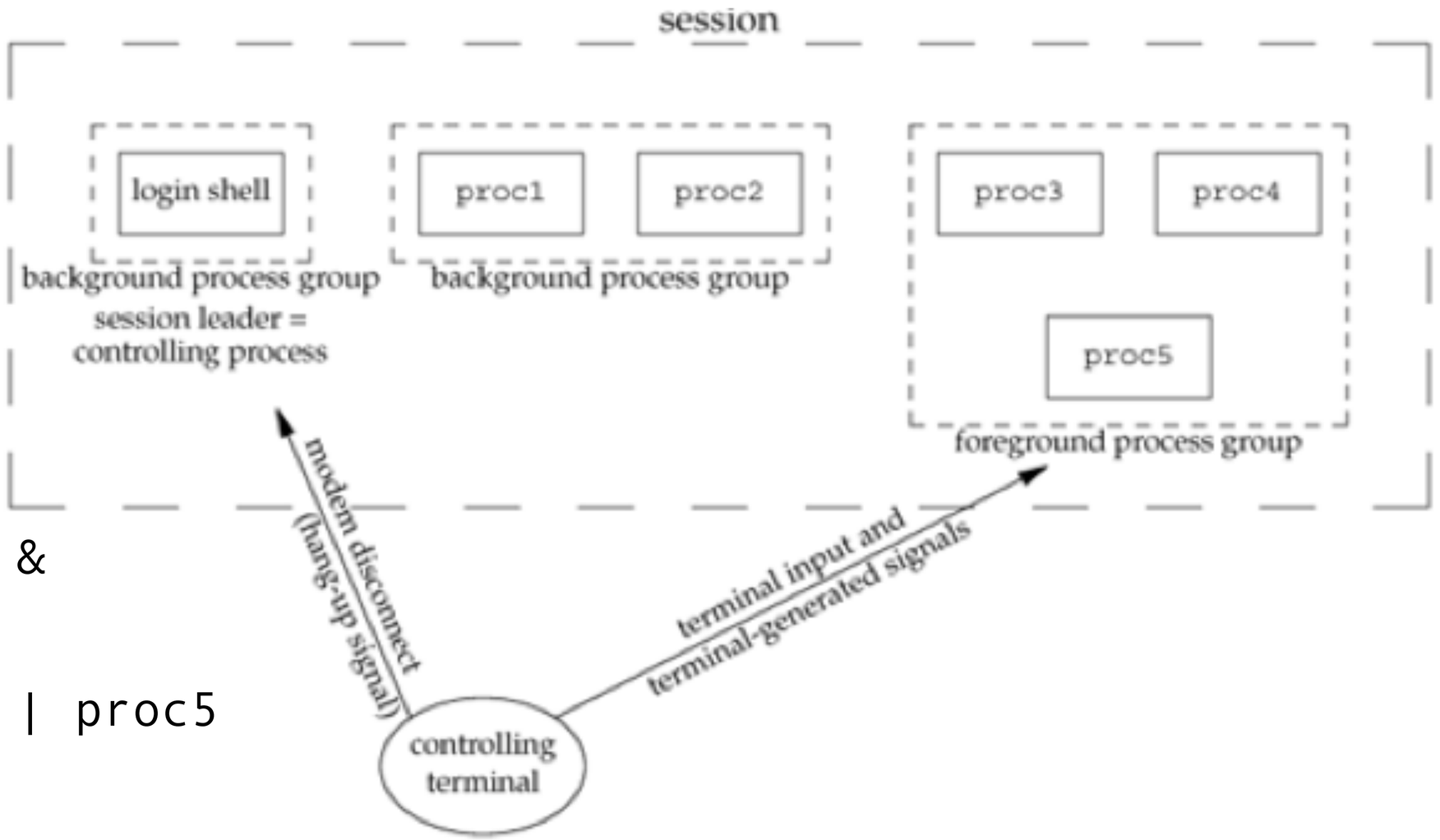
Advanced Programming in the UNIX Environment

Week 07, Segment 3:
Job Control

Department of Computer Science
Stevens Institute of Technology

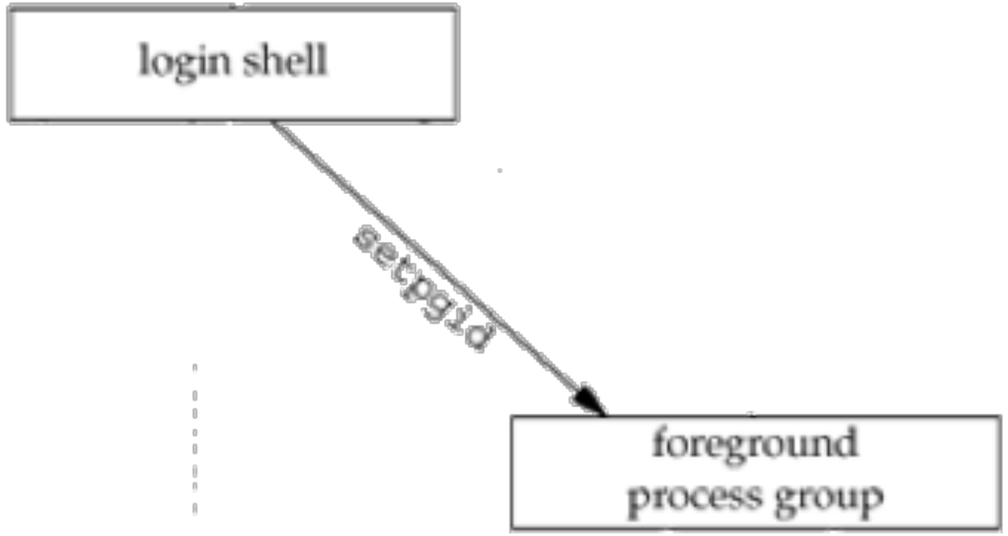
Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

Process Groups

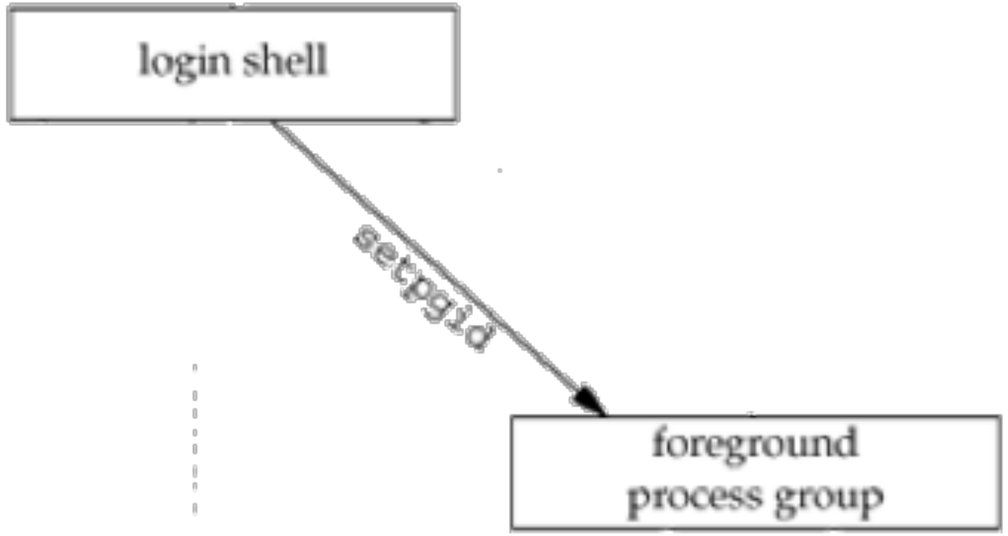



```
$ proc1 | proc2 &
[1] 10306
$ proc3 | proc4 | proc5
```

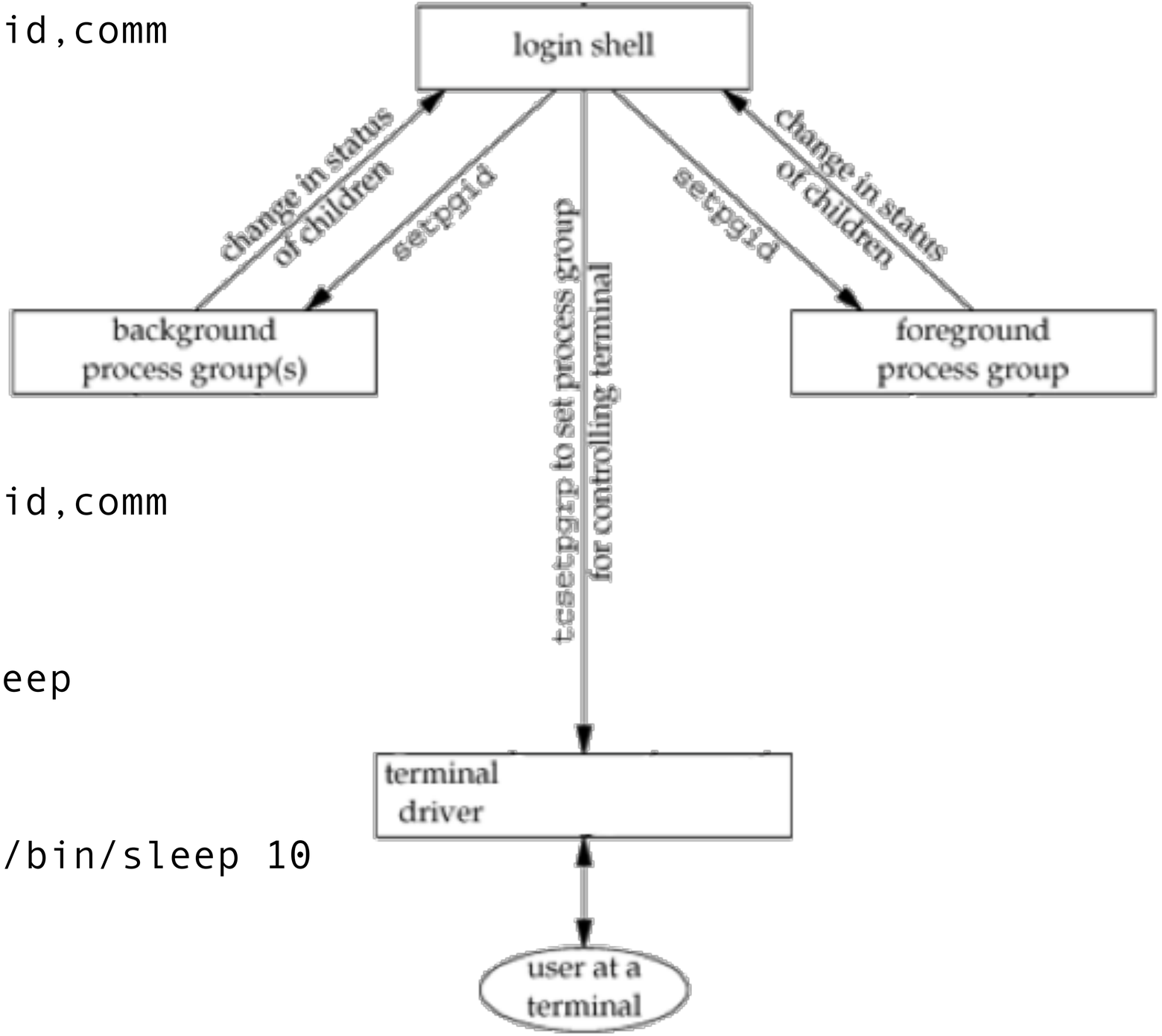
```
apue$ ps -o pid,ppid,pgid,sid,comm
  PID PPID PGID  SID COMMAND
1188 1107 1188 1188 -ksh
1455 1188 1455 1188 ps
apue$
```



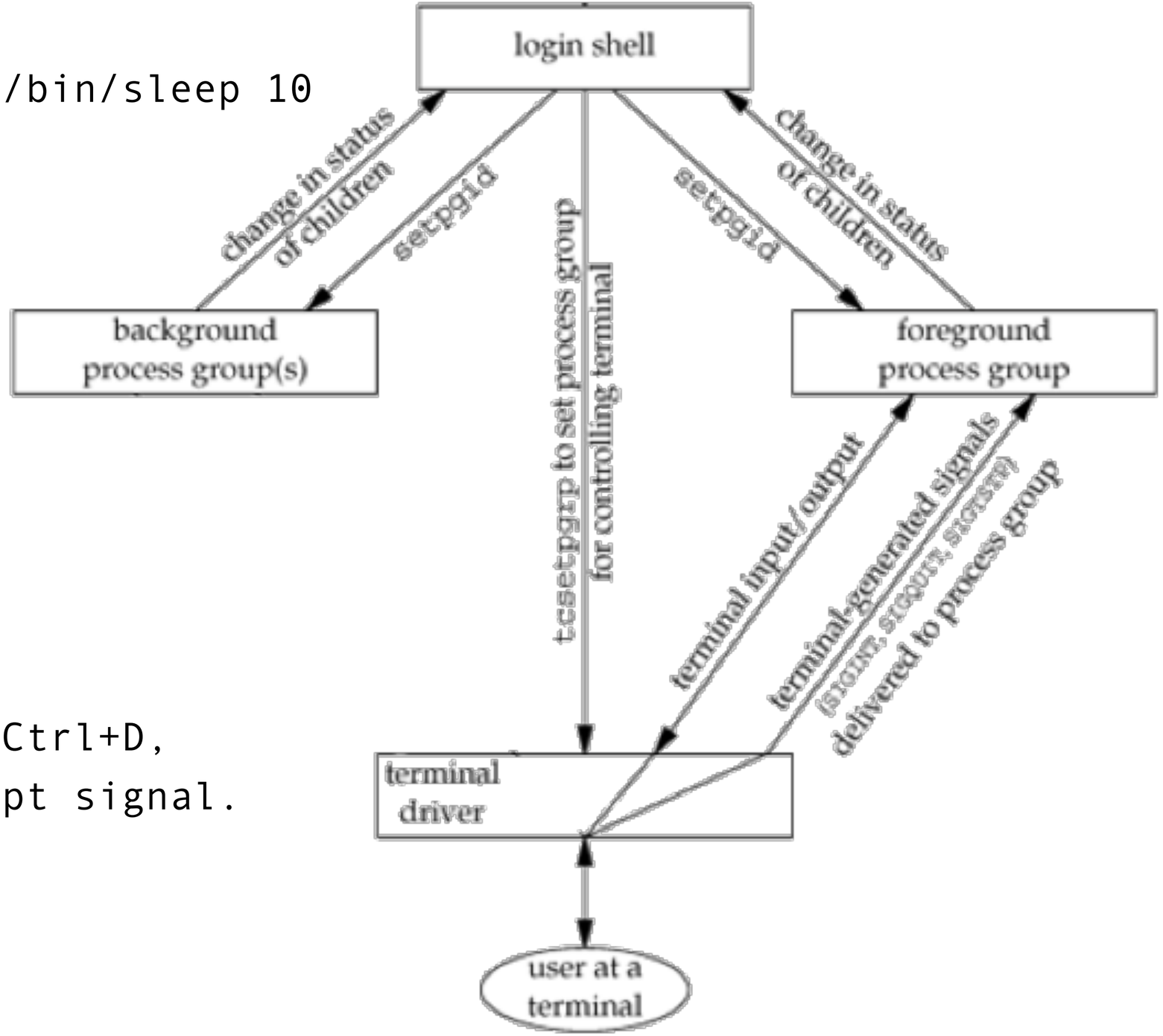
```
apue$ ps -o pid,ppid,pgid,sid,comm
 PID PPID PGID  SID COMMAND
1188 1107 1188 1188 -ksh
1455 1188 1455 1188 ps
apue$ echo $?
0
apue$
```



```
apue$ ps -o pid,ppid,pgid,sid,comm
 PID PPID PGID  SID COMMAND
1188 1107 1188 1188 -ksh
1455 1188 1455 1188 ps
apue$ echo $?
0
apue$ /bin/sleep 10 &
[1] 1446
apue$ ps -o pid,ppid,pgid,sid,comm
 PID PPID PGID  SID COMMAND
1188 1107 1188 1188 -ksh
1446 1188 1446 1188 /bin/sleep
1589 1188 1589 1188 ps
apue$
[1] + Done                    /bin/sleep 10
apue$
```

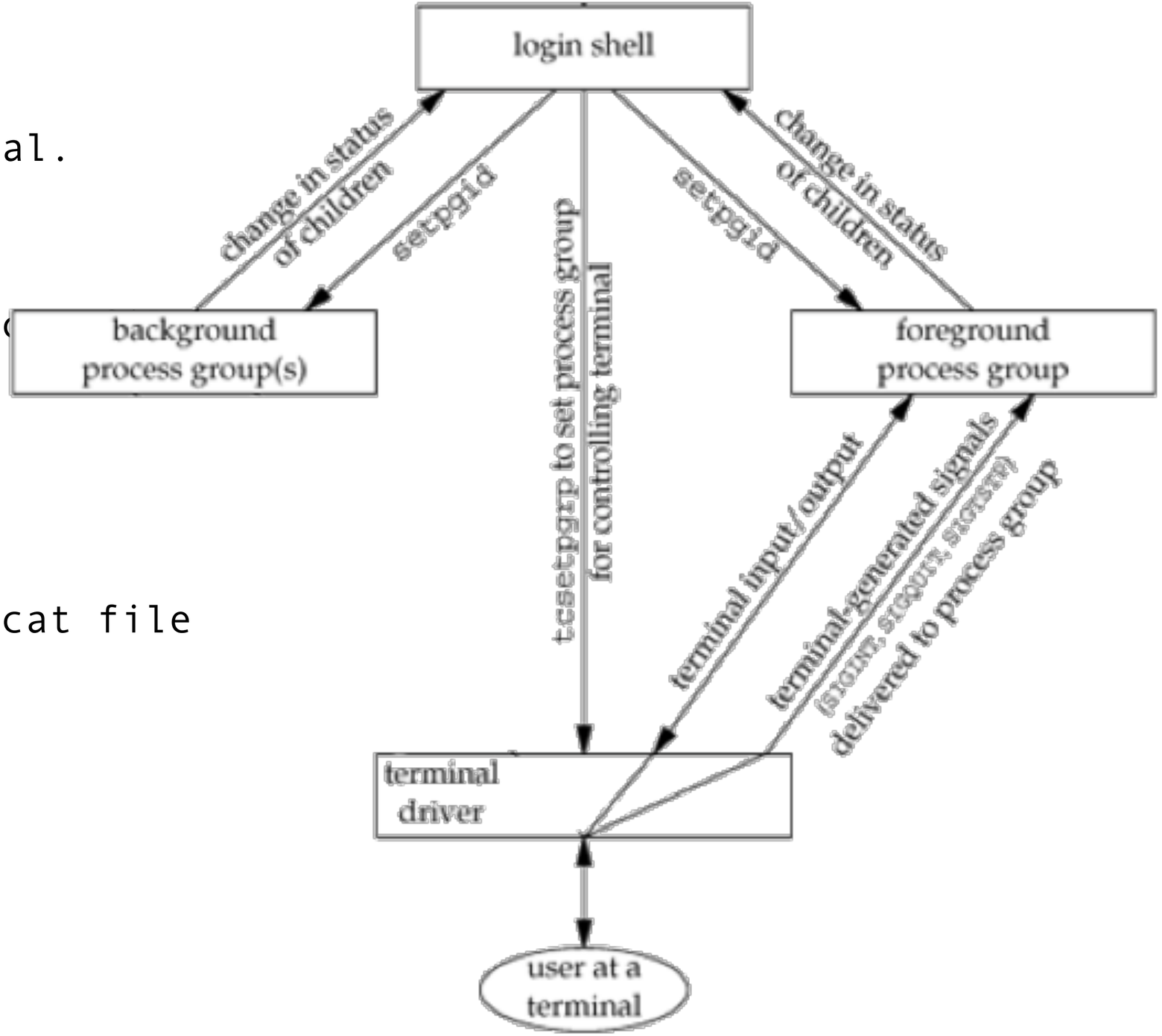


```
apue$
[1] + Done                       /bin/sleep 10
apue$ cat >file
Input from the terminal.
Output to the terminal.
apue$ cat file
Input from the terminal.
Output to the terminal.
apue$ cat >/dev/null
Same here.
cat(1) calls read(2), which
now happily blocks forever.
We can either send EOF via Ctrl+D,
or we could send an interrupt signal.
^C
apue$
```



```
apue$ cat file &
[1] 341
apue$ Input from the terminal.
Output to the terminal.

[1] + Done
apue$ stty tostop
apue$ cat file &
[1] 345
apue$
[1] + Stopped (tty output) cat file
apue$ fg
cat file
Input from the terminal.
Output to the terminal.
apue$
```



```
apue$ proc1 | proc2 &
[1] 1776 549
apue$ jobs -l
[1] +  1776 Stopped (tty output) proc1 |
        549                      proc2
apue$ vim ~/01/simple-cat.c
[2] + Stopped                vim ~/01/simple-cat.c
apue$ cc ~/01/simple-cat.c
apue$ ./a.out </dev/zero | ./a.out | ./a.out >/dev/null
^Z[3] + Stopped                ./a.out < /dev/zero | ./a.out | ./a.out > /dev/null
apue$ jobs -l
[3] +  1653 Stopped              ./a.out < /dev/zero |
       1591                      ./a.out |
       1780                      ./a.out > /dev/null
[2] -  1466 Stopped              vim ~/01/simple-cat.c
[1]    1776 Stopped (tty output) proc1 |
        549                      proc2
apue$ bg %3
[3] ./a.out < /dev/zero | ./a.out | ./a.out > /dev/null
apue$ jobs -l
[2] +  1466 Stopped              vim ~/01/simple-cat.c
[1] -  1776 Stopped (tty output) proc1 |
        549                      proc2
[3]    1653 Running              ./a.out < /dev/zero |
       1591                      ./a.out |
       1780                      ./a.out > /dev/null
```

```
apue$ fg %1
proc1 | proc2
apue$ jobs -l
[2] +  1466 Stopped                 vim ~/01/simple-cat.c
[3] -  1653 Running                 ./a.out < /dev/zero |
       1591                         ./a.out |
       1780                         ./a.out > /dev/null
apue$ kill -TSTP 1591
apue$ jobs -l
[2] +  1466 Stopped                 vim ~/01/simple-cat.c
[3] -  1653 Running                 ./a.out < /dev/zero |
       1591 Stopped                 ./a.out |
       1780 Running                 ./a.out > /dev/null
apue$ kill -CONT 1591
apue$ kill 1591
apue$ jobs -l
[2] +  1466 Stopped                 vim ~/01/simple-cat.c
[3] -  1653 Broken pipe             ./a.out < /dev/zero |
       1591 Terminated              ./a.out |
       1780 Done                    ./a.out > /dev/null
apue$ jobs -l
[2] +  1466 Stopped                 vim /home/jschauma/01/simple-cat.c
apue$ fg
vim ~/01/simple-cat.c
```

Job Control

- both background and foreground process groups may report a change in status to the login shell
- the foreground process group can perform I/O on the controlling terminal
- the controlling terminal can generate signals via keyboard interrupts to send to the foreground process group
- the background process group may be able to write to the controlling terminal
- the background process group may generate a signal to send to the controlling terminal if it needs to perform I/O
- the shell may move process groups into the foreground or background, suspend or continue them
- we can send any signal to any process via `kill(1)`