

pointers can point to data or code/instructions

dynamic allocation is necessary because the size / number of inputs is unknown at compile time

having multiple references to one memory address can lead to (security) problems (e.g. freeing memory that is still referenced)

having data on the stack can lead to problems with the stack pointer has been increased again, but this (sensitive) data has not been scrubbed (i.e. dereferencing & deleting) <sup>i.e. gone back</sup>

### memory corruption & buffer overflow



more checks & bounds → slower execution

about 70% at MSFT → memory corruption

C/C++ are still frequently used (because they are so efficient)   
 even though they are far more susceptible to memory corruption attacks

buffer overflow example: give more input than intended, leading to overwrite return address (and possibly store malicious code, which the return address can point to)

can also insert lots of null operations, <sup>which just get skipped</sup> to make sure that the actual code gets reached, even if its address is unknown <sup>cached</sup>

other strategy: change return address to skip out of loop which was used to require the user to correctly enter a password

code injection: put code into input field

orc injection: jump to different position in program by inserting return address

return-to-libc → call function from library which is already in memory

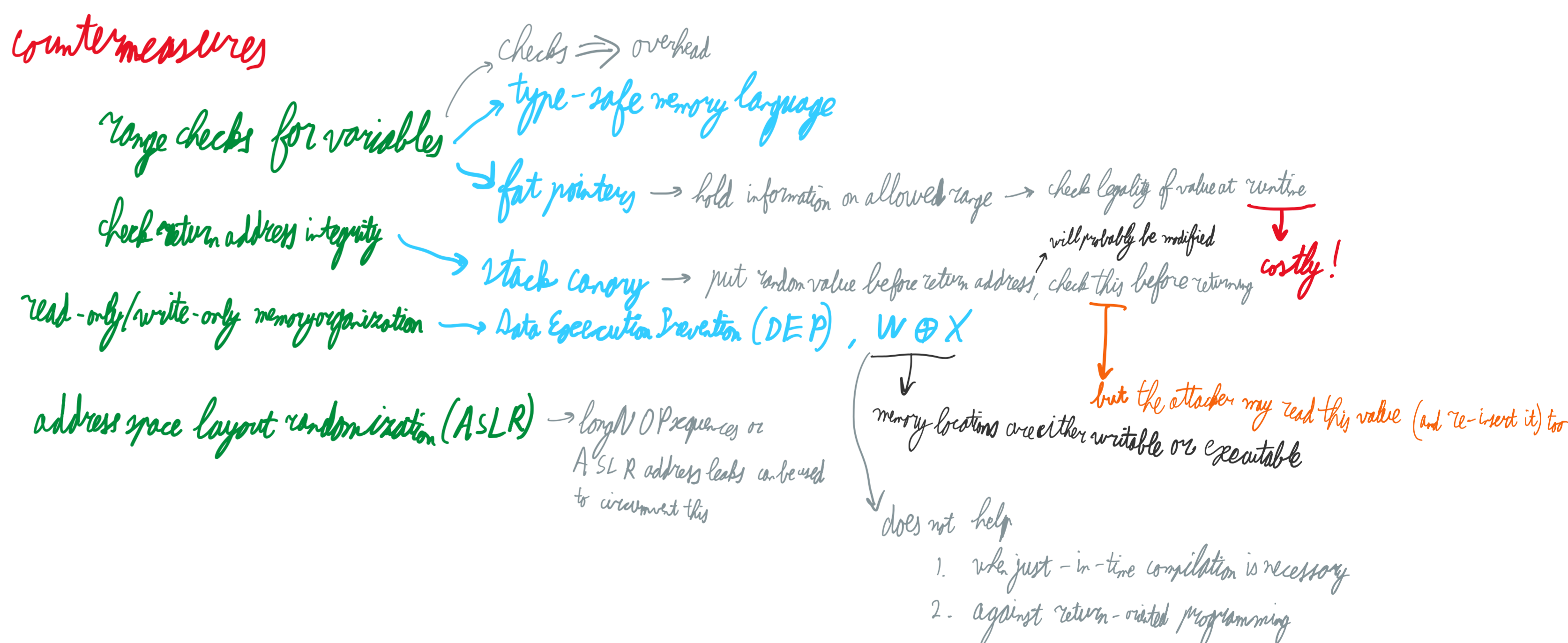
return-oriented programming → change byte alignment (i.e. starting point for interpretation) to execute code in an 'unintended' way

these 'gadgets' can be Turing complete e.g. in x86

### observations:

1. buffer overflow is possible
2. overwriting of return address does not raise alarms
3. malicious code must be executable
4. return address must map to malicious code

### countermeasures



### other problems:

heap overflow

corrupting vtables (virtual tables) → effectively: call function of the wrong type of object

NULL pointer → NULL is a dedicated address that pointers cannot access

### pointer security problems

- NULL dereference: assuming pointer to map to a value, while it actually maps to the NULL pointer
- dangling pointer: a pointer maps to memory address which has already been freed
- use after-free / double free: a pointer is used after it has been freed
- memory leak: allocated memory is never freed