Lesson 1.3: Spawning Subprocesses

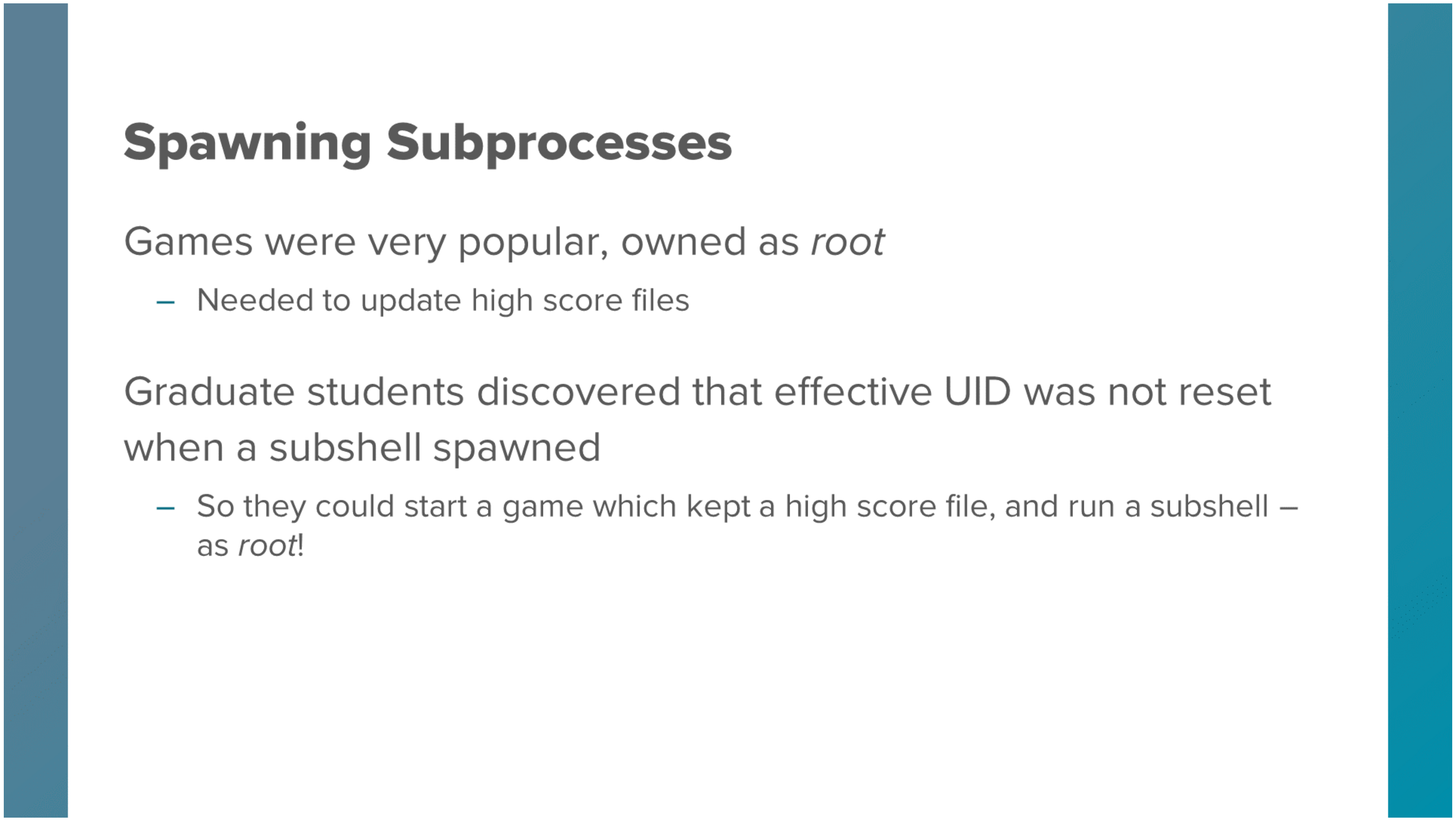
SECURITY VULNERABILITIES IN C/C++ PROGRAMMING

Spawning Subprocesses

Matt Bishop, Ph.D.
Professor of Computer Science,
UC Davis

UCDAVIS
Continuing and Professional Education

Slide 1: Spawning Subprocesses



Spawning Subprocesses

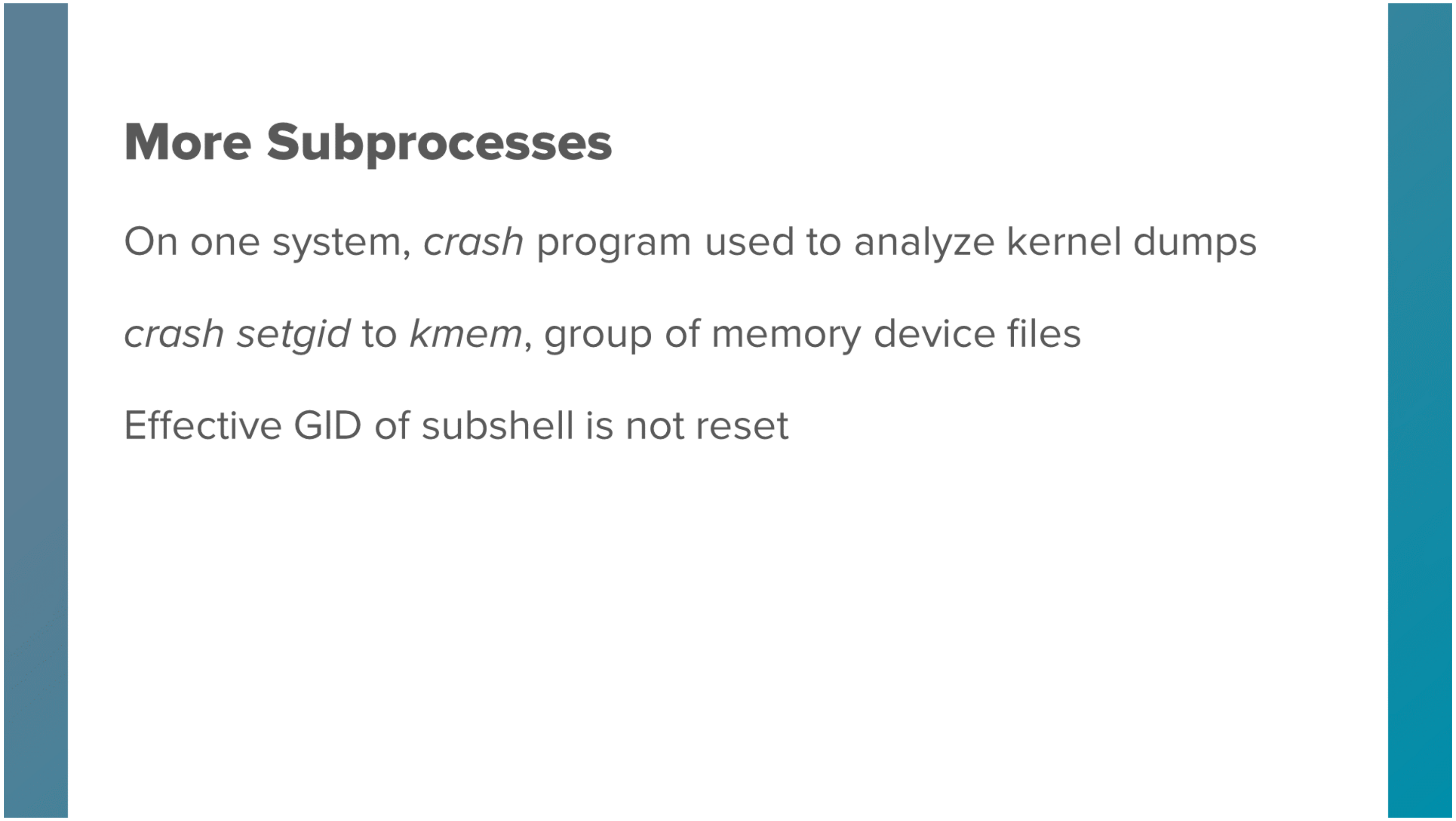
Games were very popular, owned as *root*

- Needed to update high score files

Graduate students discovered that effective UID was not reset when a subshell spawned

- So they could start a game which kept a high score file, and run a subshell – as *root*!

Slide 2: More Subprocesses



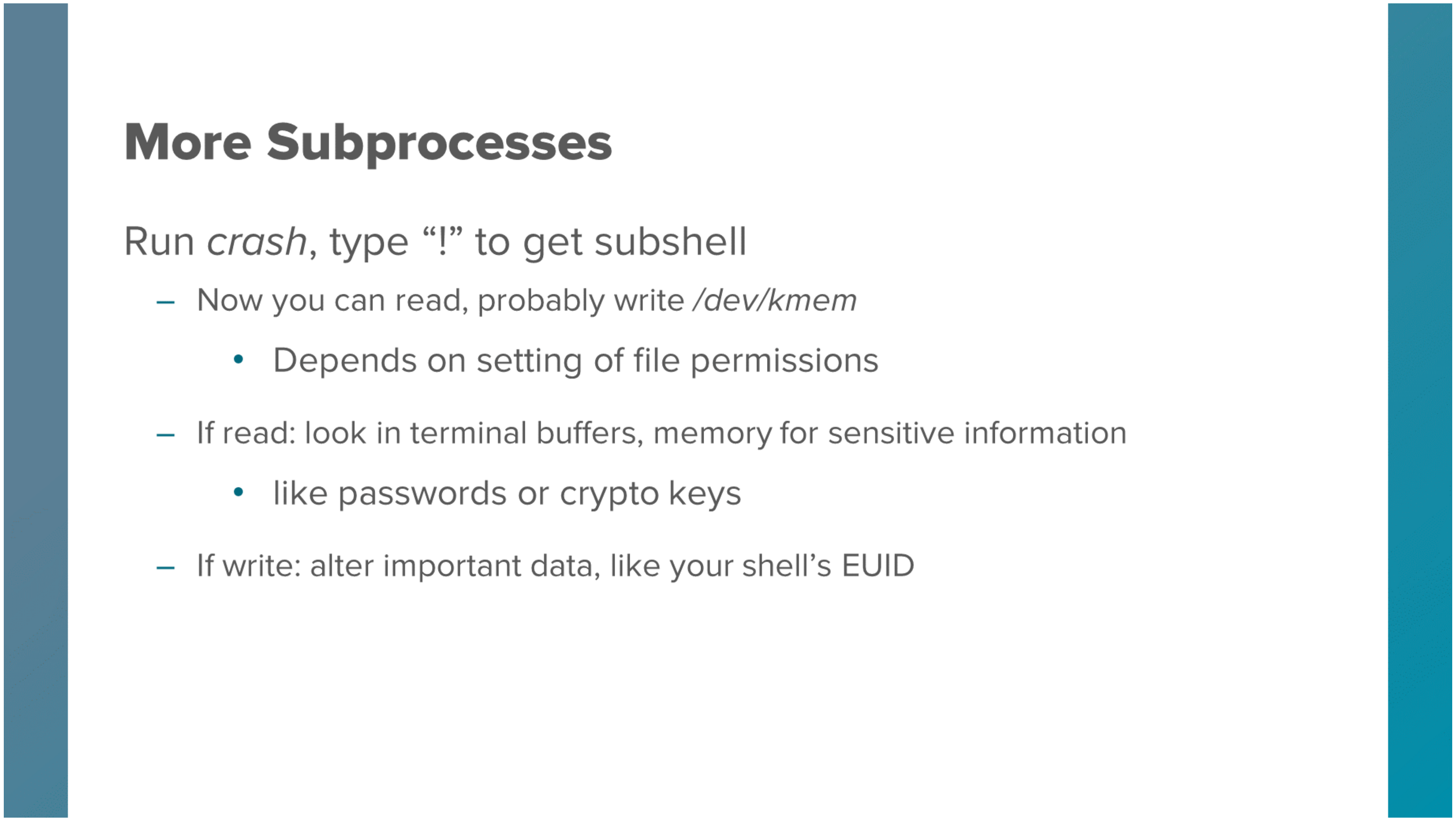
More Subprocesses

On one system, *crash* program used to analyze kernel dumps

crash setgid to *kmem*, group of memory device files

Effective GID of subshell is not reset

Slide 3: More Subprocesses

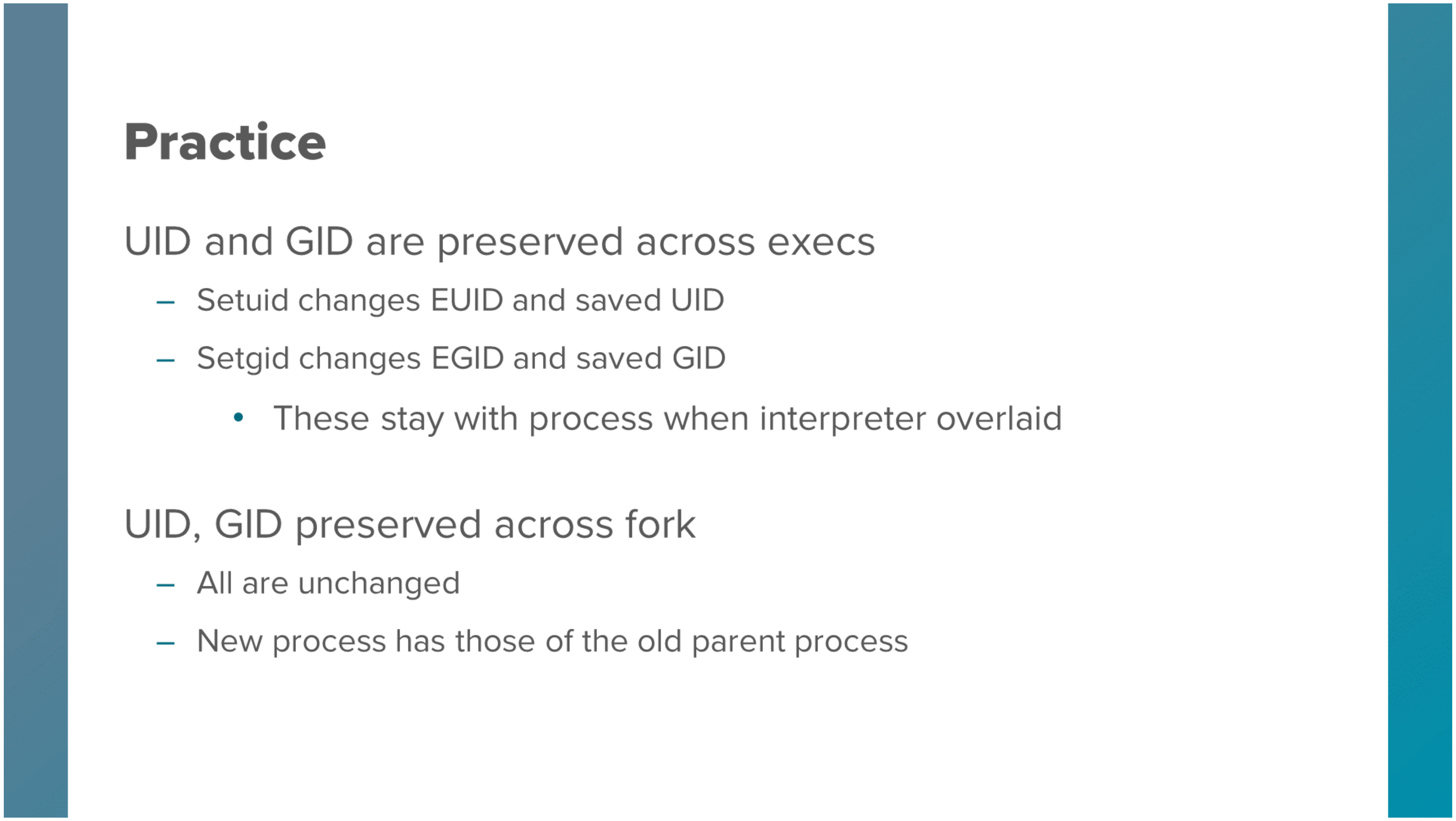


More Subprocesses

Run *crash*, type “!” to get subshell

- Now you can read, probably write */dev/kmem*
    - Depends on setting of file permissions
- If read: look in terminal buffers, memory for sensitive information
    - like passwords or crypto keys
- If write: alter important data, like your shell’s EUID

Slide 4: Practice



Practice

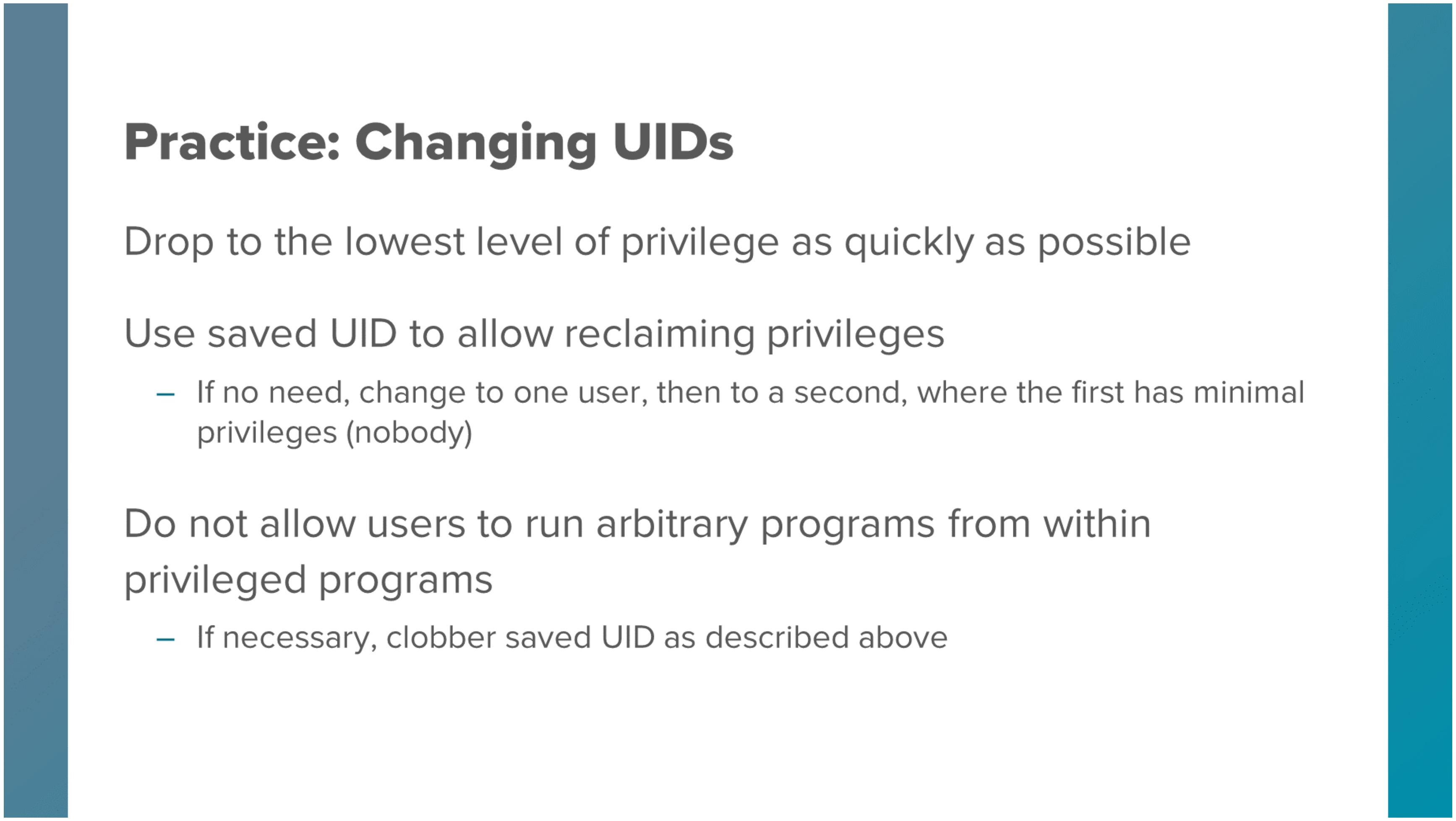
UID and GID are preserved across execs

- Setuid changes EUID and saved UID
- Setgid changes EGID and saved GID
  - These stay with process when interpreter overlaid

UID, GID preserved across fork

- All are unchanged
- New process has those of the old parent process

Slide 5: Practice: Changing UIDs



Practice: Changing UIDs

Drop to the lowest level of privilege as quickly as possible

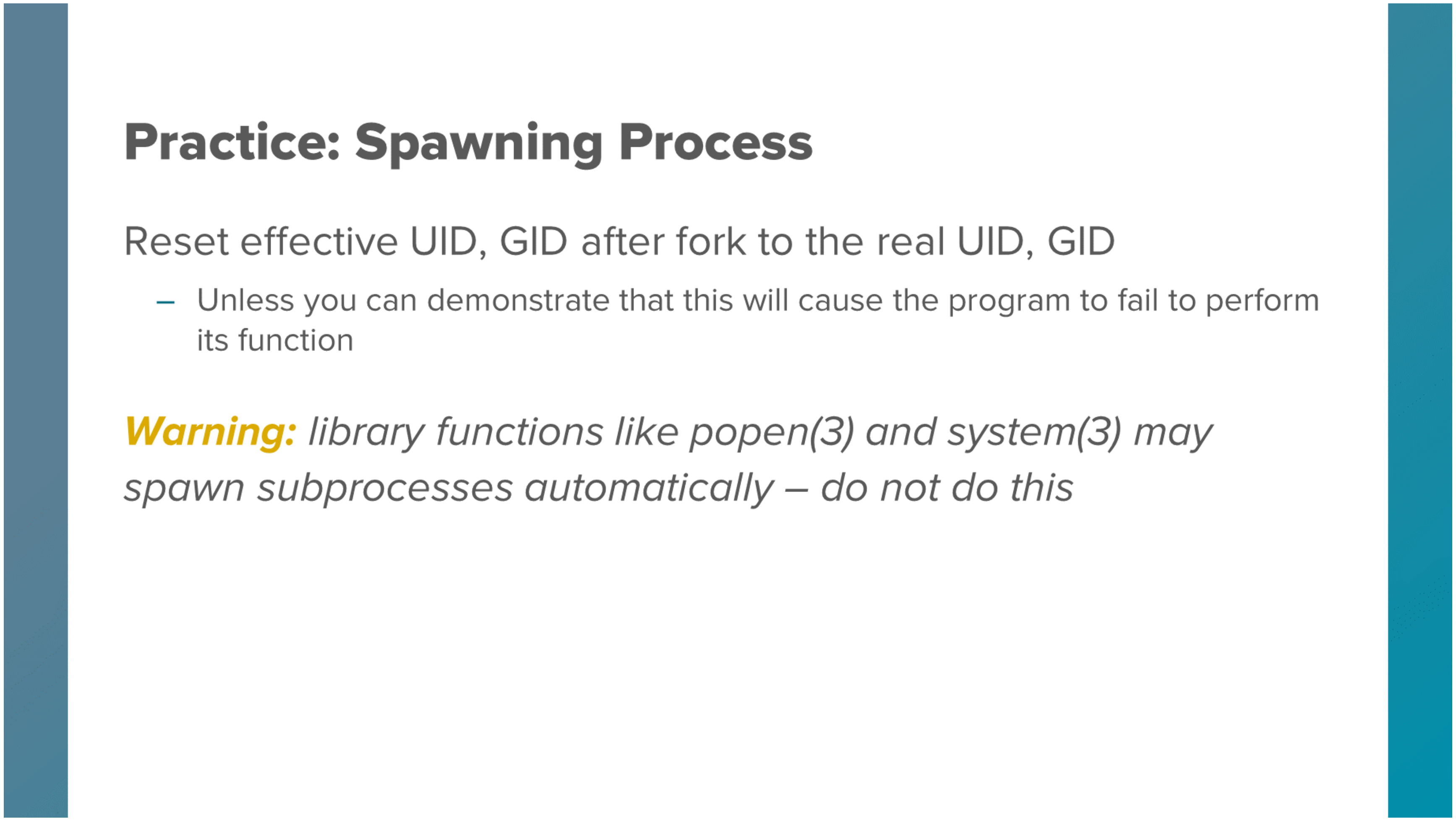
Use saved UID to allow reclaiming privileges

- If no need, change to one user, then to a second, where the first has minimal privileges (nobody)

Do not allow users to run arbitrary programs from within privileged programs

- If necessary, clobber saved UID as described above

Slide 6: Practice: Spawning Process



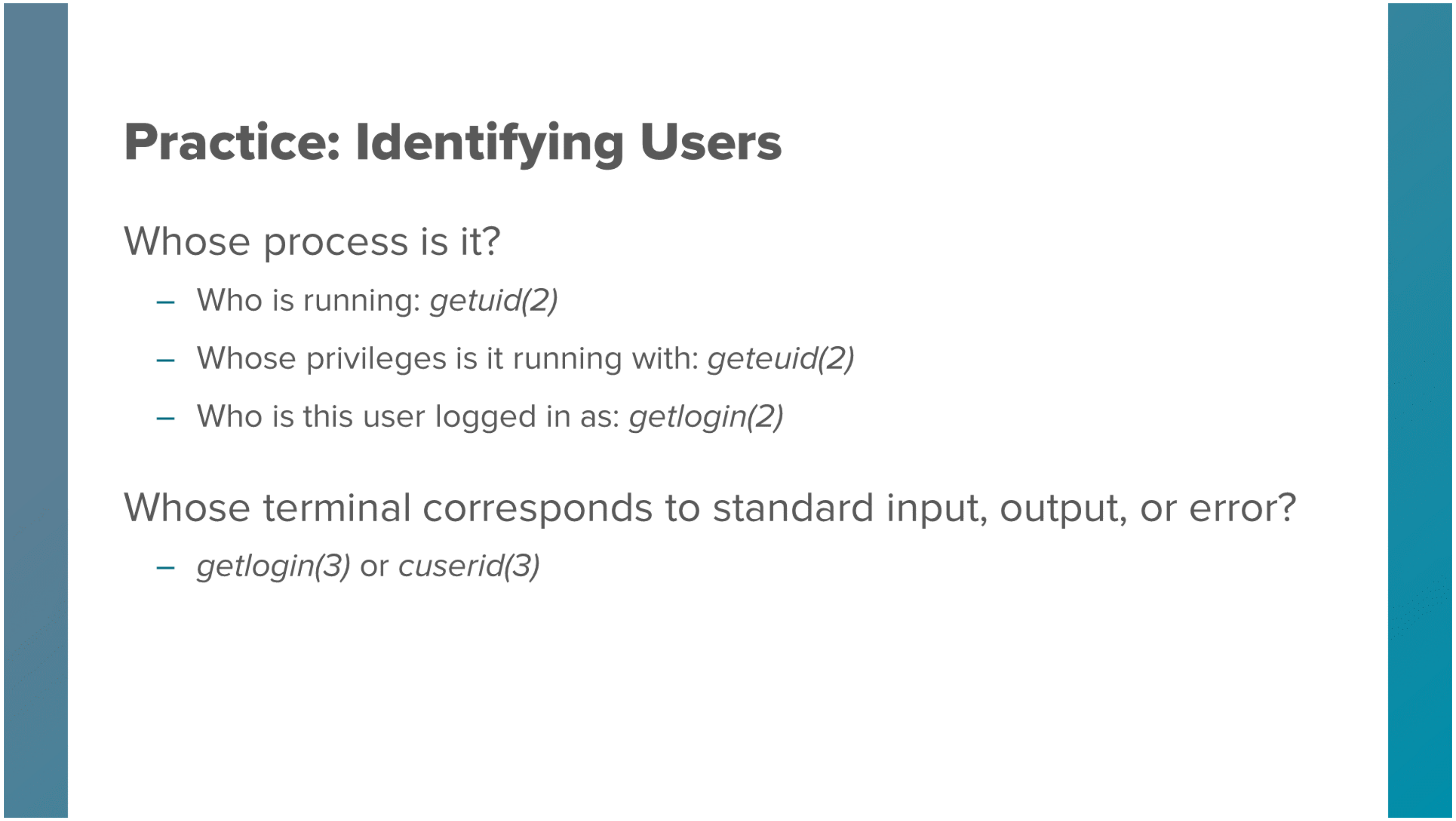
Practice: Spawning Process

Reset effective UID, GID after fork to the real UID, GID

– Unless you can demonstrate that this will cause the program to fail to perform its function

Warning: *library functions like popen(3) and system(3) may spawn subprocesses automatically – do not do this*

Slide 7: Practice: Identifying Users



Practice: Identifying Users

Whose process is it?

- Who is running: *getuid(2)*
- Whose privileges is it running with: *geteuid(2)*
- Who is this user logged in as: *getlogin(2)*

Whose terminal corresponds to standard input, output, or error?

- *getlogin(3)* or *cuserid(3)*