Lecture 2

Friday, 18 November 2022 15:27

software is very widespread; it makes things cheap + more flexible

our focus: errors in software

many incentives for attack

software security is problematic, e.g. due to the halting problem (proving security is impossible)
↓
but approximations are possible

problem: functionality is typically prioritised over security

fixing things early on is cheaper

1. input representation — most common
2. API abuse
3. improper use of security features
4. time / state
5. error handling
6. code quality
7. data encapsulation (i.e. separation of resources)
8. environment

---

source-code —compiler→ abstract syntax tree
↓
high level intermediate representation
↓ ⋮
low level intermediate representation
↓
machine code, which may be a bit different and have information removed
↓
compilation / code generation is generally irreversible

CPU can only directly execute machine code which is architecture-specific

if a binary is made available to the attacker, they can obtain (equivalent) assembly code
↳ they can test + experiment with this to find exploits/attacks

CPU
control unit (CU) manages proper program execution
arithmetic logical unit (ALU) performs computational tasks on operands stored in registers

program = sequence of instructions

data and instructions are both stored in RAM → many security problems, e.g. confusion

Von Neumann cycle
1. fetch
2. decode
3. fetch operands
4. execute

RISC — reduced instruction set architecture, computing
CISC — complete

memory access only via dedicated load + store instructions

eax: accumulator — temp store for performing arithmetic and such

ecx: counter

edx: I/O data

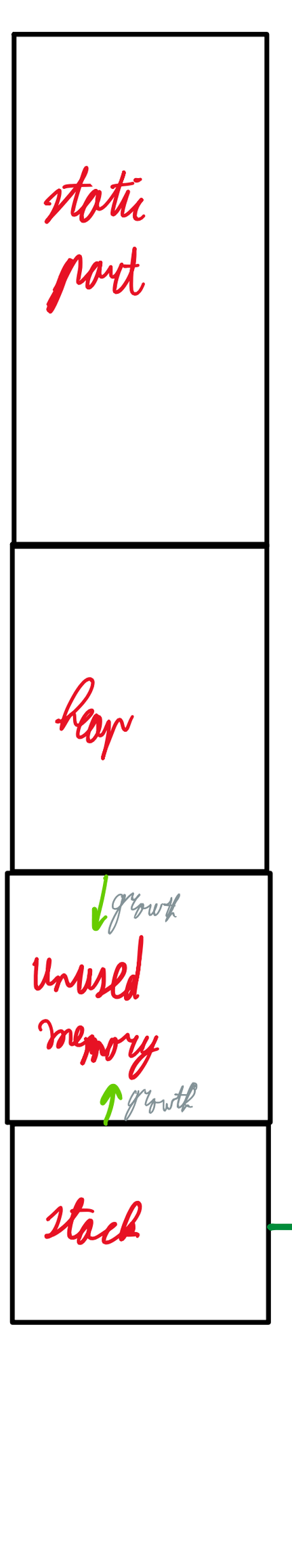
esp: stack pointer
ebp: base pointer to current stack pointer
eip: instruction pointer to next instruction

Intel syntax — target operand always on left
AT&T syntax — source before destination

pointers are evaluated

devices are accessible through RAM

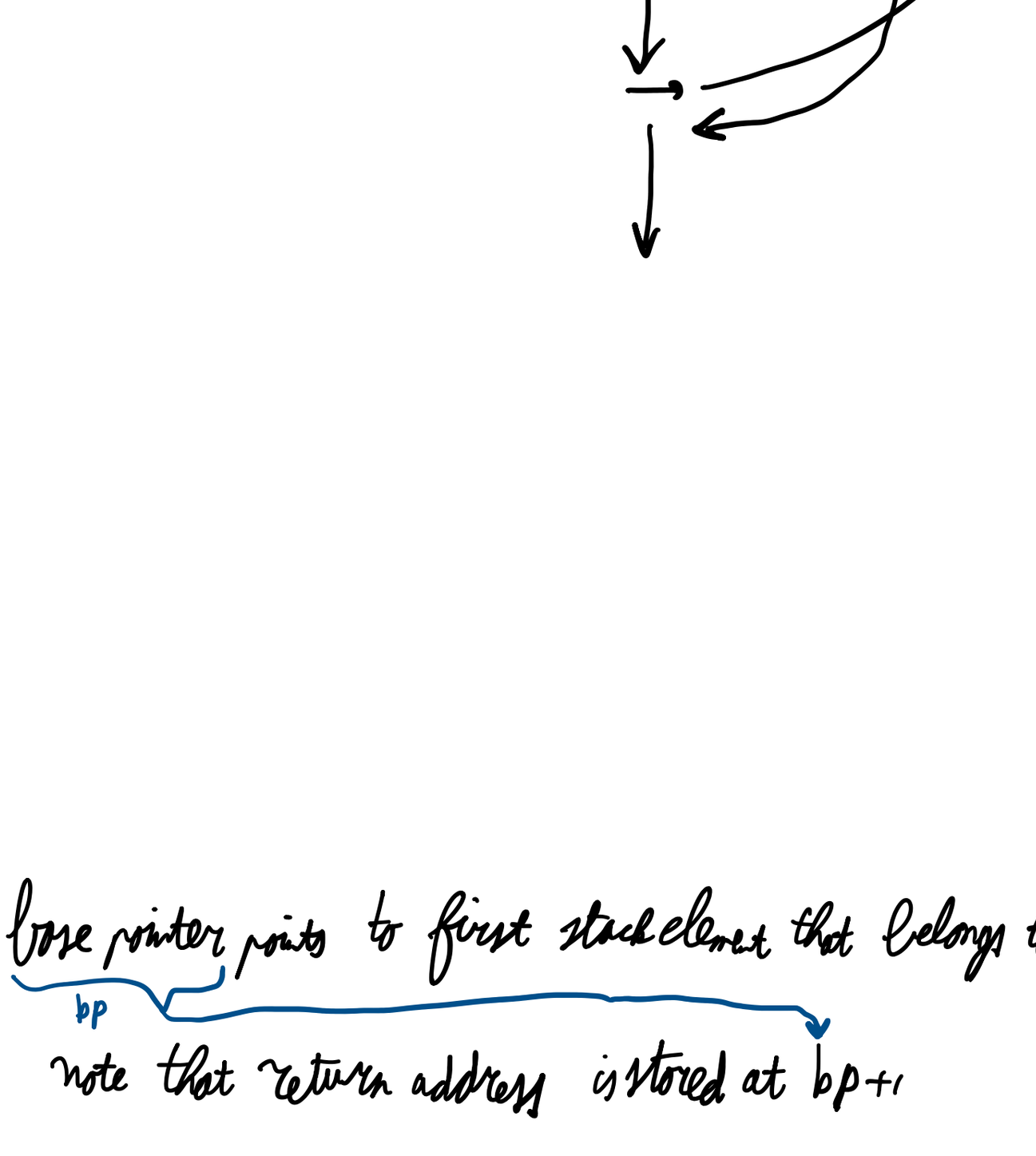
memory layout



→ last-in, first-out

push x
pop
call → stores return address on stack e.g. before function call
ret → pops return address from stack e.g. after function returns

everything above the stack pointer is garbage



base pointer (bp) points to first stack element that belongs to current function call
note that return address is stored at bp+1

heap is generally used for e.g. data structures which may have unknown size

HEAP