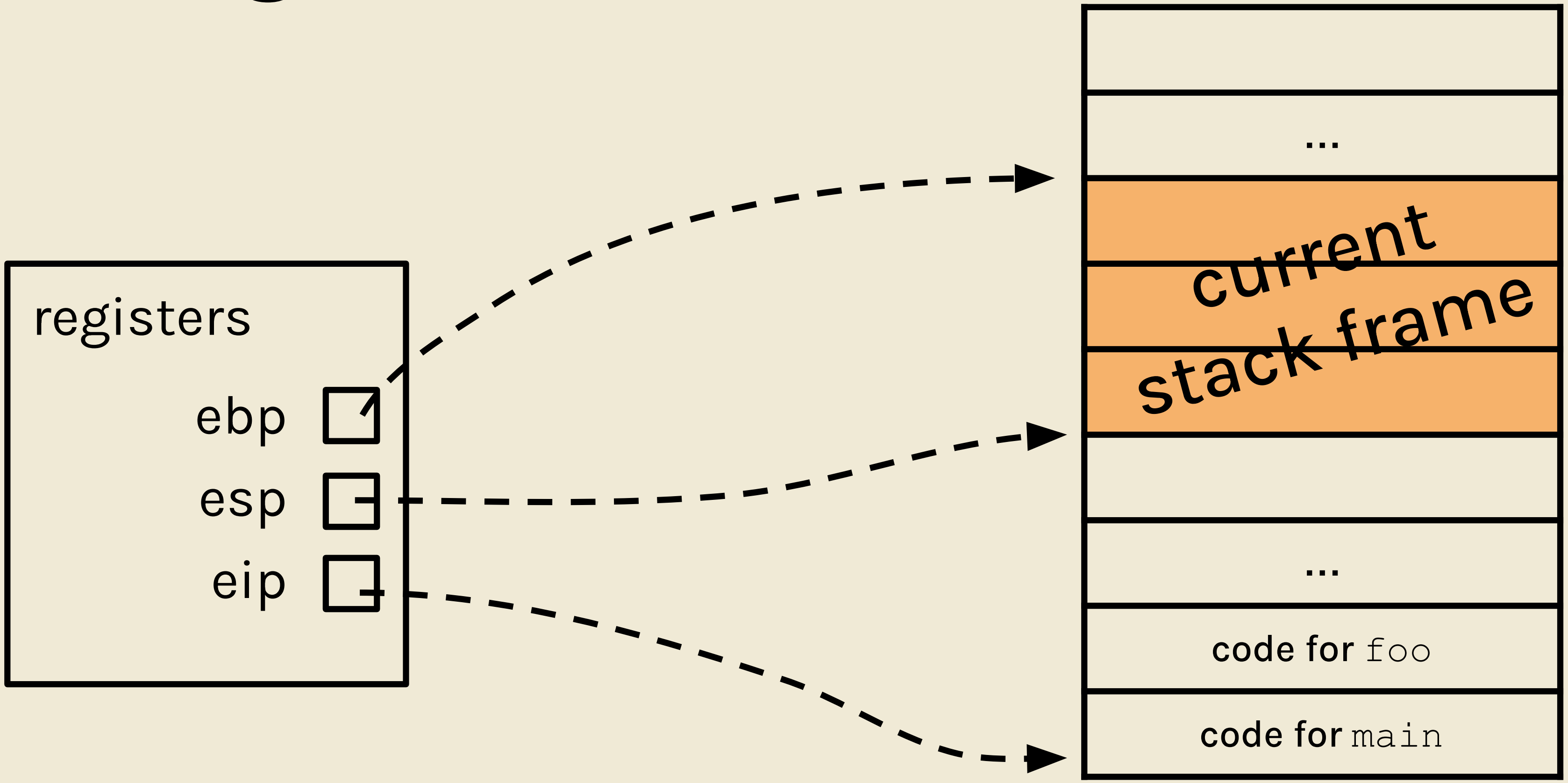
security principles

4. detect if you can’t prevent
5. defense in depth (castle walls, a moat, etc.)
6. least privilege (do i need to edit files?)
7. separation of responsibility (two people to nuke)
8. complete mediation (check ALL accesses)
9. shannon’s maxim (security through obscurity 👎)
10. use fail-safe defaults (doors lock when they fail)
11. design in security from the start

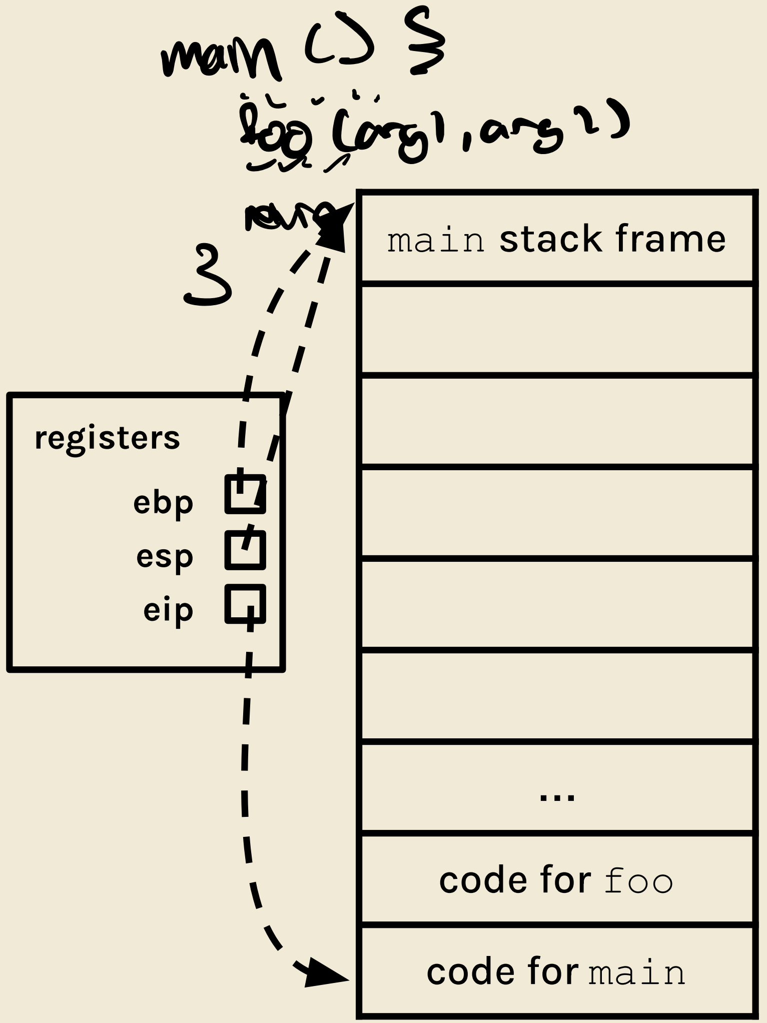
x86

no, it’s not RISC-V

the registers



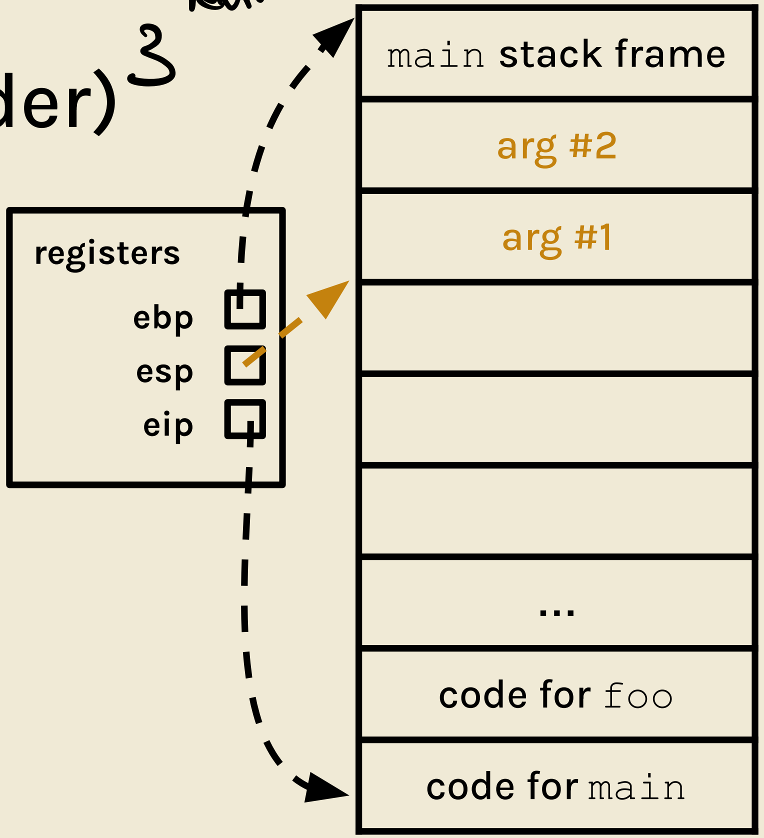
calling convention



calling convention

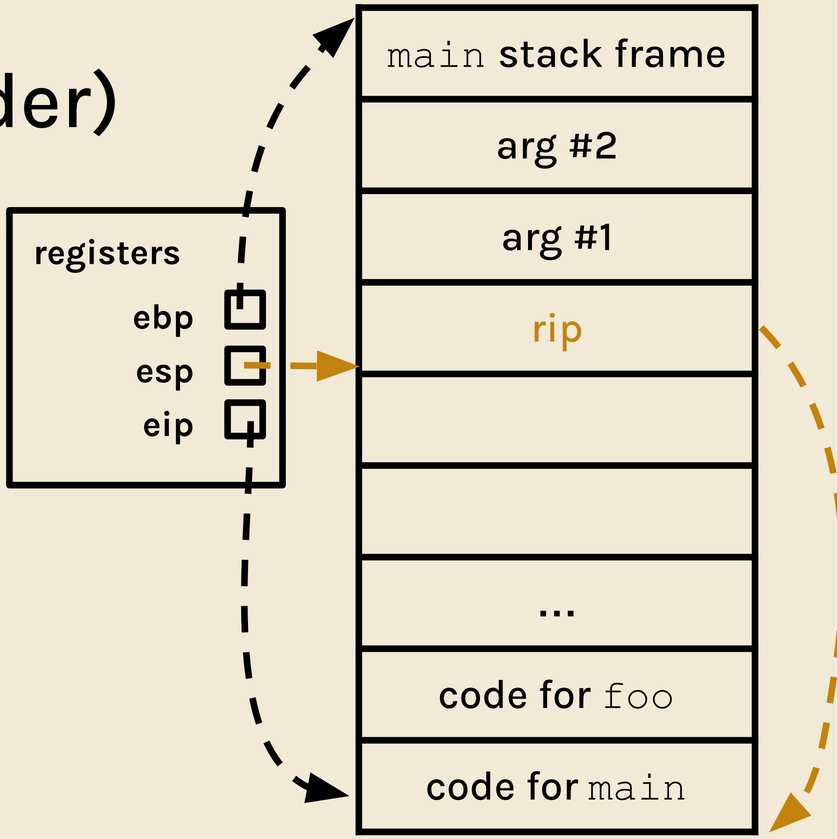
1. push arguments (reverse order)
   – adjust esp

```
main () {
    foo (arg1, arg2)
    ret
}
```



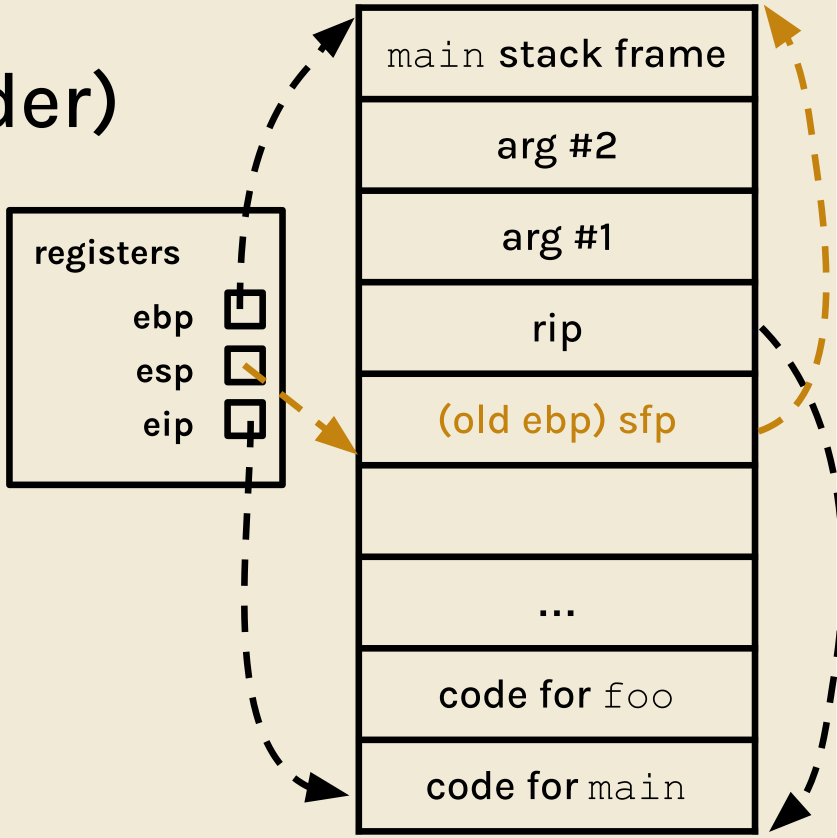
calling convention

1. push arguments (reverse order)
2. remember eip
   - like `ra` in RISC-V



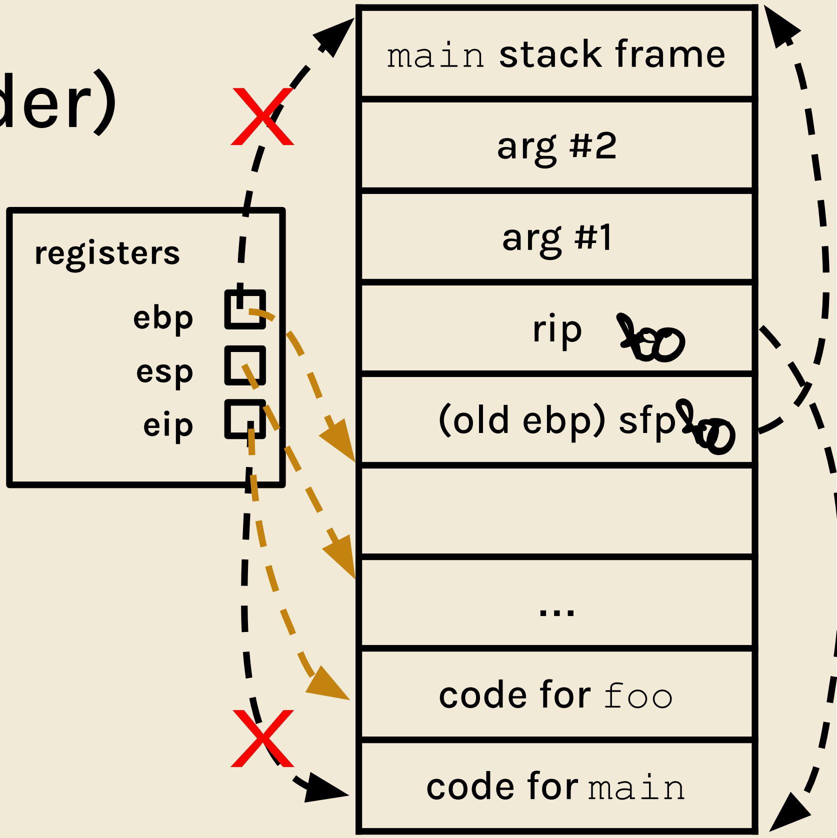
calling convention

1. push arguments (reverse order)
2. remember eip
3. remember ebp
   - to restore to top of previous stack frame



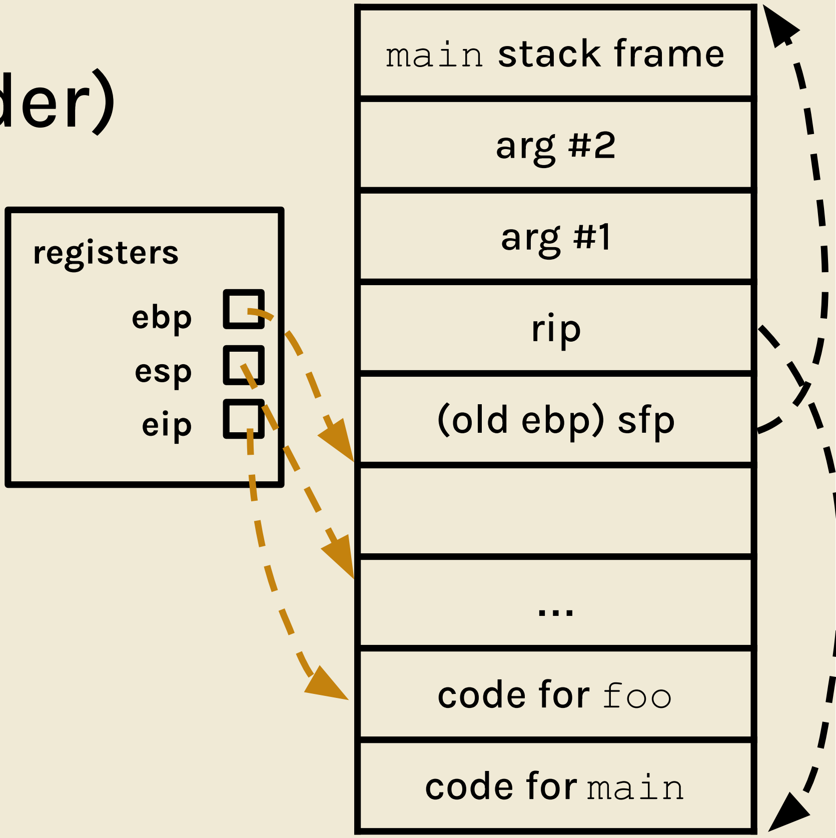
calling convention

1. push arguments (reverse order)
2. remember eip
3. remember ebp
4. adjust the stack frame
   - update ebp, esp, eip



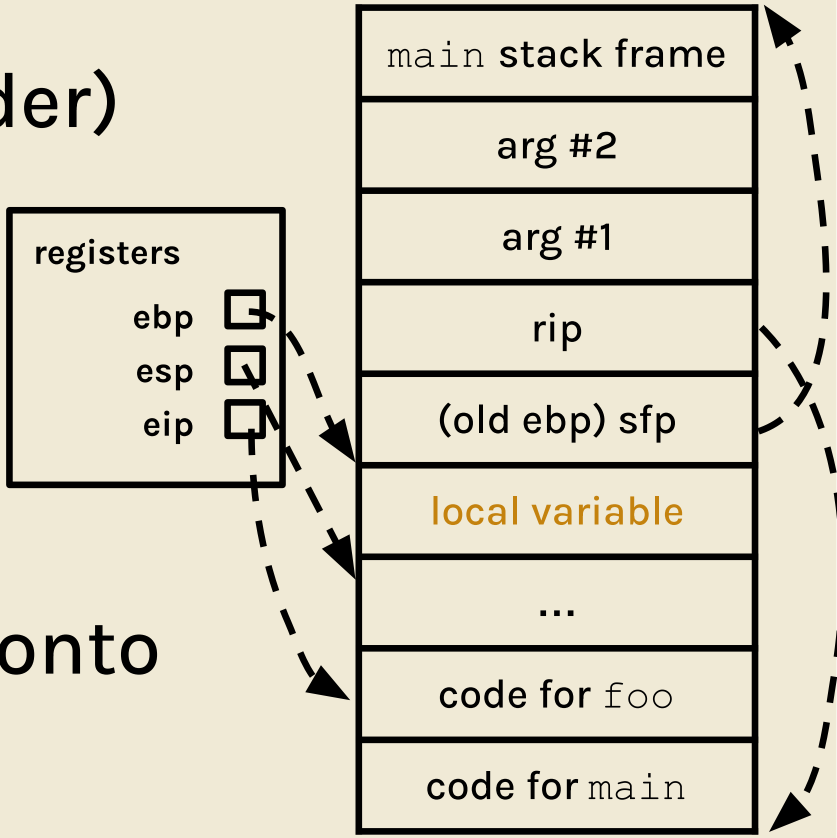
calling convention

1. push arguments (reverse order)
2. remember eip
3. remember ebp
4. adjust the stack frame
   - update ebp, esp, eip



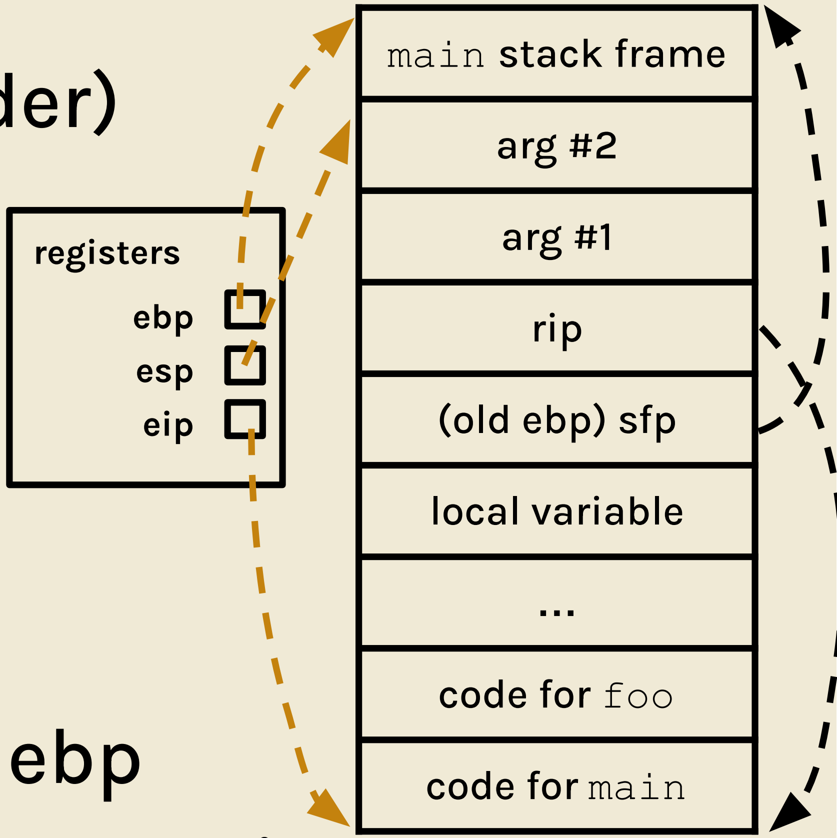
calling convention

1. push arguments (reverse order)
2. remember eip
3. remember ebp
4. adjust the stack frame
5. execute the function
   - and move local variables onto stack



calling convention

1. push arguments (reverse order)
2. remember eip
3. remember ebp
4. adjust the stack frame
5. execute the function
6. restore everything
   - use rip, sfp to restore eip, ebp
   - esp naturally moves up via popping



worksheet

(on 161 website)

OH
4-5pm
MW

4-6
T/Th

feedback

bit.ly/extended-feedback

slides: **bit.ly/cs161-disc**