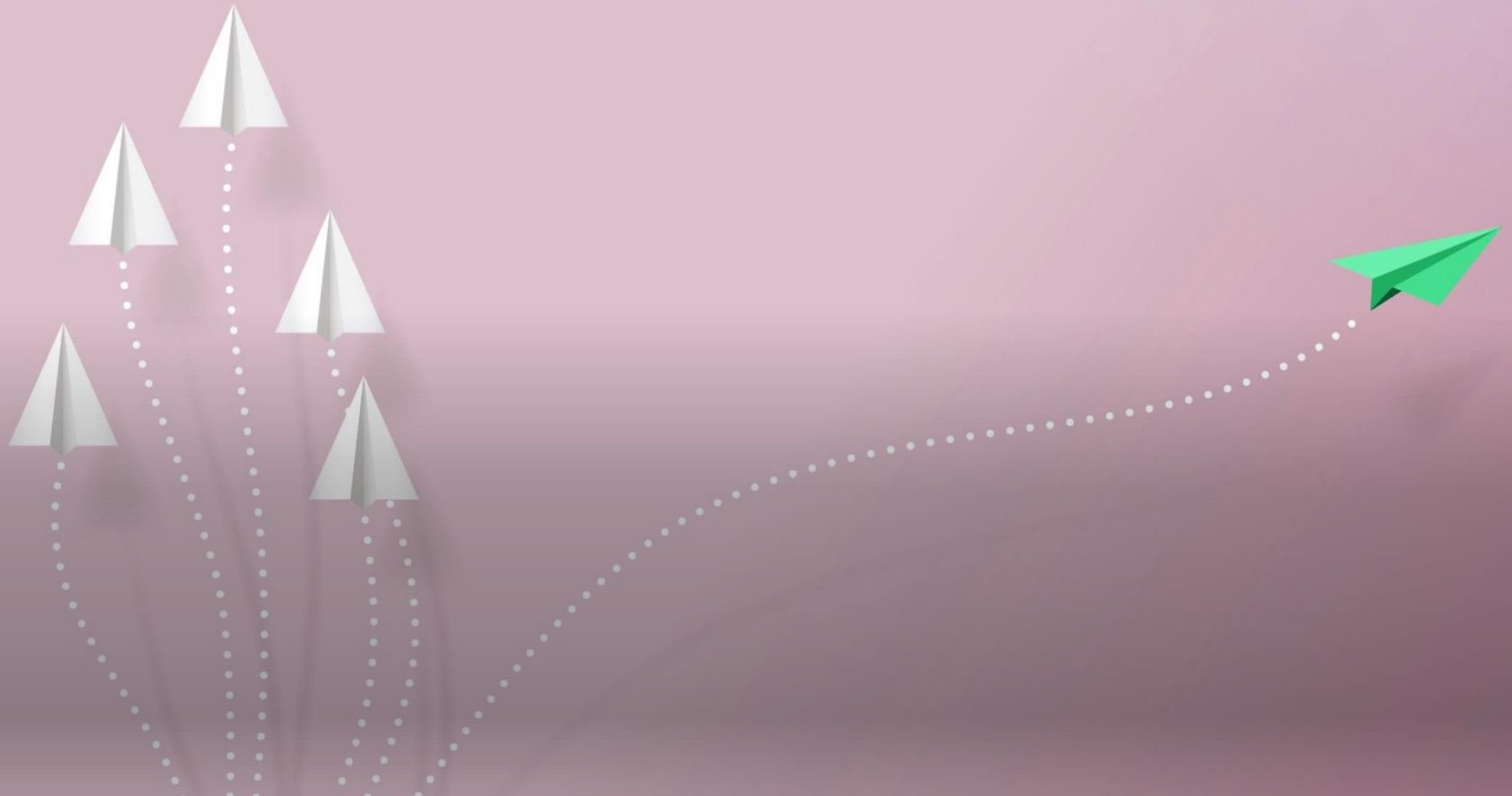
Porting Software to Rust



Why am I here today?

To tell you a story…

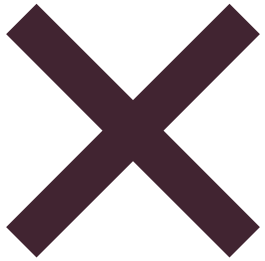
- Security
- Optimized
- Not just an OS
- Written in Rust

Tockloader

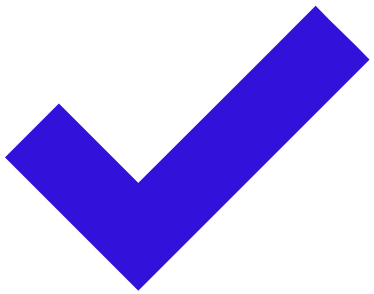
- Utility tool
- PC – kernel interface
- Written in Python

How to port software?

Line by line



Feature by feature

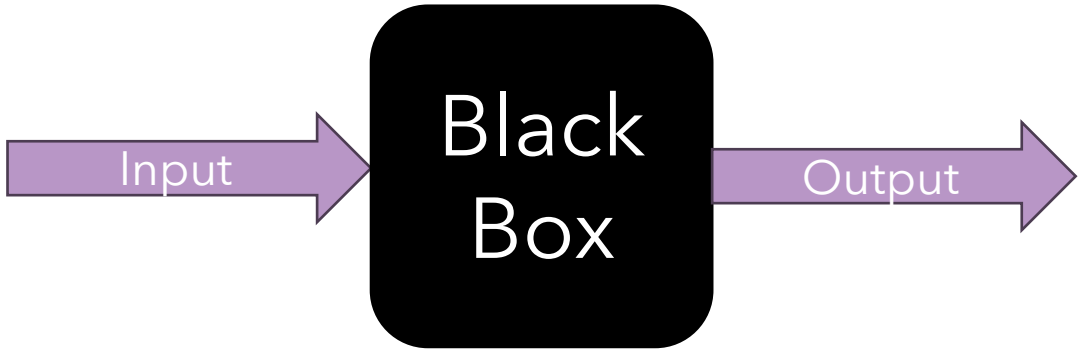


Black Box

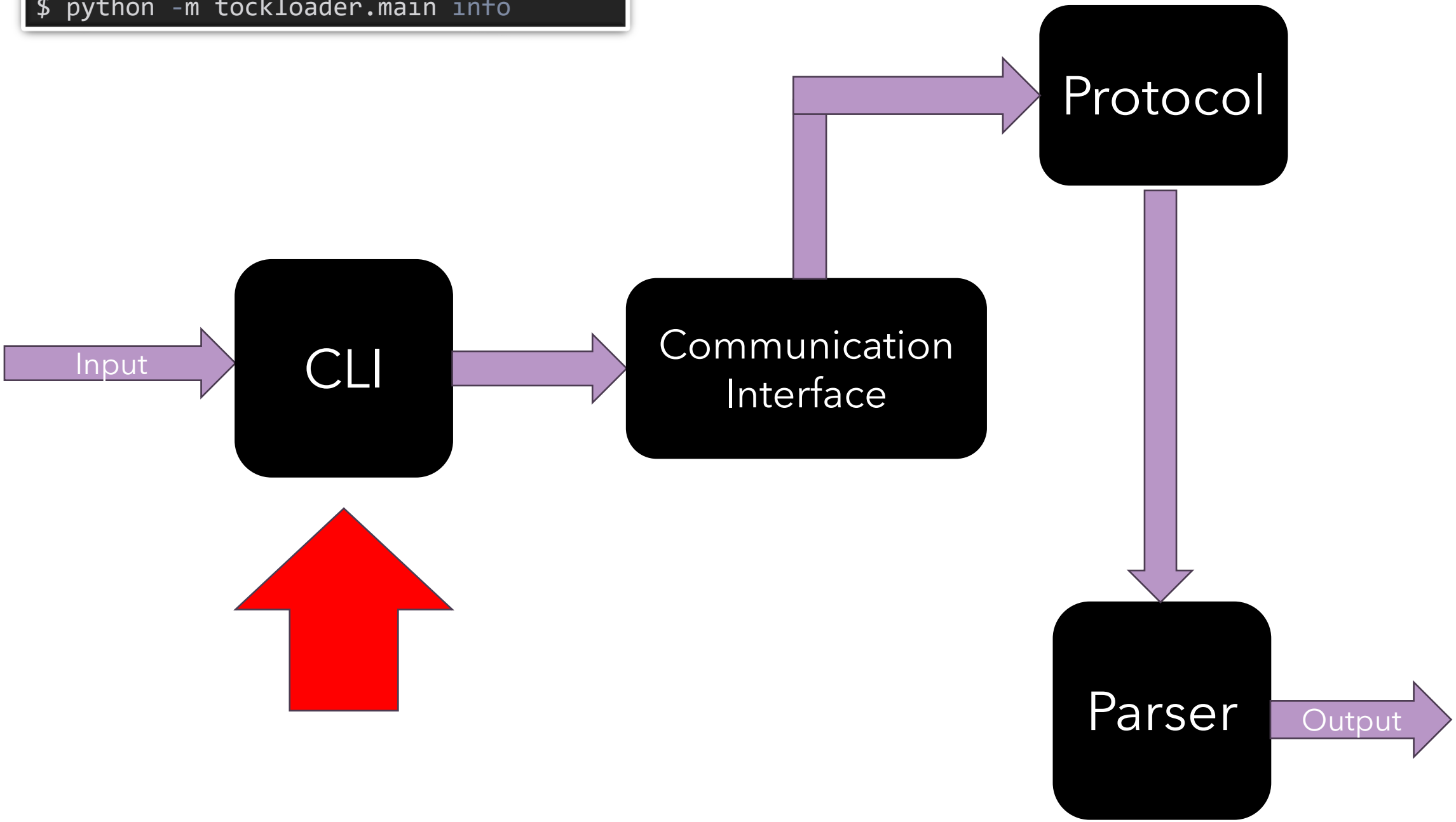
You don’t care how it was coded

You assume the user’s POV

Gives you the big picture



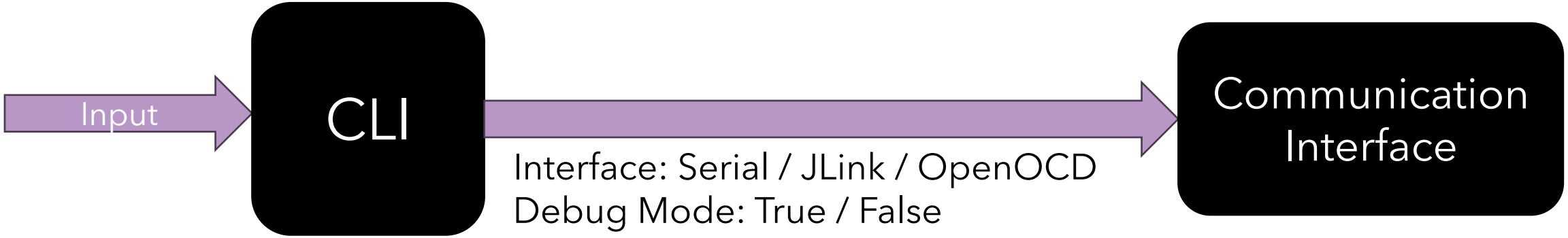
```
$ python -m tockloader.main info
```



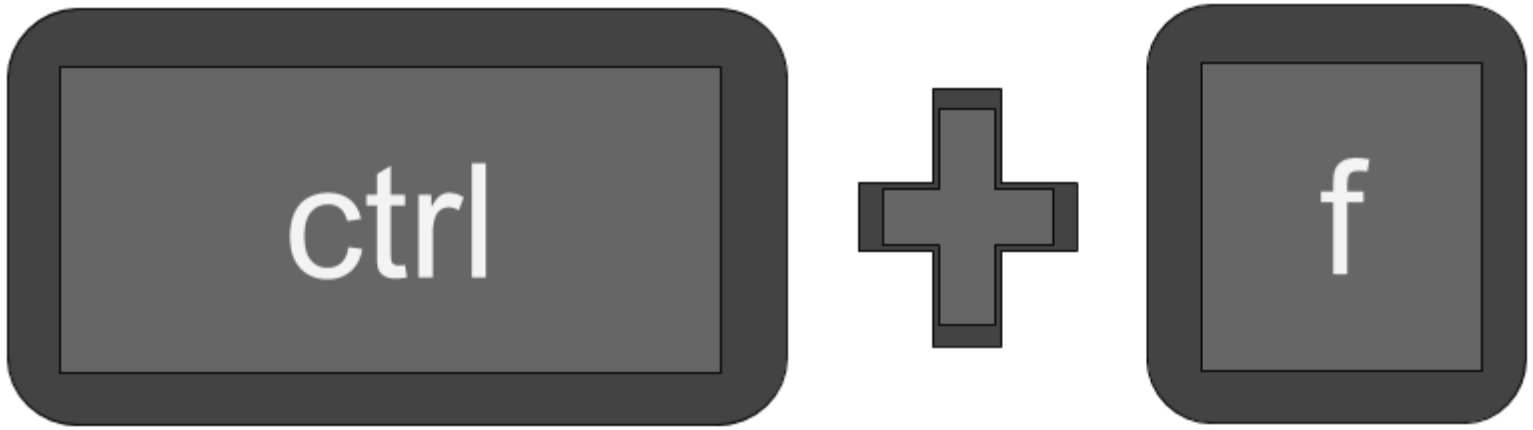
```
$ python -m tockloader.main --help
usage: main.py [-h] [--debug] [--version]  ...

options:
  -h, --help
  --debug
  --version

Commands:

    listen
    list
    info
   /* ... */
```

```
 python -m tockloader.main listen --help
usage: main.py listen /* ... */

options:
  -h, --help
  --debug
  --version
  --port STR, -p STR
  --jlink
  --openocd
  --serial
  /* ... */
```

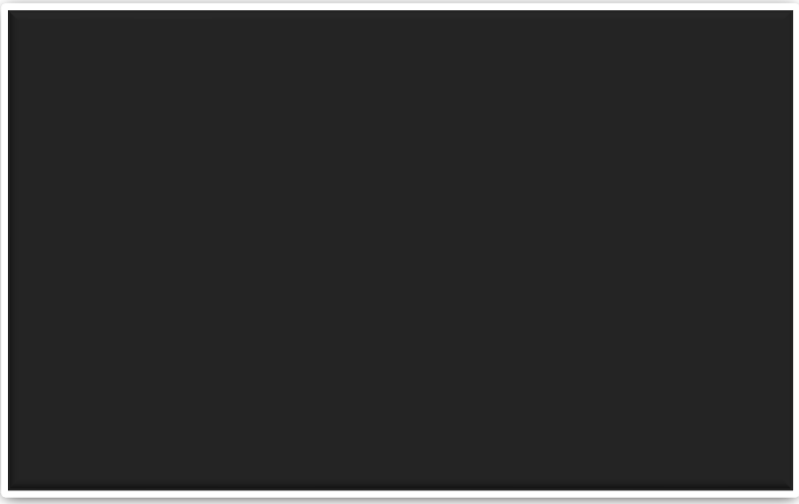
Expanding links



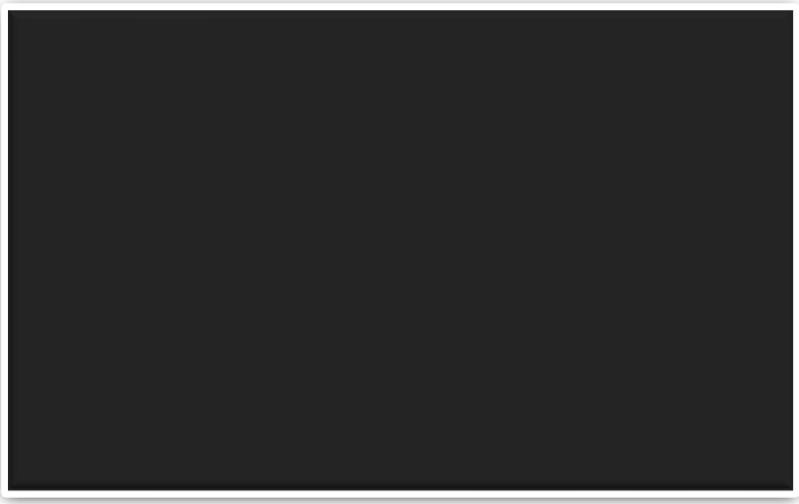
PRO TIP:



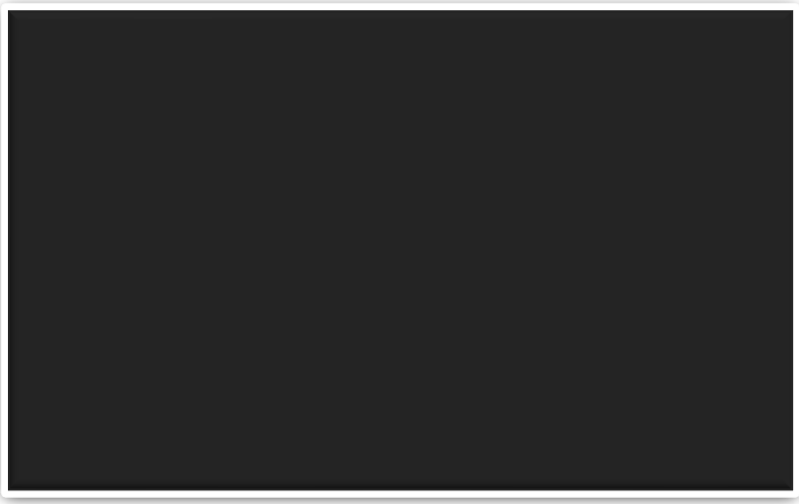
```python
parent = argparse.ArgumentParser(add_help=False)
parent.add_argument("--debug")
parent.add_argument("--version")

# The top-level parser object
parser = argparse.ArgumentParser(parents=[parent])
```



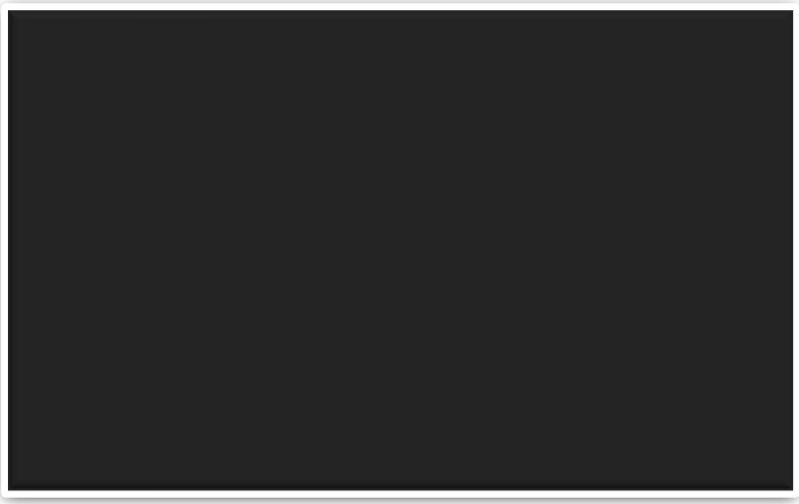
```python
parent = argparse.ArgumentParser(add_help=False)
parent.add_argument("--debug")
parent.add_argument("--version")

# The top-level parser object
parser = argparse.ArgumentParser(parents=[parent])

# Parser for all app related commands
parent_apps = argparse.ArgumentParser(add_help=False)
parent_apps.add_argument("--app-address")
# ...
```



```python
parent = argparse.ArgumentParser(add_help=False)
parent.add_argument("--debug")
parent.add_argument("--version")

# The top-level parser object
parser = argparse.ArgumentParser(parents=[parent])

# Parser for all app related commands
parent_apps = argparse.ArgumentParser(add_help=False)
parent_apps.add_argument("--app-address")
# ...

# Parser for commands that configure the communication channel
parent_channel = argparse.ArgumentParser(add_help=False)
parent_channel.add_argument("--port", "-p")
parent_channel.add_argument("--serial")
# ...
```



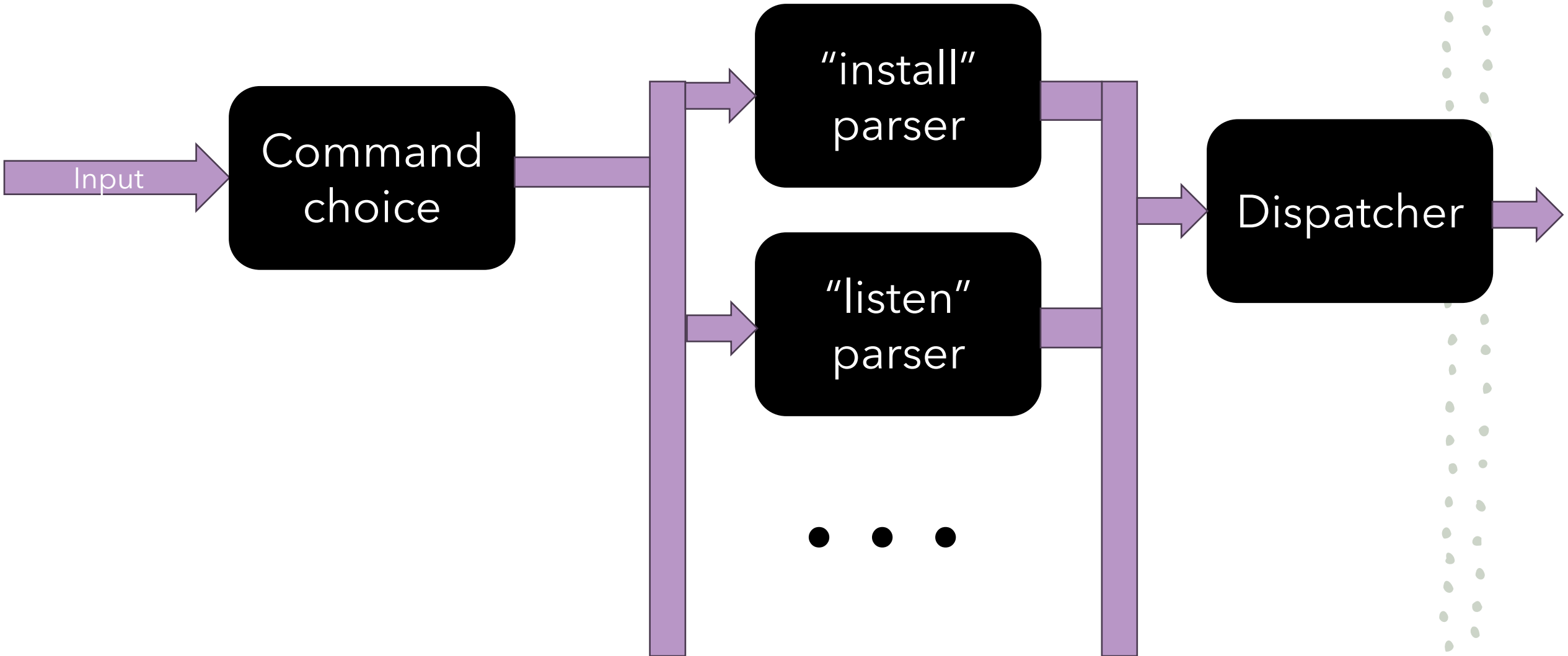
```python
parent = argparse.ArgumentParser(add_help=False)
parent.add_argument("--debug")
parent.add_argument("--version")

# The top-level parser object
parser = argparse.ArgumentParser(parents=[parent])

# Parser for all app related commands
parent_apps = argparse.ArgumentParser(add_help=False)
parent_apps.add_argument("--app-address")
# ...

# Parser for commands that configure the communication channel
parent_channel = argparse.ArgumentParser(add_help=False)
parent_channel.add_argument("--port", "-p")
parent_channel.add_argument("--serial")
# ...

install = subparser.add_parser(
  "install",
  parents=[parent, parent_apps, parent_channel],
  help="Install apps on the board",
)
```



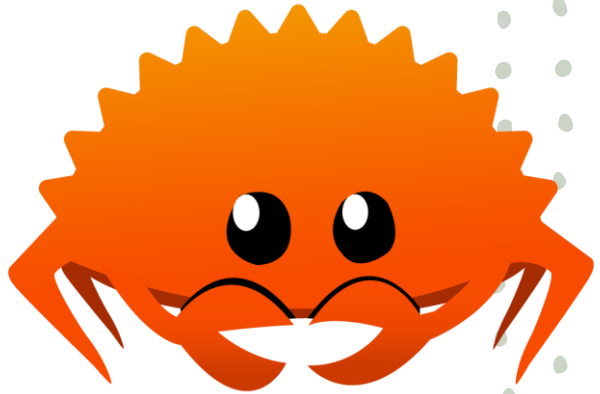
```python
parent = argparse.ArgumentParser(add_help=False)
parent.add_argument("--debug")
parent.add_argument("--version")

# The top-level parser object
parser = argparse.ArgumentParser(parents=[parent])

# Parser for all app related commands
parent_apps = argparse.ArgumentParser(add_help=False)
parent_apps.add_argument("--app-address")
# ...

# Parser for commands that configure the communication channel
parent_channel = argparse.ArgumentParser(add_help=False)
parent_channel.add_argument("--port", "-p")
parent_channel.add_argument("--serial")
# ...

install = subparser.add_parser(
  "install",
  parents=[parent, parent_apps, parent_channel],
  help="Install apps on the board",
)
```

```python
def command_install(args):
    # ...

def command_listen(args):
    # ...

# ...
```

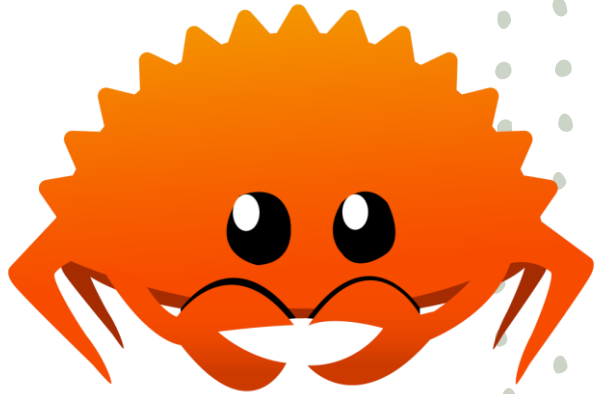
Expanding boxes



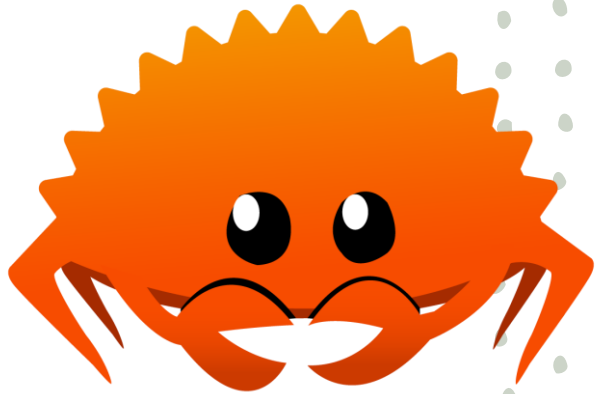
```rust
use clap::*;

pub fn make_cli() -> Command {
    Command::new("tockloader")
        .version(crate_version!())
        .subcommands(get_subcommands())
        .args([arg!(--debug "Print additional debugging information")])
}
```



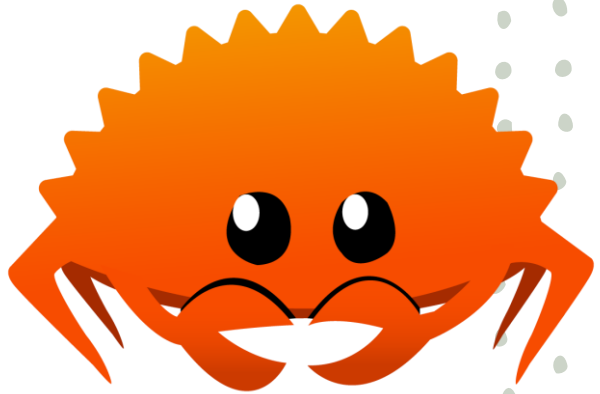
```rust
use clap::*;

pub fn make_cli() -> Command {
    Command::new("tockloader")
        .version(crate_version!())
        .subcommands(get_subcommands())
        .args([arg!(--debug "Print additional debugging information")])
}

fn get_subcommands() -> Vec<Command> {
    vec![
        Command::new("listen")
            .args(get_channel_args()),
        Command::new("info")
            .args(get_app_args())
            .args(get_channel_args()),
    ]
}
```

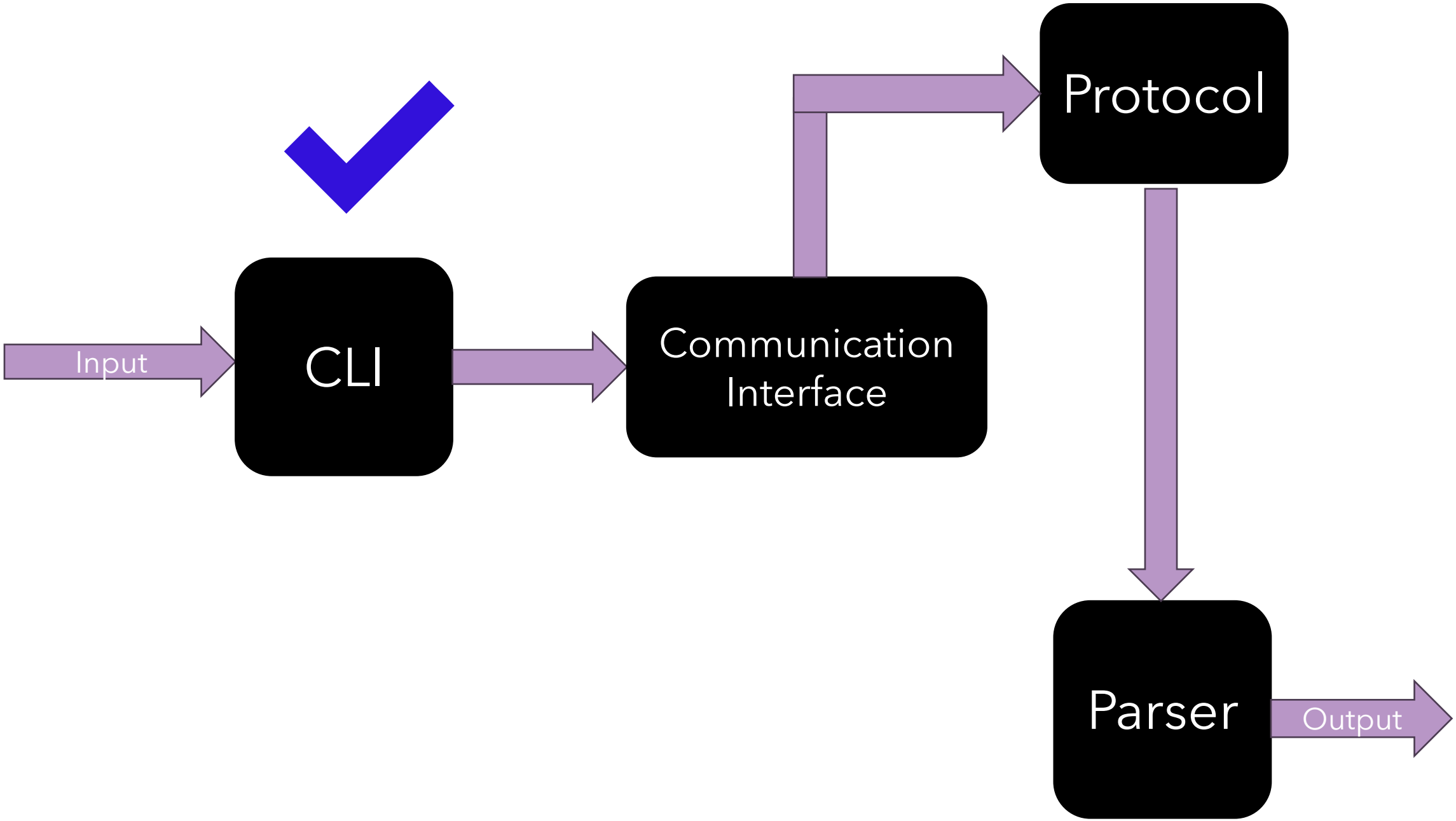


```rust
use clap::*;

pub fn make_cli() -> Command {
    Command::new("tockloader")
        .version(crate_version!())
        .subcommands(get_subcommands())
        .args([arg!(--debug "Print additional debugging information")])
}

fn get_subcommands() -> Vec<Command> {
    vec![
        Command::new("listen")
            .args(get_channel_args()),
        Command::new("info")
            .args(get_app_args())
            .args(get_channel_args()),
    ]
}

fn get_channel_args() -> Vec<clap::Arg> {
    vec![
        arg!(-p --port <PORT> "The serial port or device name to use"),
        arg!(--serial "Use the serial bootloader to flash"),
        // ...
    ]
}
```

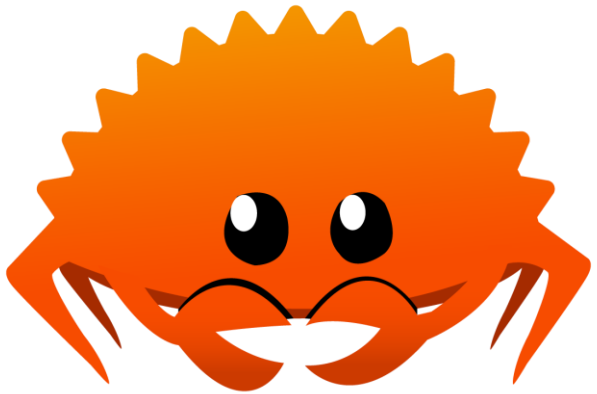


```rust
fn main() {
    let matches = make_cli().get_matches();

    match matches.subcommand() {
        Some(("listen", sub_matches)) => {
            // ...
        }
        Some(("info", sub_matches)) => {
            // ...
        }
        _ => {
            println!("Could not run the provided subcommand.");
            make_cli().print_help();
        }
    }
}
```




Protocol
Input
CLI
Communication Interface
Parser
Output

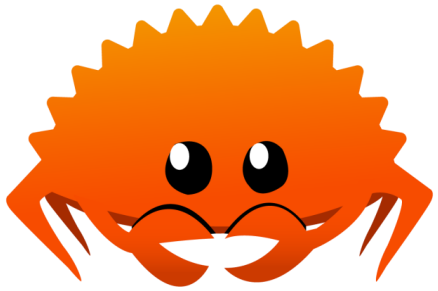
Paradigms



Paradigms

```
class TockLoader:
    """
    Implement all Tockloader commands ...
    """
def open(self):
        if self.args.jlink:
            self.channel = JLinkExe(self.args)
        elif self.args.openocd:
            self.channel = OpenOCD(self.args)
        elif self.args.serial:
            self.channel = Serial(self.args)

        self.channel.open_link_to_board()
```

```
class BoardInterface:
    def open_link_to_board(self):
        return
    # ...

class JLinkExe(BoardInterface):
    # ...

class OpenOCD(BoardInterface):
    # ...

class Serial(BoardInterface):
    # ...
```

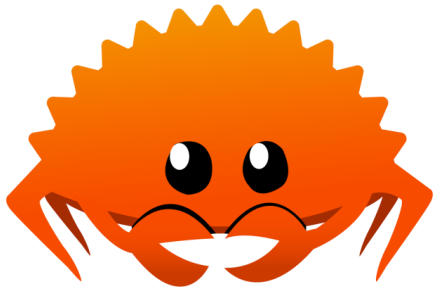


Paradigms

```rust
pub struct JLinkInterface {/* ... */}
pub struct OpenOCDInterface {/* ... */}
pub struct SerialInterface {/* ... */}

pub enum Interface {
    Serial(SerialInterface),
    OpenOCD(OpenOCDInterface),
    JLink(JLinkInterface),
}
```

```rust
pub trait BoardInterface {
    fn open_link_to_board(&mut self)
        -> Result<(), TockloaderError>;
}

impl BoardInterface for JLinkInterface { ... }
impl BoardInterface for OpenOCDInterface { ... }
impl BoardInterface for SerialInterface { ... }
```

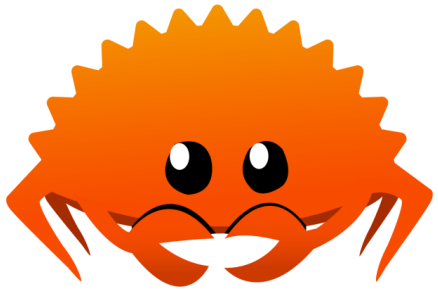
```rust
let mut interface = Interface::Serial(SerialInterface::new(args));

interface.open_link_to_board()?;
```



Paradigms

```rust
let mut interface = Interface::Serial(SerialInterface::new(args));

interface.open_link_to_board()?;
```

```
error[E0599]: no method named `open_link_to_board` found for enum `Interface`
in the current scope
  --> src\main.rs:42:15
   |
9  | pub enum Interface {
   | ------------------ method `open_link_to_board` not found for this enum
...
42 |     interface.open_link_to_board()?;
   |               ^^^^^^^^^^^^^^^^^^ method not found in `Interface`
```

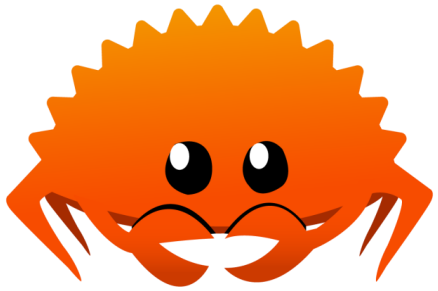


Paradigms

```rust
let mut interface = Interface::Serial(SerialInterface::new(args));

interface.open_link_to_board()?;
```

```rust
impl BoardInterface for Interface {
    fn open_link_to_board(&mut self) -> Result<(), TockloaderError> {
        match self {
            Interface::Serial(i) => i.open_link_to_board(),
            Interface::OpenOCD(i) => i.open_link_to_board(),
            Interface::JLink(i) => i.open_link_to_board(),
        }
    }
}
```

```rust
Box<dyn BoardInterface> ?
```



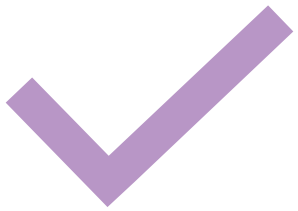
Paradigms

```rust
pub struct JLinkInterface {/* ... */}
pub struct OpenOCDInterface {/* ... */}
pub struct SerialInterface {/* ... */}

#[enum_dispatch(BoardInterface)]
pub enum Interface {
    Serial(SerialInterface),
    OpenOCD(OpenOCDInterface),
    JLink(JLinkInterface),
}
```

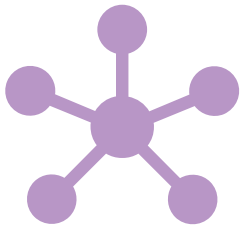
```rust
#[enum_dispatch]
pub trait BoardInterface {
    fn open_link_to_board(&mut self)
        -> Result<(), TockloaderError>;
}

impl BoardInterface for JLinkInterface { ... }
impl BoardInterface for OpenOCDInterface { ... }
impl BoardInterface for SerialInterface { ... }
```

```rust
let mut interface = Interface::Serial(SerialInterface::new(args));

interface.open_link_to_board()?;
```

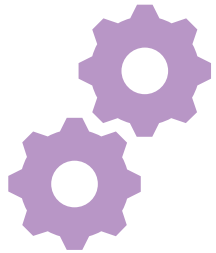
todo!()



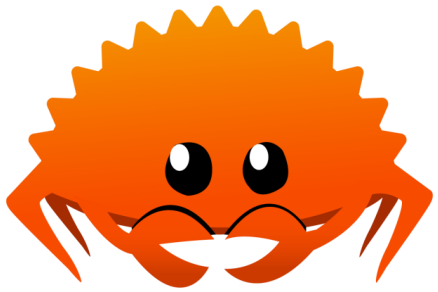
Prototype



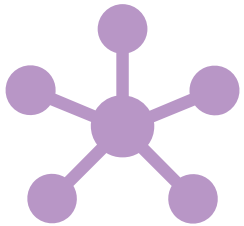
Integrate



Complete



Integrate



```rust
impl BoardInterface for SerialInterface {
    pub fn get_port(args: &ArgMatches) -> Result<String, TockloaderError> {
        if let Some(user_port) = args.get_one::<String>("port") {
            Ok(user_port.clone())
        } else {
            let available_ports = tokio_serial::available_ports()?;

            if available_ports.is_empty() {
                return Err(TockloaderError::NoPortAvailable);
            } else if available_ports.len() == 1 {
                Ok(available_ports[0].port_name.clone())
            } else { // available_ports.len() > 1
                todo!("Make user choose out of multiple available ports")
            }
        }
    }
}
```

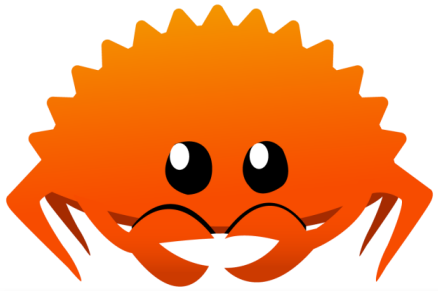
Bugs

- Bugs in original code
- Bugs in libraries
- Bugs in pipeline

Bugs

```
def command_listen(args):
    tock_loader = TockLoader(args)
    tock_loader.run_terminal()

def run_terminal(self):
    # Use trusty miniterm
    self.miniterm =
        serial.tools.miniterm.Miniterm(
            self.sp, echo=False, eol="crlf",
            filters=filters)

    # Set encoding.
    self.miniterm.set_rx_encoding("UTF-8")
    self.miniterm.set_tx_encoding("UTF-8")

    # And go!
    self.miniterm.start()
```

```
$ python3 -m tockloader.main listen
Initialization complete. Entering main loop.
tock$ help
Welcome to the process console.
Valid commands are: help status list stop start fault
boot terminate process kernel reset panic console-
start console-stop

tock$
```



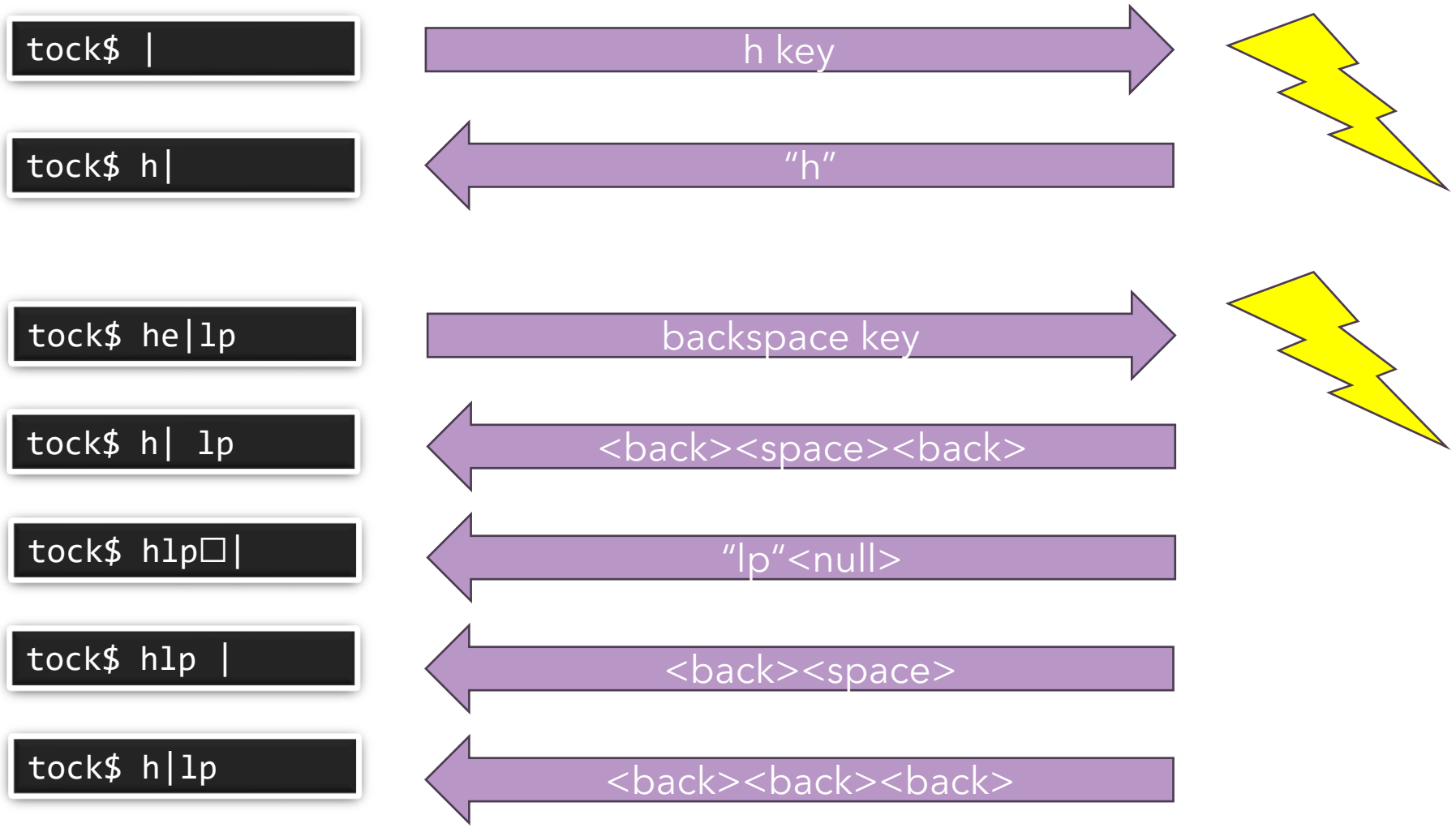
Bugs

```rust
async fn run_terminal(&mut self) -> Result<(), TockloaderError> {
    let (mut writer, mut reader) = self.stream.split();

    loop {
        if let Some(buffer) = get_key().await? {
            writer.send(buffer).await?
        }
    }
}

use console::Term;
async fn get_key() -> Result<Option<String>, TockloaderError> {
    let key = Term::stdout().read_key().await?;

    Ok(match key {
        // ...
        console::Key::Backspace => Some("\x08".into()),
        // ...
    })
}
```

```
Dec  Char
---------
  0  NUL (null)
  1  SOH (start of heading)
  2  STX (start of text)
  3  ETX (end of text)
  4  EOT (end of transmission)
  5  ENQ (enquiry)
  6  ACK (acknowledge)
  7  BEL (bell)

  8  BS  (backspace)

  9  TAB (horizontal tab)
 10  LF  (NL line feed, new line)
 11  VT  (vertical tab)
```

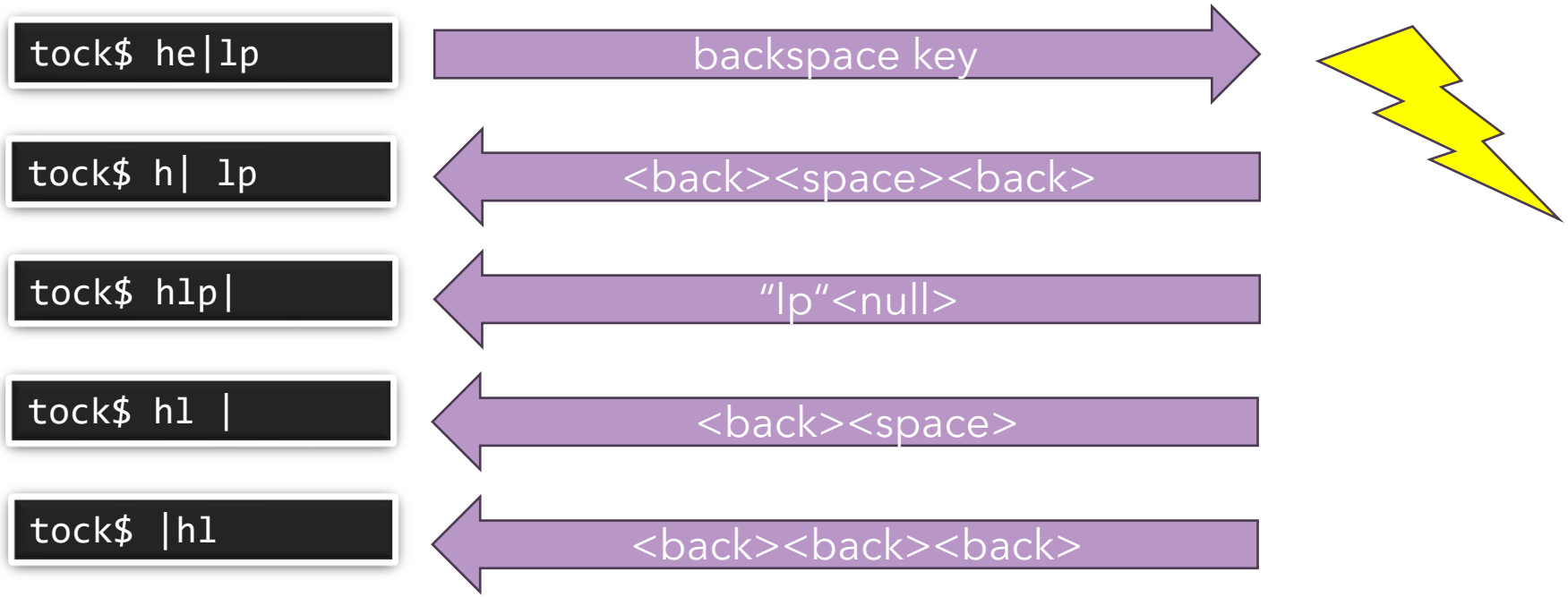
PC

Board


tock$ |
h key
tock$ h|
"h"
tock$ he|lp
backspace key
tock$ h| lp
<back><space><back>
tock$ h|p□|
"lp"<null>
tock$ hlp |
<back><space>
tock$ h|lp
<back><back><back>

PC

Board



Unicode Character
(U+0000)
<Null> (NUL)

Unicode Character "␀" (U+2400)
"Symbol For Nul"

Solutions?

Solution 1

```rust
fn clean_input(input: &str) -> String {
    input.replace('\x00', "\u{2400}")
}
```

Solution 2

FIX IT IN THE KERNEL

How to port software?

- Find the big picture
  - Black box
  - Expand
- Design the architecture
- Prototype, Integrate, Complete
- Be wary of unexpected bugs

Questions?

George Cosma
linkedin.com/in/cosma-george