Little Languages

Lecture 16:

Shell Redirection & Pipelines

Concepts to Study - Languages and PS

- Regular Definitions & Grammars
  - Operators

    Concatenation

    | Alternation

    * Zero or More

    ? Zero or One

    + One or More
  - Ability to compare an input string with a grammar and determine whether valid or invalid
  - Ability to derive and draw parse trees

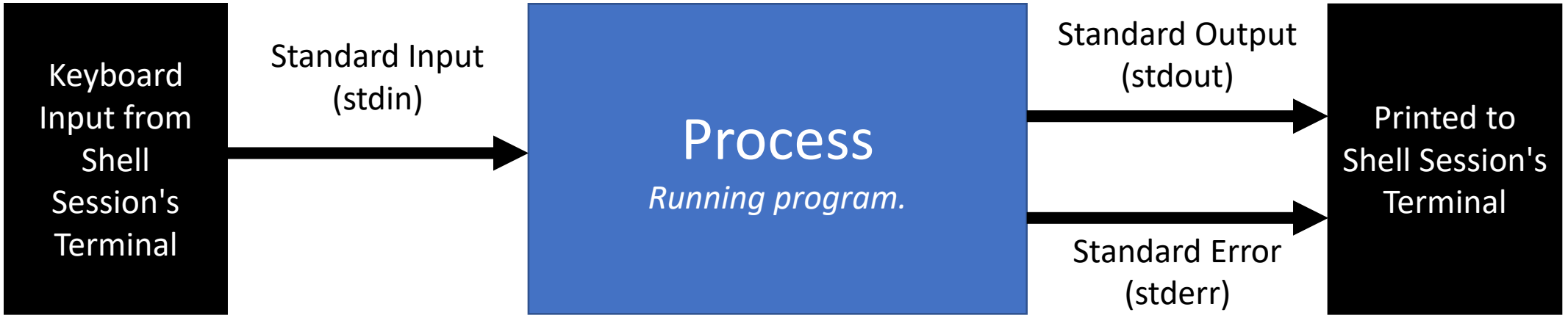
- Understanding of problem sets:
  - th**echo** - Escaping
  - th**dc** - Tokenization
  - th**dc** - Direct Interpretation
  - th**bc** - Parsing
  - th**bc** - Translation (next midterm)

- Higher level understanding of the purpose of each step and how it would translate to processing other languages.
- Lower level understanding of architecture of each program / how the components of each fit together and interact.

Concepts to Study - Systems Programming & Tools

- Programming Concepts (General)
  - Call Stack and Heap
  - null
  - Command-line Arguments
    - std::env::args() in Rust
  - Environment Variables
    - std::env::vars() in Rust
  - Test-driven Development
  - High-level understanding of arrays in C
  - Block Scope
- Programming Concepts (Rust)
  - References
  - Mutability
  - Lifetimes (high-level not syntax)
  - Standard interfaces and enums: `Iterator`, `Option<T>`, `Result<T, E>`, `Box<T>`
  - Pattern matching: if-let, while-let, match
  - struct vs. enum
- Shell and CLI Tools
  - Structure of commands (i.e. command vs. arguments)
  - File system: . vs ..
  - Purpose of $PATH
  - Purpose of commands: ls, pwd, cd, echo, which, egrep, git
- GRQs

Unix-like Process Model - Standard Input/Output Streams



- Every instance of a running program in an operating system is called a **process**.
- Each process has **standard inputs and outputs** as shown above.
- When a process is run *interactively* in a shell session, by default:
  - When the process *reads* from standard input, then you can type text input in the shell
  - When the process *writes* to standard output or error, then you will see text output in the shell

Standard Input/Output in Rust

- In the **thbc** program, the main.rs file reads **standard input** via:

```
fn read_line() -> Result<String, io::Error> {
    let mut input = String::new();
    std::io::stdin().read_line(&mut input)?;
    Ok(input)
}
```

- Writes to **standard output** via:

```
println!("{}", dc_gen::to_dc(&statement));
```

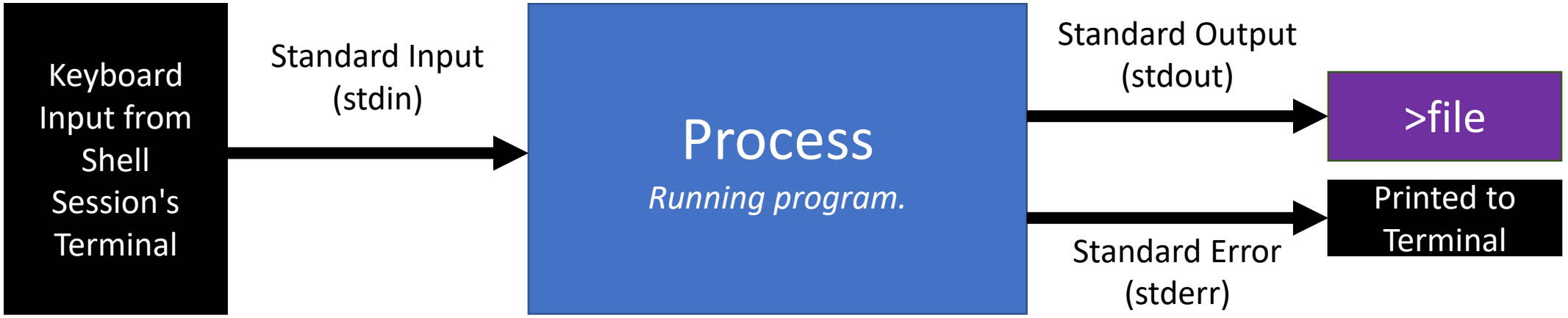
- Writes to **standard error** via (notice the leading e):

```
eprintln!("thbc: {}", msg)
```

Redirection

- Overview of redirection:
  - https://youtu.be/XvDZLjaCJuw?t=254
  - 4:15 to 11:07

Standard Output Redirection - The **>** Operator

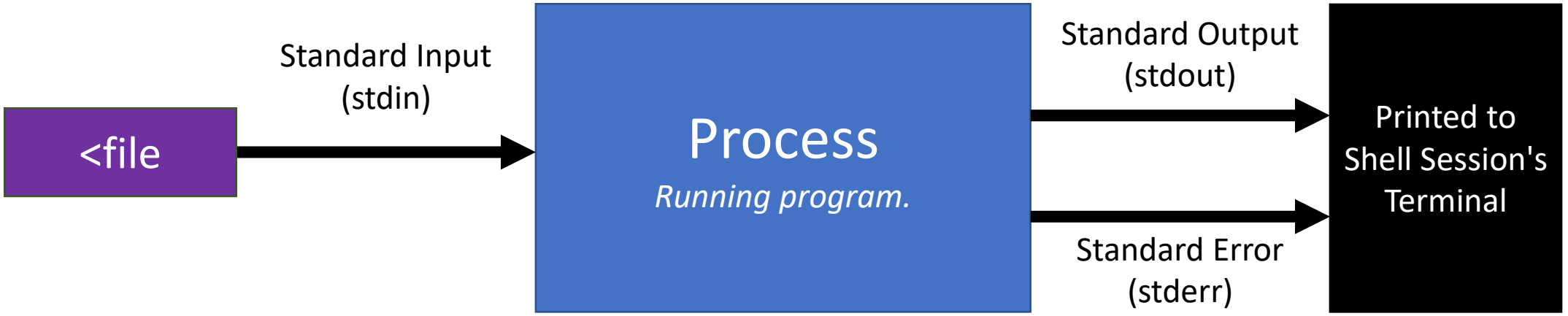


- When you want a process' output to be saved in a file:
- Use the **output redirection >** operator followed by a file path.

```
$ echo '1 2 + p' >formula.txt
```

Begin a process for the echo program and give it '1 2 + p' as an argument. Redirect its output to a blank new file named formula.txt.

Standard Input Redirection - The < Operator

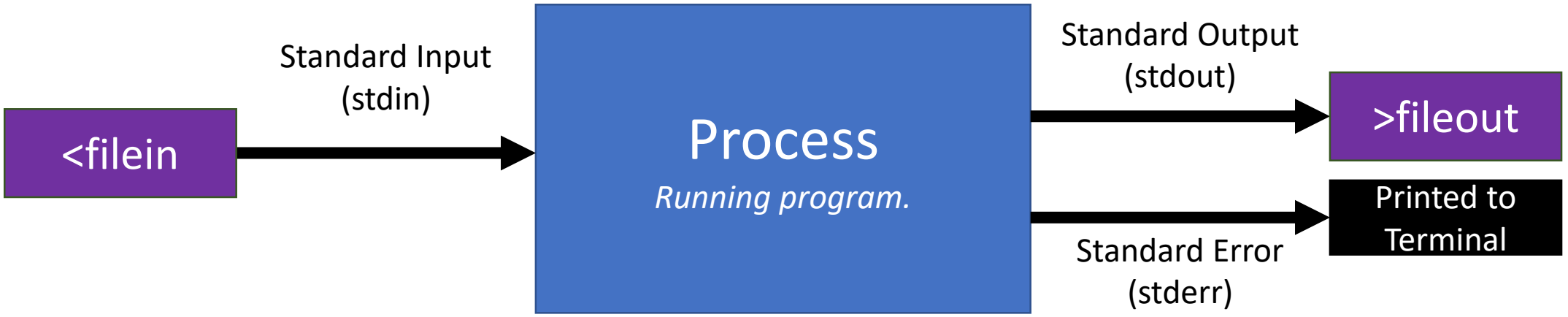


- When you want a process' input to be read in from a file:
- Use the **input redirection <** operator followed by a file path.

```
$ dc <formula.txt
```

Begin a process for the dc program and use the contents of formula.txt as its standard input.

Standard I/O Redirection - Composition



- You can combine both operators, as well:

`$ dc <formula.txt >solution.txt`

Begin a process for the dc program and use the contents of formula.txt as its standard input. Write its output to the file solution.txt.

Standard I/O Redirection - Notes and Disclaimers

- ***Every single concept here applies to every program run in a shell regardless of what programming language it was written in*.**

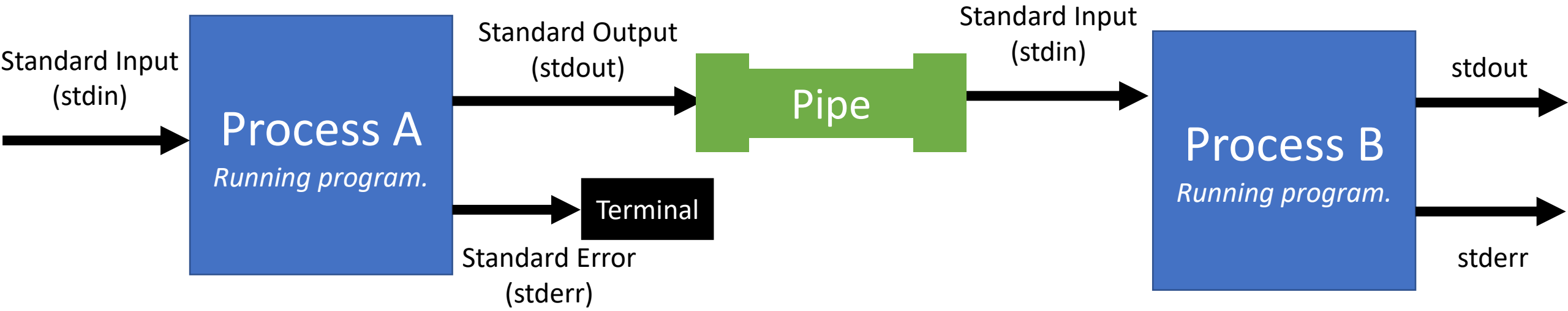
- You can *append* standard output to a file rather than *overwrite* it with the append redirection operator: **>>**
  - Example: **`$ dc <formula-2.txt >>solution.txt`**

- You can also redirect **stderr** with the cryptic looking **2>** redirection operator. We'll learn what that 2 is when we learn about file handle ids.
  - Spoiler: the single > symbol is a convenient shorthand for 1> where 1 is stdout's id.

Pipelines

- https://youtu.be/XvDZLjaCJuw?t=798
- 13:18 to 16:42

Standard I/O - Pipelines



- **Piping** connects the **stdout** of one process to the **stdin** of another.
- In **bash**, the **pipe operator** is the single vertical bar **|**

```
$ echo '1 2+p' | dc
```

Begin processes for echo and then pipe its standard output into the standard input of another new process begun for dc.

Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".

Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. [..] Don't insist on interactive input.

Doug McIlroy
1978 Bell System Technical Journal
Inventor of Unix Pipelines