UNIX 102

Ryan DeRue, Senior Computational Scientist

PURDUE UNIVERSITY | Rosen Center for Advanced Computing

Unix 102

Outline

Rosen Center for Advanced Computing

What to expect from Unix 102

Objectives

- Examine the most useful features of shells from a productivity perspective

- Become familiar with file descriptors, the standard streams, and how these interact with commands

- Learn how to develop pipelines through composing several commands together

Unix 102

Reexamining Unix Shells

Reexamining Unix Shells

Helpful features of most Unix Shells

- Maintains a history of your most recently used commands
    - Commands can be re-accessed using the up and down arrow keys for quick re-use
    - Reverse search using ctrl+R
- File name substitution/completion
    - Hitting the tab key once while typing a file will auto-complete the rest of the filename for you if no other files match your current pattern
    - Hitting the tab key twice will list all the files which match your current pattern
- Wildcarding
    - The wildcard (*) character can be used for using commands on many files at once

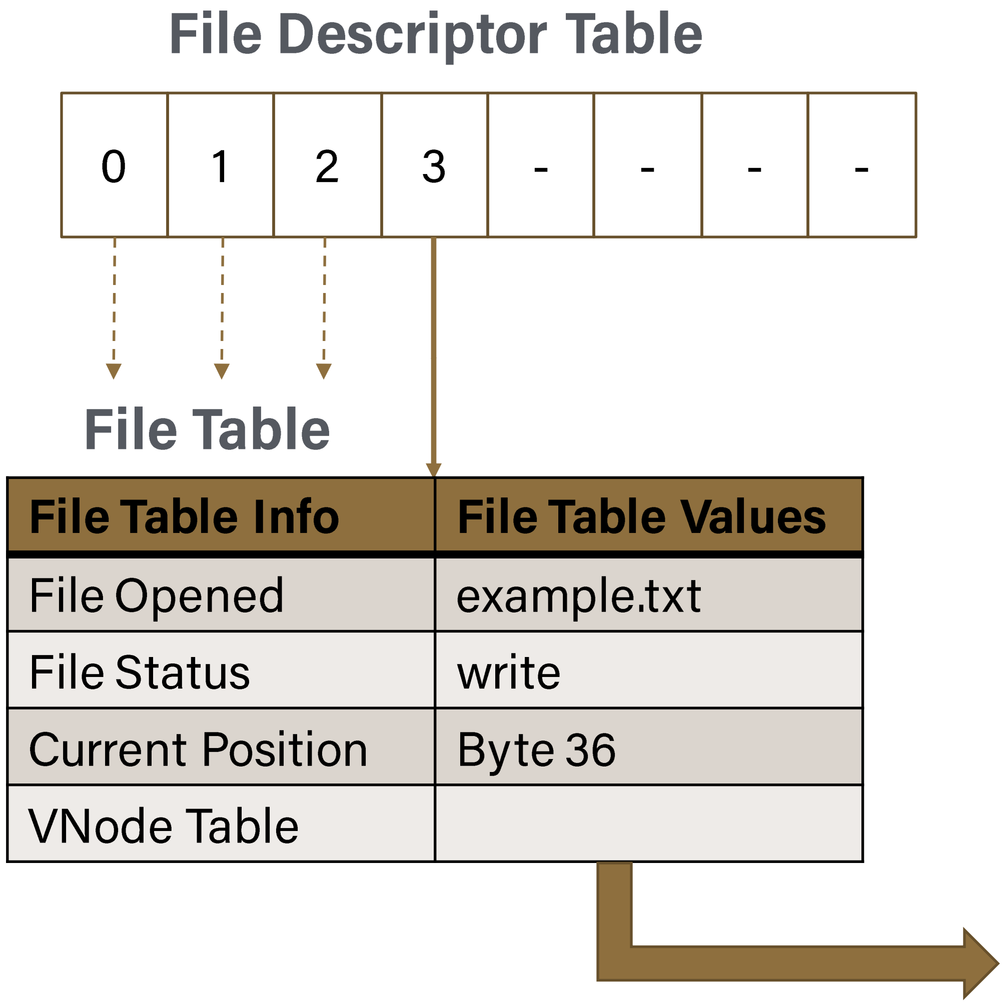
Unix 102

File I/O in Unix

File I/O in Unix

What happens when we open a file?

- The kernel creates a file descriptor for that file
  - File descriptor: Non-negative integer index
- The open file will have certain information tracked
  - File status flags (mode)
  - Current position in the file
- Example: `nano example.txt`



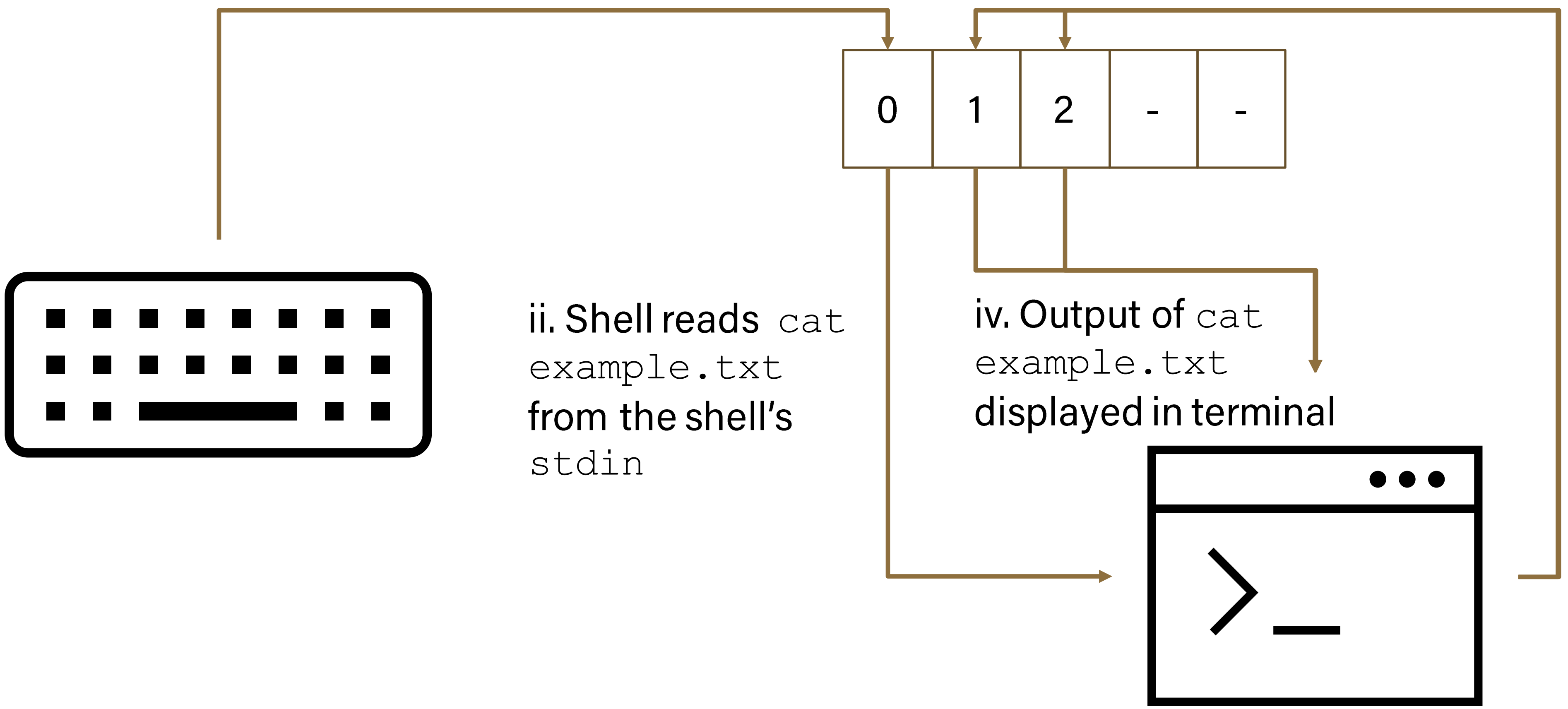
File Table Info	File Table Values
File Opened	example.txt
File Status	write
Current Position	Byte 36
VNode Table	

File I/O in Unix

What about the first 3 file descriptors?

- Recall the third point of the Unix Philosophy:
  - "Write programs to handle text streams, because that is a universal interface."
- A stream is a channel through which we can transfer data
  - Just like a stream of water, it has an inflow, and an outflow
- The standard streams
  - `0: stdin` – The default source for input data
  - `1: stdout` – The default destination for output data
  - `2: stderr` – The default destination for error data

Example of stream I/O

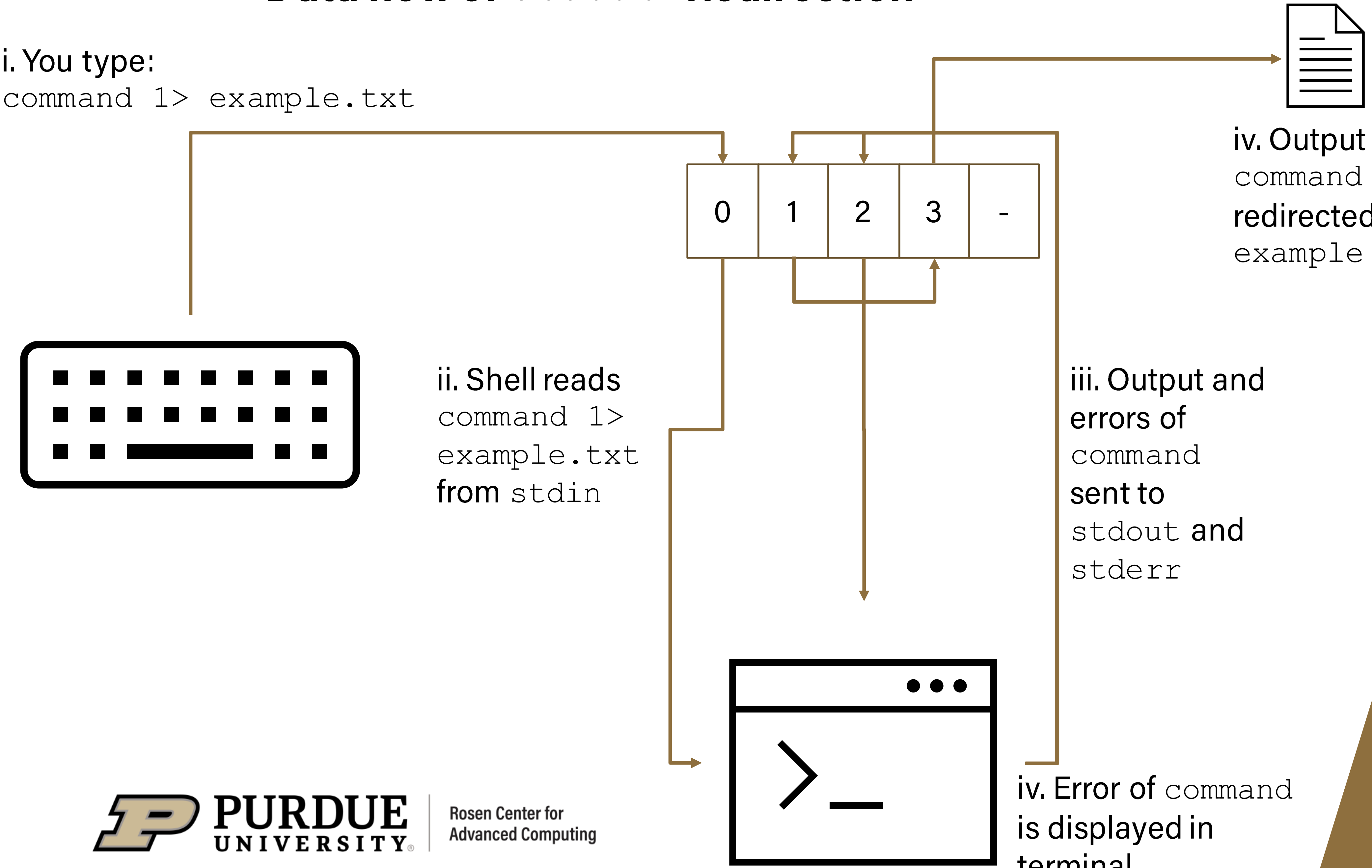

i. You type `cat example.txt`
0
1
2
-
-
ii. Shell reads `cat example.txt` from the shell's `stdin`
iii. Output and errors of `cat example.txt` sent to `stdout` and `stderr`
iv. Output of `cat example.txt` displayed in terminal

File I/O in Unix

What if I want to send the output somewhere else?

- File descriptor redirection
  - We can modify the "things" our file descriptors point to
- The redirection operators
  - `command [fd1]> [ FILE | &fd2 ]`
    - Execute `command` while pointing `fd1` to the file descriptor belonging to `FILE`
    - *Overwrites the contents of `FILE`*
  - `command [fd1]>> [ FILE | &fd2 ]`
  - Execute `command` while pointing `fd1` to the file descriptor belonging to `FILE`
    - Appends the output to `FILE`
  - `command < FILE`
    - Execute `command` while treating the file descriptor belonging to `FILE` as `stdin`

Data flow of `stdout` Redirection



i. You type:
`command 1> example.txt`

ii. Shell reads `command 1> example.txt` from `stdin`

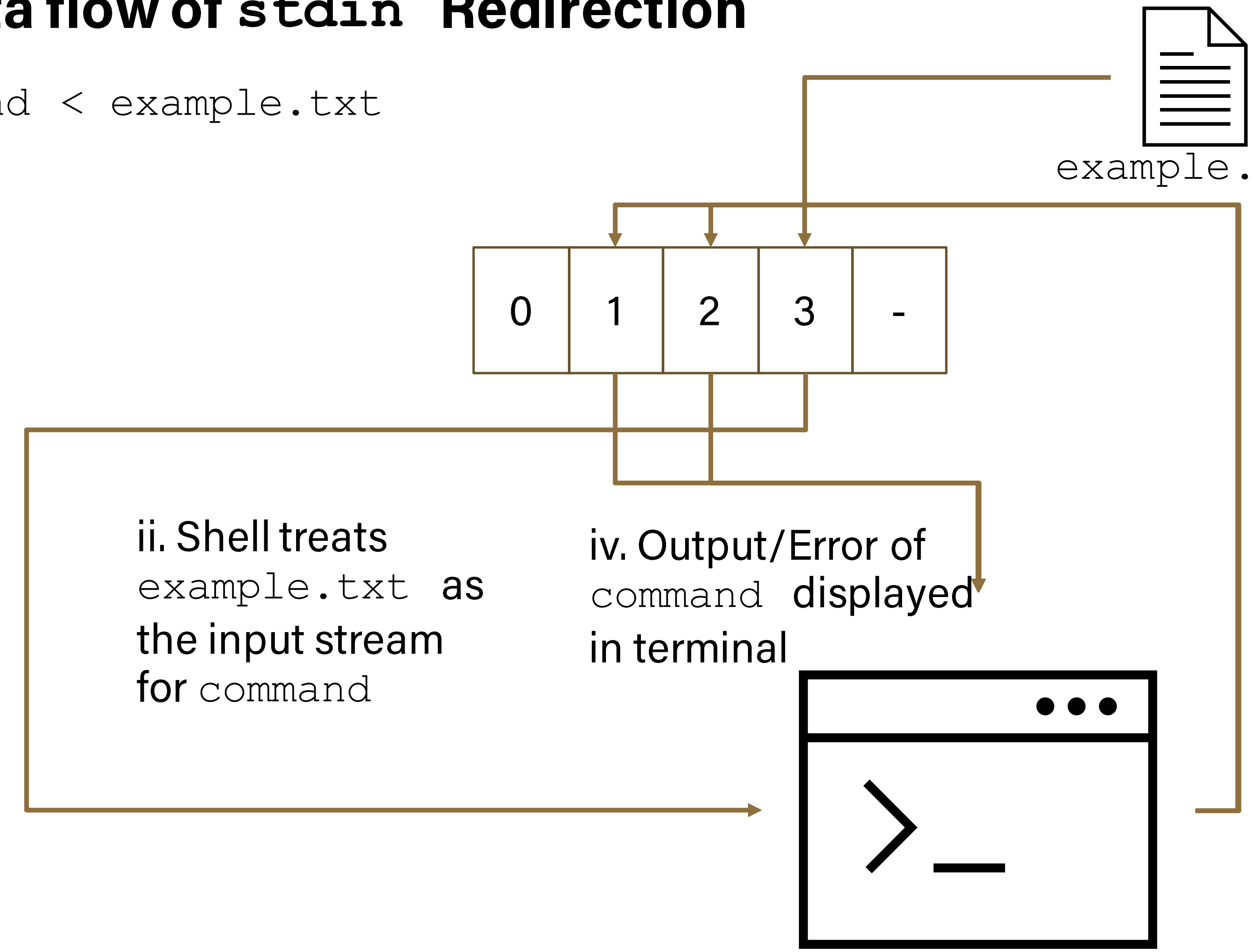
iii. Output and errors of `command` sent to `stdout` and `stderr`

iv. Output of `command` is redirected to `example.txt`

iv. Error of `command` is displayed in terminal

Data flow of `stdin` Redirection

i. You type `command < example.txt`



iii. Output and errors of `command` sent to `stdout` and `stderr`

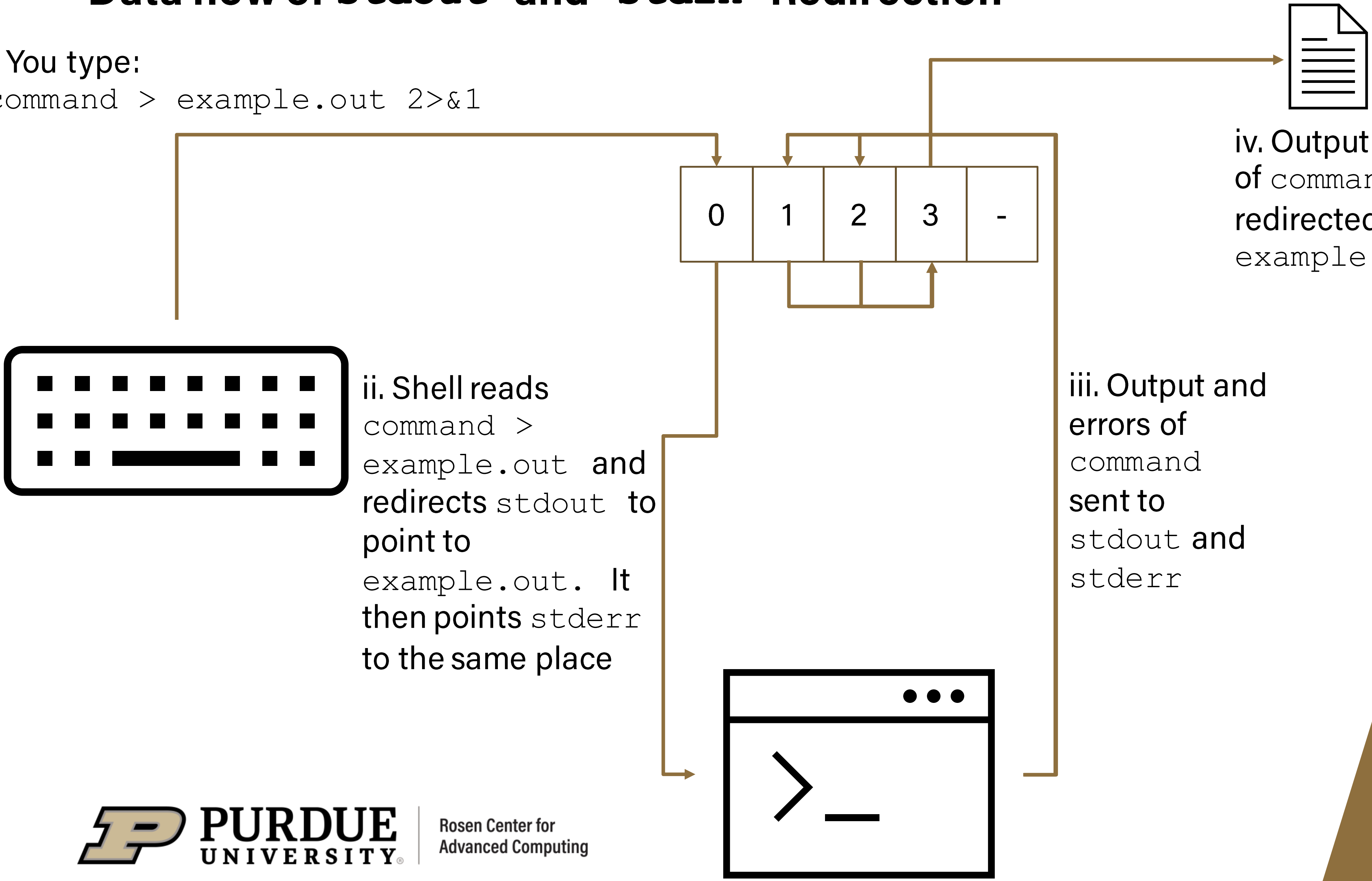
ii. Shell treats `example.txt` as the input stream for `command`

iv. Output/Error of `command` displayed in terminal

Data flow of `stdout` and `stdin` Redirection

i. You type:

`command > example.out 2>&1`



ii. Shell reads `command > example.out` and redirects `stdout` to point to `example.out`. It then points `stderr` to the same place

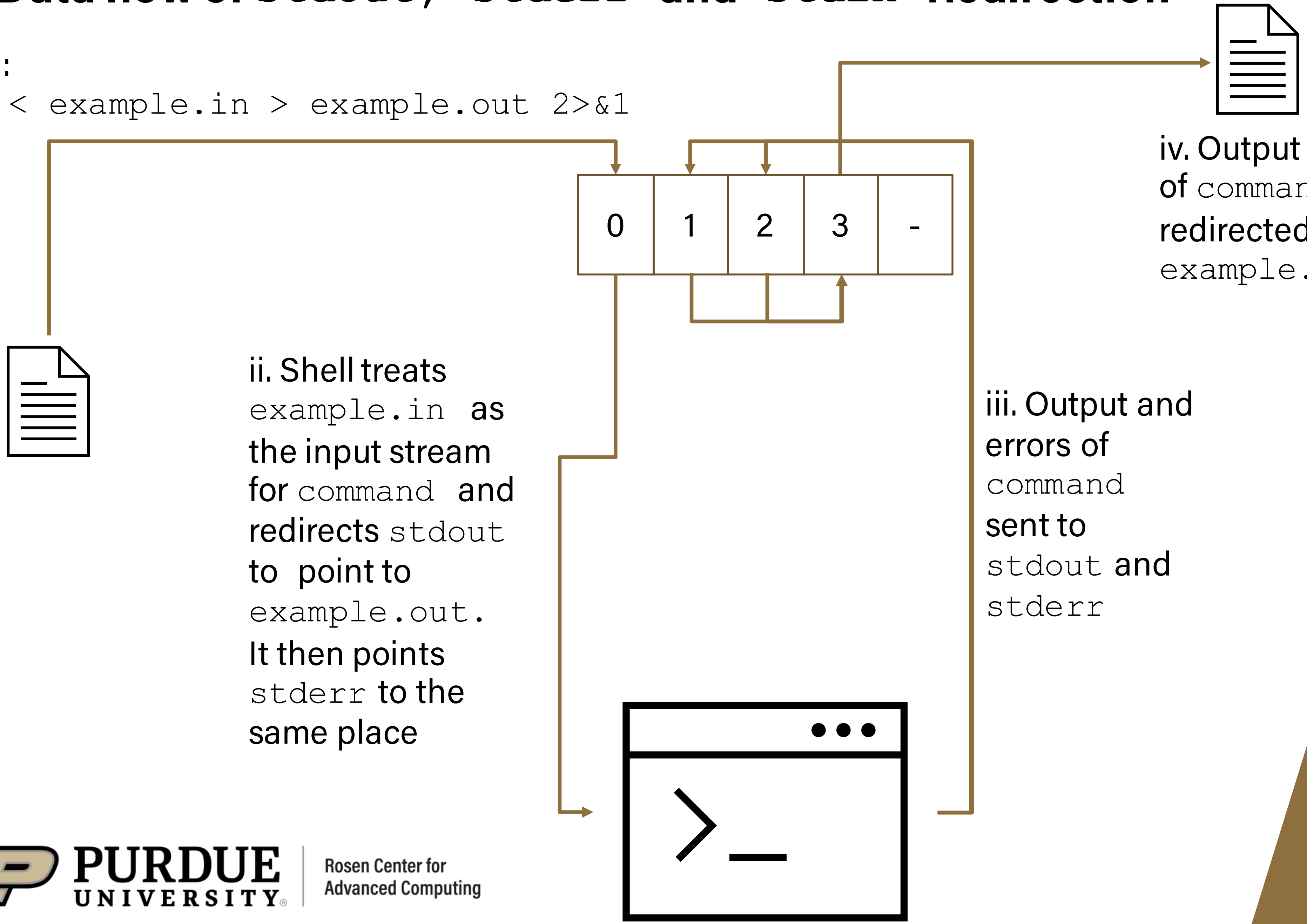
iii. Output and errors of `command` sent to `stdout` and `stderr`

iv. Output and error of `command` is redirected to `example.out`

Data flow of `stdout`, `stderr` and `stdin` Redirection

i. You type:

```
command < example.in > example.out 2>&1
```



ii. Shell treats `example.in` as the input stream for `command` and redirects `stdout` to point to `example.out`. It then points `stderr` to the same place

iii. Output and errors of `command` sent to `stdout` and `stderr`

iv. Output and error of `command` is redirected to `example.out`

Rosen Center for Advanced Computing

File I/O in Unix

How can we use this?

- First a couple of new commands:
  1. A command for printing output: `echo`
     - Prints a given string to `stdout`
     - Usage: `echo [-options] [STRING]`
  2. A command for analyzing files: `wc`
     - Prints newline, word, and byte counts for a file
     - Usage: `wc [-options] [FILE]…`

- `echo “Hello Unix 102! Redirection is cool” > example.txt`
  - Prints the string to example.txt
- `wc < example.txt`
  - Prints the number of lines, words, and bytes in `example.txt`

Example of redirection

```
rderue@gilbreth-fe02:~/teaching/unix102 $ ls
rderue@gilbreth-fe02:~/teaching/unix102 $ echo "Hello Unix 102!
Redirection is cool." > example.txt
rderue@gilbreth-fe02:~/teaching/unix102 $ cat example.txt
    Hello Unix 102! Redirection is cool.
rderue@gilbreth-fe02:~/teaching/unix102 $ echo "Hello Unix 102!
Redirection is cool." >> example.txt
rderue@gilbreth-fe02:~/teaching/unix102 $ cat example.txt
    Hello Unix 102! Redirection is cool.
    Hello Unix 102! Redirection is cool.
rderue@gilbreth-fe02:~/teaching/unix102 $ wc example.txt
    2 12 74 example.txt
rderue@gilbreth-fe02:~/teaching/unix102 $ wc < example.txt
    2 12 74
```

Unix 102

Composing Pipelines

PURDUE UNIVERSITY | Rosen Center for Advanced Computing

Composing Pipelines

Can we send the output of a command as the input for another?

- Recall the second point of the Unix Philosophy:
  - “Write programs to work together.”
- How can we do this with redirection?
  - `command1 > file1`
  - `command2 < file1`
  - This works, but now we have a file we don’t need!
- The pipe character (`|`)
  - Syntax is: `command1 | command2 [ … | command n]`

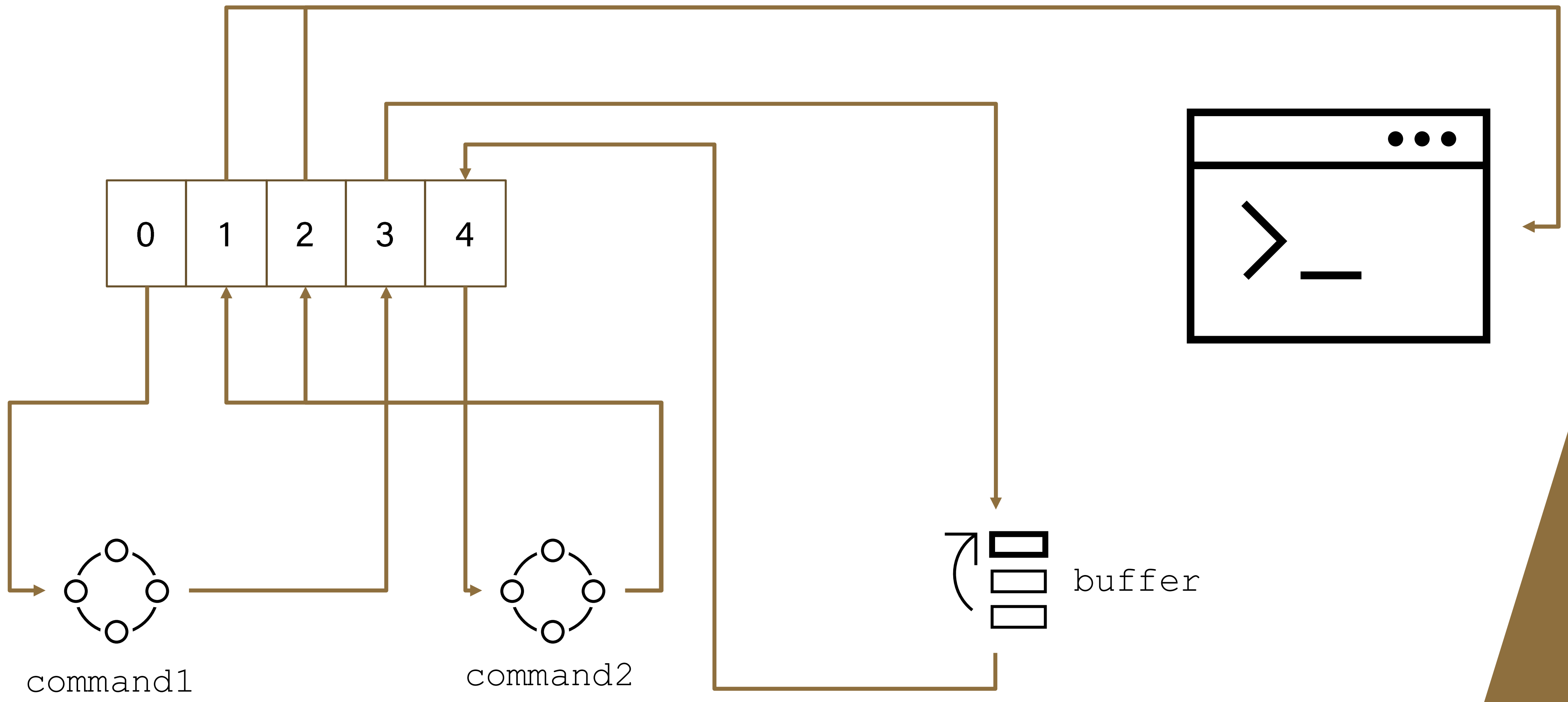
Composing Pipelines

How do Unix pipes accomplish this?

- Each command in the pipeline opens up two new file descriptors (FD)
  - For each command, one is a read FD, and the other is a write FD
- The commands in the pipeline are executed simultaneously
  - If a command need the output of the command before it to run, it will wait
- All input and output between commands is buffered
  - This can lead to pipelines stalls

Data Flow of Unix Pipes

You type: `command1 | command2`



Composing Pipelines

How can we use this?

- More new commands
  1. A command for searching output: `grep`
     - **G**lobally search for **R**egular **E**xpressions and **P**rint matching lines
     - Usage: `grep [-options] PATTERN [FILE]…`
  2. A command for modifying output: `tr`
     - **Tr**anslate or delete characters
     - Usage: `tr [-options]`
  3. A command for examining parts of output: `cut`
     - Prints selected parts of a file
     - Usage: `cut [-options] [FILE]…`

Example of Pipelining

```
rderue@gilbreth-fe02:~/teaching/unix102 $ cat jobs.log
JOBID          USER      ACCOUNT      NAME               NODES   CPUS  TIME_LIMIT
ST TIME
599478         lev   standby      SGO                  1       8      1:00:00 PD
0:00
599477         lev   standby      SGO                  1       8      1:00:00 PD
0:00
599476         lev   standby      SGO                  1       8      1:00:00 PD
0:00
599475         lev   standby      SGO                  1       8      1:00:00 PD
0:00
599474         lev   standby      SGO                  1       8      1:00:00 PD
0:00
...
```

PURDUE UNIVERSITY | Rosen Center for Advanced Computing

Example of Pipelining

```
rderue@gilbreth-fe02:~/teaching/unix102 $ cat jobs.log | grep kelley
599617        kelley  standby      job_AnisoLEO_13.    1      12      4:00:00  R
2:43:05
599618        kelley  standby      job_AnisoLEO_14.    1      12      4:00:00  R
2:43:05
599616        kelley  standby      job_AnisoLEO_12.    1      12      4:00:00  R
2:43:08
599614        kelley  standby      job_AnisoLEO_10.    1      12      4:00:00  R
2:43:11
599615        kelley  standby      job_AnisoLEO_11.    1      12      4:00:00  R
2:43:11
599613        kelley  standby      job_AnisoLEO_9.s    1      12      4:00:00  R
2:43:14
599612        kelley  standby      job_AnisoLEO_8.s    1      12      4:00:00  R
2:43:17
```

Example of Pipelining

```
rderue@gilbreth-fe02:~/teaching/unix102 $ cat jobs.log | grep kelley | tr -s " "
599617 kelley standby job_AnisoLEO_13. 1 12 4:00:00 R 2:43:05
599618 kelley standby job_AnisoLEO_14. 1 12 4:00:00 R 2:43:05
599616 kelley standby job_AnisoLEO_12. 1 12 4:00:00 R 2:43:08
599614 kelley standby job_AnisoLEO_10. 1 12 4:00:00 R 2:43:11
599615 kelley standby job_AnisoLEO_11. 1 12 4:00:00 R 2:43:11
599613 kelley standby job_AnisoLEO_9.s 1 12 4:00:00 R 2:43:14
599612 kelley standby job_AnisoLEO_8.s 1 12 4:00:00 R 2:43:17
599611 kelley standby job_AnisoLEO_7.s 1 12 4:00:00 R 2:43:21
599608 kelley standby job_AnisoLEO_4.s 1 12 4:00:00 R 2:43:28
599609 kelley standby job_AnisoLEO_5.s 1 12 4:00:00 R 2:43:28
599607 kelley standby job_AnisoLEO_3.s 1 12 4:00:00 R 2:43:31
```

Example of Pipelining

```
rderue@gilbreth-fe02:~/teaching/unix102 $ cat jobs.log | grep kelley | tr -s
" " | cut -d " " -f 1
599617
599618
599616
599614
599615
599613
599612
599611
599608
599609
599607
```

Rosen Center for
Advanced Computing

Unix 102

What Comes Next?

Rosen Center for
Advanced Computing

What Comes Next?

Upcoming Seminars

- Unix 201: February 3rd
  - Unix Processes
  - Subshells
  - Shell Variables
  - Bash Start-up Files
- Unix 202: February 10th

THANK YOU

Feel free to reach out to rderue@purdue.edu with questions.

Slides are posted at:
https://www.rcac.purdue.edu/training/unix102

PURDUE UNIVERSITY | Rosen Center for Advanced Computing