poked

GNU poke beyond the CLI

Mohammad-Reza Nabipoor
mnabipoor@gnu.org

GNU Project

User Interface (UI)

This talk is about user interfaces!

User Interface (UI)

What is a UI?

The user interface (UI) is the space where interactions between humans and machines occur [1].

[1] https://en.wikipedia.org/wiki/User_interface

User Interface (UI)

Altair 8800

Figure 1: https://altairclone.com/

This is a UI: LEDs and Switches!

What is this talk all about?

In this talk I want to show you that:

- UIs should be designed to be automation-friendly.

  Automation must be a core design principle, not an after-thought.

  Good examples:

What is this talk all about?

In this talk I want to show you that:

- UIs should be designed to be automation-friendly.

  Automation must be a core design principle, not an after-thought.

  Good examples:

  - Oberon System

What is this talk all about?

In this talk I want to show you that:

- UIs should be designed to be automation-friendly.

  Automation must be a core design principle, not an after-thought.

  Good examples:

  - Oberon System
  - acme editor (Plan 9 from Bell Labs)

What is this talk all about?

In this talk I want to show you that:

- UIs should be designed to be automation-friendly.

  Automation must be a core design principle, not an after-thought.

  Good examples:

  - Oberon System
  - acme editor (Plan 9 from Bell Labs)
  - Emacs editor

What is this talk all about?

In this talk I want to show you that:

- UIs should be designed to be automation-friendly.

  Automation must be a core design principle, not an after-thought.

  Good examples:

  - Oberon System
  - acme editor (Plan 9 from Bell Labs)
  - Emacs editor

- As a consequence, integration of this program with other programs is easily achievable (Then, you are not forcing users to leave their favorite environments (terminals, editors, IDEs, web browsers, etc.) to use your program)

What is GNU poke?

```
GNU poke = libpoke  #   Poke programming language
                    #     incremental compiler (PKL)
                    # &
                    #   Poke Virtual Machine (PVM)
                    # &
                    #   IO Space (IOS)
                    #
           + poke   #   CLI (Command-Line Interface)
                    #     based on libpoke
           ;
```

How poke (the program) does look like?

- It's a REPL (Read-Evaluate-Print-Loop) program.

How poke (the program) does look like?

- It's a REPL (Read-Evaluate-Print-Loop) program.
- User can declare Poke variables, units, types, and functions.

How poke (the program) does look like?

- It's a REPL (Read-Evaluate-Print-Loop) program.
- User can declare Poke variables, units, types, and functions.
- User can compile and run statements, and see the result.

How poke (the program) does look like?

- It's a REPL (Read-Evaluate-Print-Loop) program.
- User can declare Poke variables, units, types, and functions.
- User can compile and run statements, and see the result.
- It provides some facilities to make the life of user easier (like dot commands (`.set`, `.help`, `.file`, etc.)).

How poke (the program) does look like? (Demo)

```
(poke) var a = 1
(poke) type T = struct { int i; long l; }
(poke) fun f = (int x, string y) offset<ulong,B>:
{ return x'size + y'size; } // On a single line
(poke) f (10, "Hello, binary t00ls summit!")
0x20UL#B
(poke) .set obase 10
(poke) f (10, "Hello, binary t00ls summit!")
32UL#B
(poke) .set
```

How poke (the program) works?

(Let's ignore dot commands)

How poke (the program) works?

It relies on functions from `libpoke`:

- `pk_compile_buffer`
- `pk_compile_statement`

How poke (the program) works?

It relies on functions from `libpoke`:

- `pk_compile_buffer`
- `pk_compile_statement`

If the input starts with `var`, `unit`, `type` or `fun` keywords, it uses `pk_compile_buffer`.

Otherwise, it uses the `pk_compile_statement`.

How poke (the program) works?

Example

```
// This will invoke `pk_compile_buffer` C function
(poke) var a = 1; a += 1; a += 1
```

How poke (the program) works?

Example

```
// This will invoke `pk_compile_buffer` C function
(poke) var a = 1; a += 1; a += 1

// This will invoke `pk_compile_statement` C function
(poke) a += 1
```

How poke (the program) works?

Example

```
// This will invoke `pk_compile_buffer` C function
(poke) var a = 1; a += 1; a += 1

// This will invoke `pk_compile_statement` C function
(poke) a += 1

// This is a syntax error because it expects a
// single statement but gets two.
(poke) a += 1; a += 1
```

How poke (the program) works?

Let's look inside the `poke`: the main loop

How poke (the program) works?

Over-simplified pseudo-code

```
struct input
{
  const char* first; // first word
  const char* line;  // input
} *input;

struct input *read_command_from_terminal (void);

#define STREQ(a,b) (strcmp ((a), (b)) == 0)
#define CHK(...) /* handle errors ... */
#define CHK_PRINT(...) /* handle errors and \
                          print the `val` ... */
```

How poke (the program) works?

Over-simplified pseudo-code

```
while ((input = read_command_from_terminal ())) {
  if (STREQ (input->first, "var")
      || STREQ (input->first, "unit")
      || STREQ (input->first, "type")
      || STREQ (input->first, "fun"))
    CHK (pk_compile_buffer (
      poke_compiler, input->line, /*end*/ NULL,
      &exit_exception));
  else
    CHK_PRINT (pk_compile_statement (
      poke_compiler, input->line, /*end*/ NULL,
      &val, &exit_exception));
  free (input);
}
```

Some limitations of CLI

Let's talk about limitations :)

Some limitations of CLI

- Not automation-friendly.

Some limitations of CLI

- Not automation-friendly.
  - You're limited to one `readline` input and one output.

Some limitations of CLI

- Not automation-friendly.
  - You're limited to one `readline` input and one output.
  - You cannot embed poke (the program) in your program/editor/IDE/etc.

Some limitations of CLI

- Not automation-friendly.
  - You're limited to one `readline` input and one output.
  - You cannot embed poke (the program) in your program/editor/IDE/etc.
- Not that much user-friendly.

Some limitations of CLI

- Not automation-friendly.
  - You're limited to one `readline` input and one output.
  - You cannot embed poke (the program) in your program/editor/IDE/etc.
- Not that much user-friendly.
  - Everything should be in a single line.

Some limitations of CLI

- Not automation-friendly.
  - You're limited to one `readline` input and one output.
  - You cannot embed poke (the program) in your program/editor/IDE/etc.
- Not that much user-friendly.
  - Everything should be in a single line.
  - You cannot have more than one statement per input (workaround: `load`).

Some limitations of CLI

- Not automation-friendly.
  - You're limited to one `readline` input and one output.
  - You cannot embed poke (the program) in your program/editor/IDE/etc.
- Not that much user-friendly.
  - Everything should be in a single line.
  - You cannot have more than one statement per input (workaround: `load`).
  - Utility of hex dumps are limited to the capabilities of terminal emulators.

Some limitations of CLI

- Not automation-friendly.
  - You're limited to one `readline` input and one output.
  - You cannot embed poke (the program) in your program/editor/IDE/etc.
- Not that much user-friendly.
  - Everything should be in a single line.
  - You cannot have more than one statement per input (workaround: `load`).
  - Utility of hex dumps are limited to the capabilities of terminal emulators.
- ...

The solution

`poked`

The solution: `poked`

- `poked` is a daemon (background process)

The solution: `poked`

- `poked` is a daemon (background process)
- It listens on a unix socket (default: `poked.ipc`)

The solution: `poked`

- `poked` is a daemon (background process)
- It listens on a unix socket (default: `poked.ipc`)
- It has a notion of channels (input channels and output channels)

The solution: `poked`

- `poked` is a daemon (background process)
- It listens on a unix socket (default: `poked.ipc`)
- It has a notion of channels (input channels and output channels)
- Input/Output channels are independent.

The solution: `poked`

- `poked` is a daemon (background process)
- It listens on a unix socket (default: `poked.ipc`)
- It has a notion of channels (input channels and output channels)
- Input/Output channels are independent.
- Each connection can only has one *role* (either write (publish) to input channel or read from (subscribe to) output channel).

The solution: `poked`

- `poked` is a daemon (background process)
- It listens on a unix socket (default: `poked.ipc`)
- It has a notion of channels (input channels and output channels)
- Input/Output channels are independent.
- Each connection can only has one *role* (either write (publish) to input channel or read from (subscribe to) output channel).
- Many clients (pokelets) can simultaneously interact with `poked`.

poked

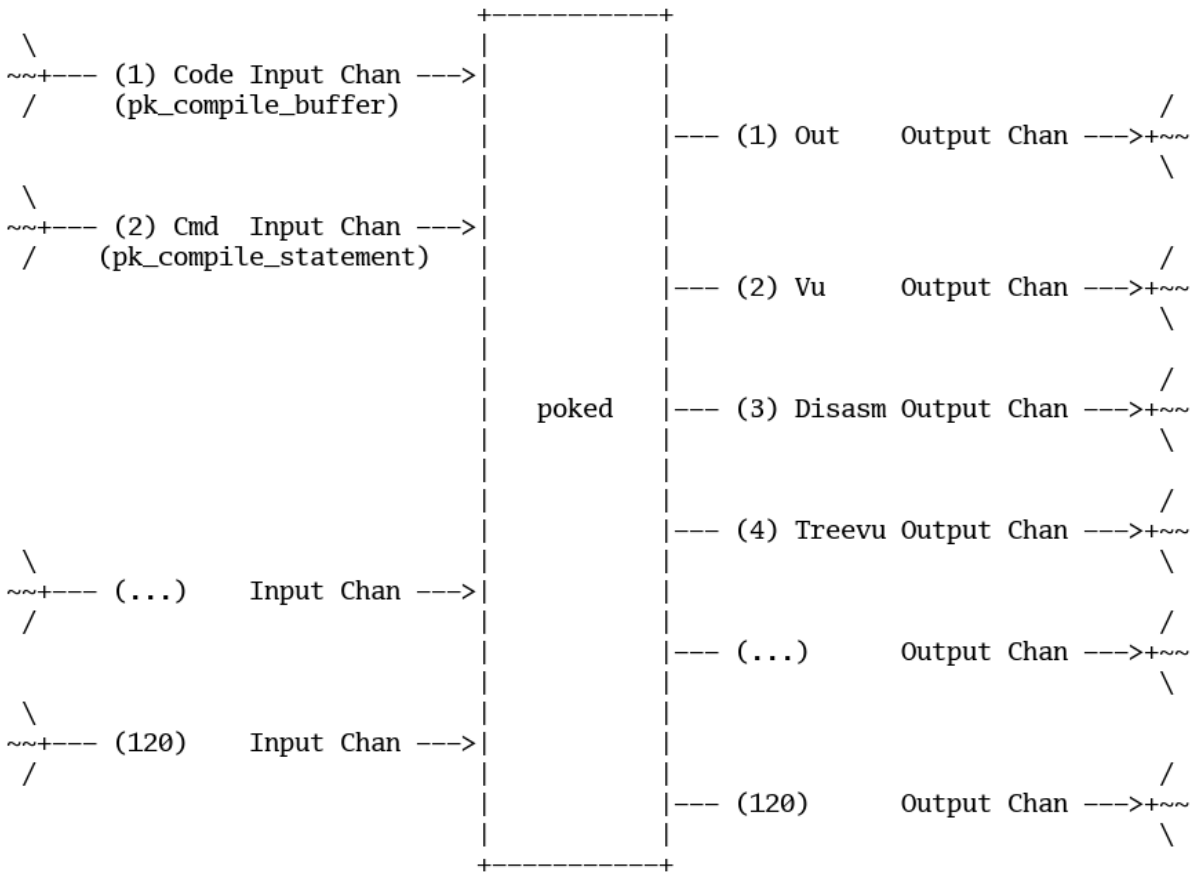


Figure 2: poked

poked, a simple message broker

If you are familiar with MQTT, you'll see some similarities:

MQTT	poked
topic	channel
publisher	pokelet
subscriber	pokelet

When a **publisher**, publishes something on a **topic**, all **subscribers** will receive it. More than one **publisher** can publish data on **topic**s.

How poked works?

Let's look inside the poked: the main loop

How poked works?

Over-simplified pseudo-code (I)

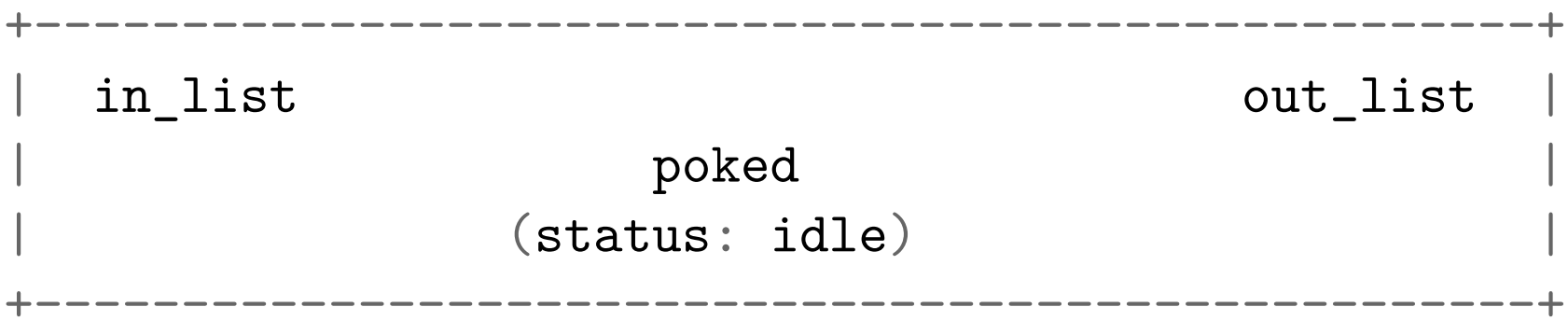
```
struct input
{
  struct chan chan;
  const char* data; // null-terminated string
                    // of Poke code
} *input;
struct input *read_from_input_channels (void);
#define CHAN_IS_INPUT_P(chan) ...
#define CHAN_NUM(chan) ...
#define CHK(...) /* handle errors ... */
#define CHK_PUBLISH(...) /* handle errors and \
                            publish the `val` ... */
static size_t n_iter;
```

How poked works?

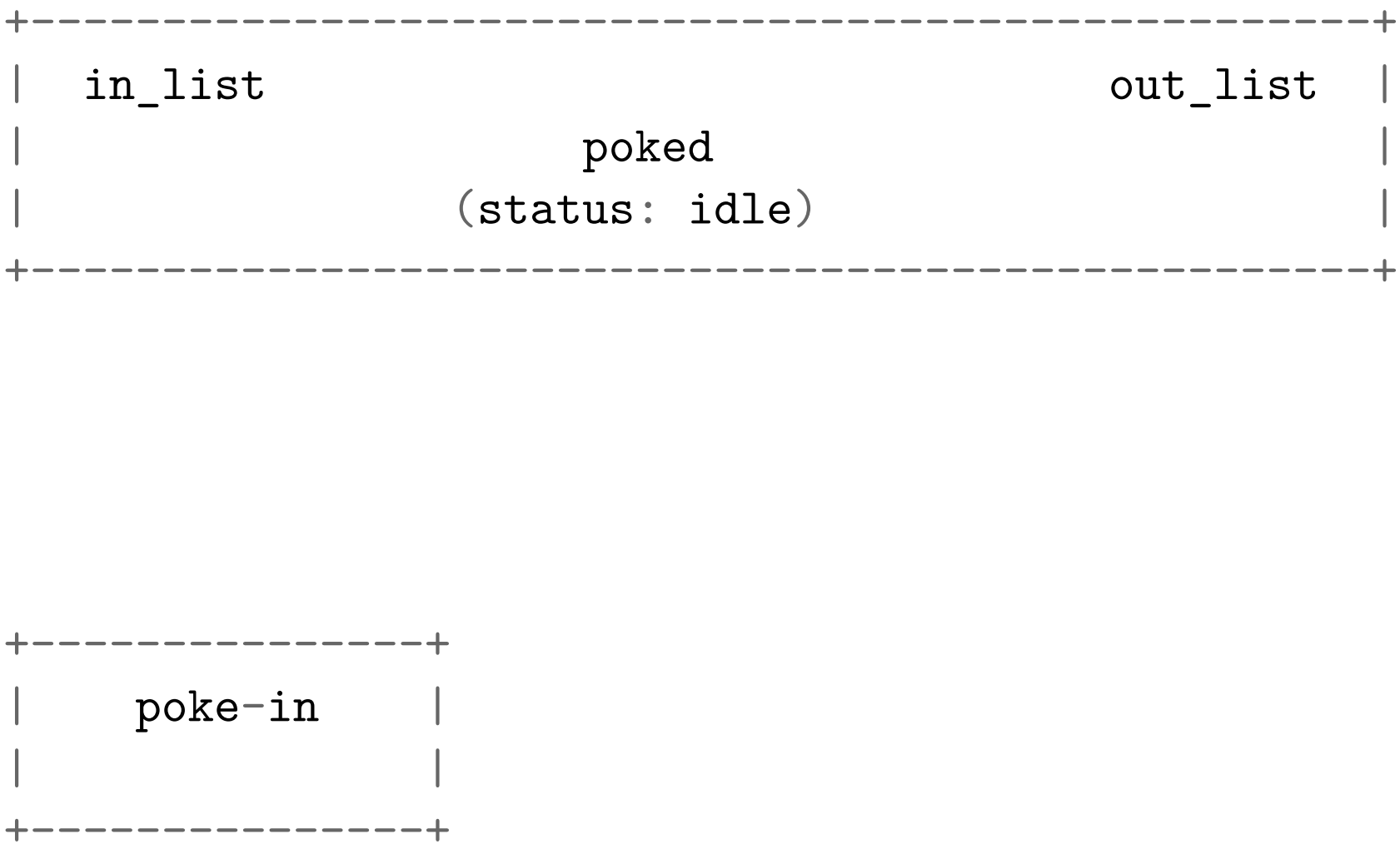
Over-simplified pseudo-code (II)

```
while ((input = read_from_input_channels ())) {
  ++n_iter; /* also publish a new iteration msg */
  if (CHAN_NUM(input->chan) == CHAN_INPUT_CODE)
    CHK (pk_compile_buffer (
      poke_compiler, input->data,
      /*end*/ NULL, &exit_exception));
  else if (CHAN_NUM(input->chan) == CHAN_INPUT_CMD)
    CHK_PUBLISH (pk_compile_statement (
      poke_compiler, input->data, /*end*/ NULL,
      &val, &exit_exception));
  else /* ignore */;
  free (input);
}
```

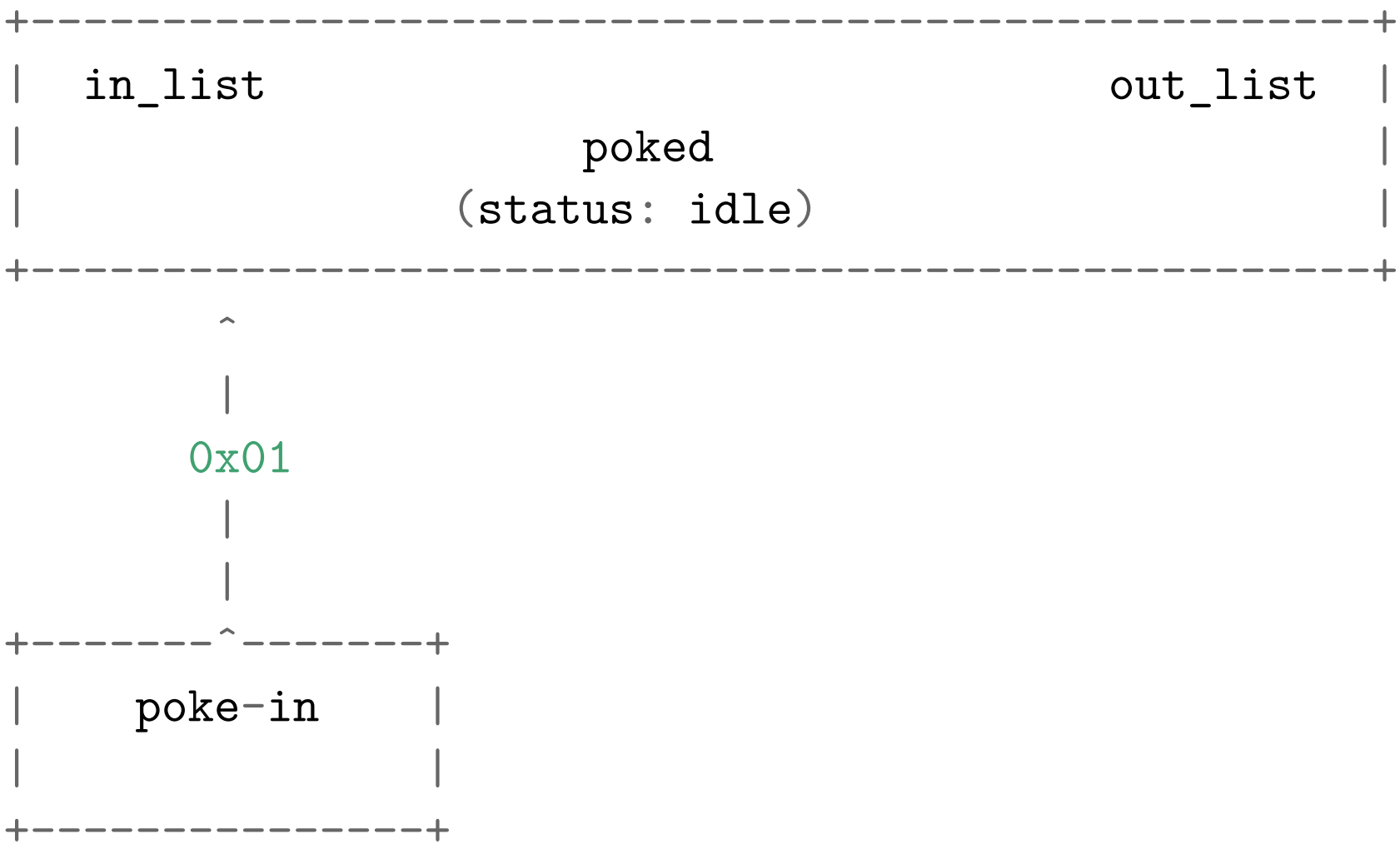
Example: poked & pokelet poke-in & pokelet poke-out



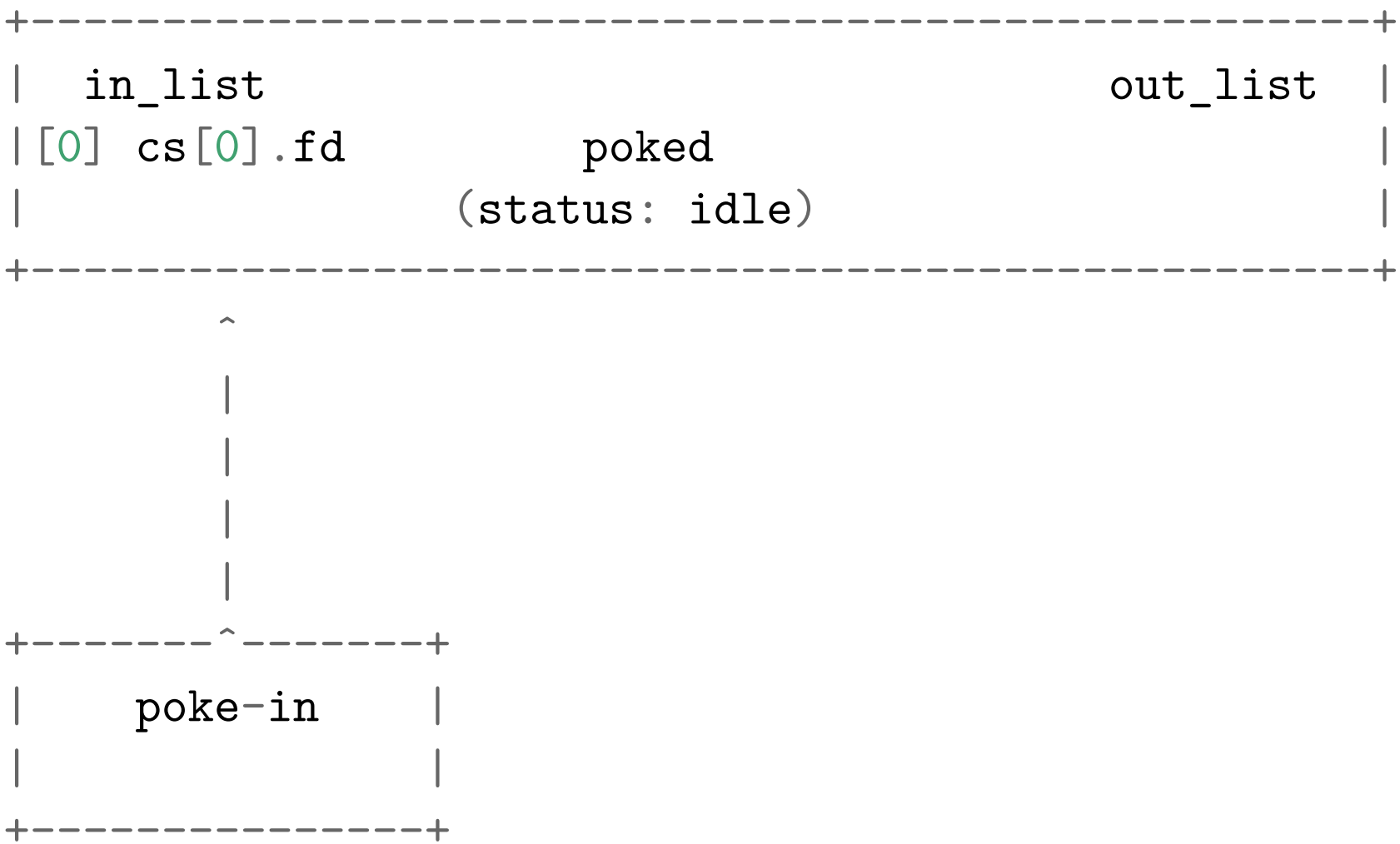
Example: poked & pokelet poke-in & pokelet poke-out



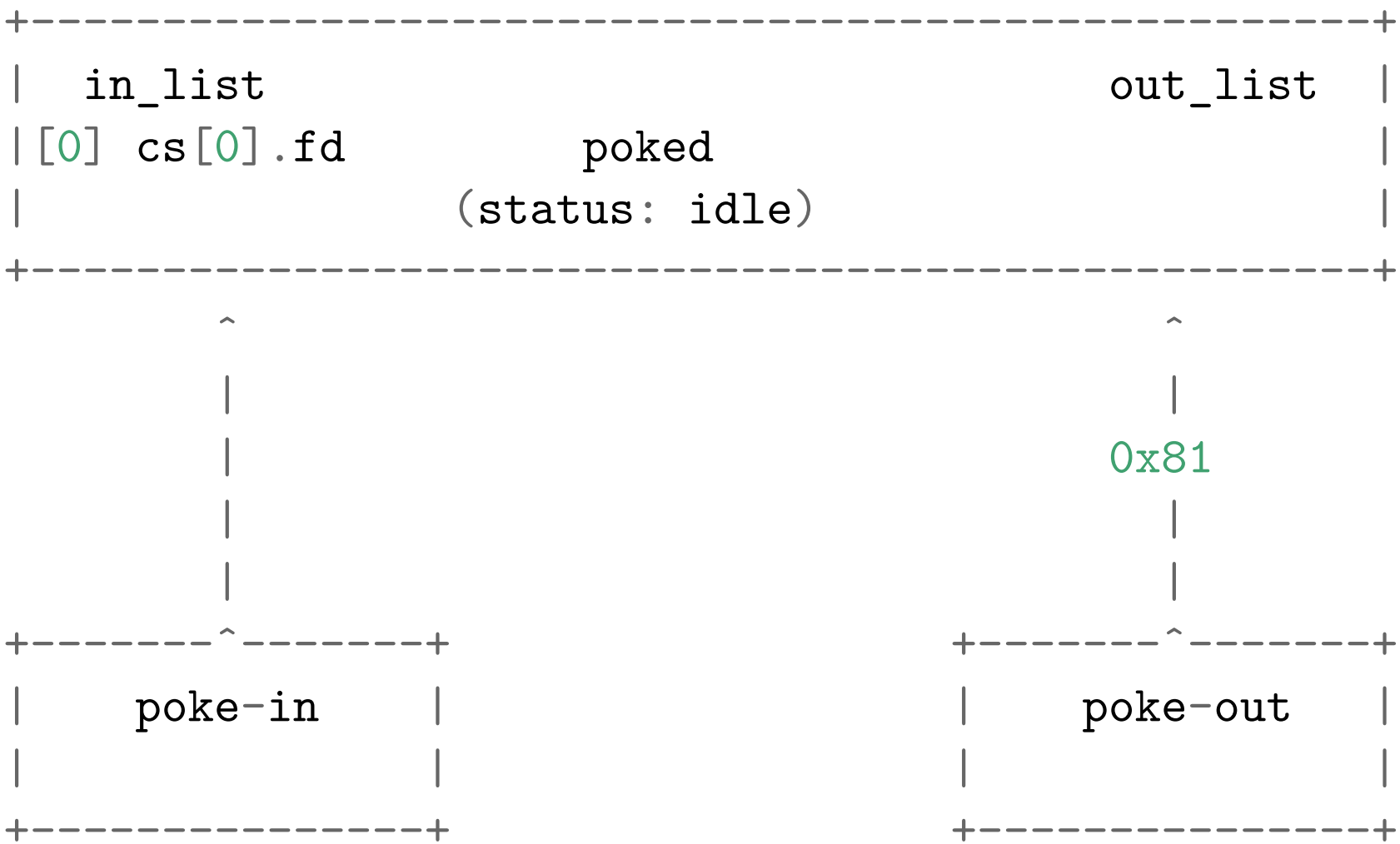
Example: poked & pokelet poke-in & pokelet poke-out



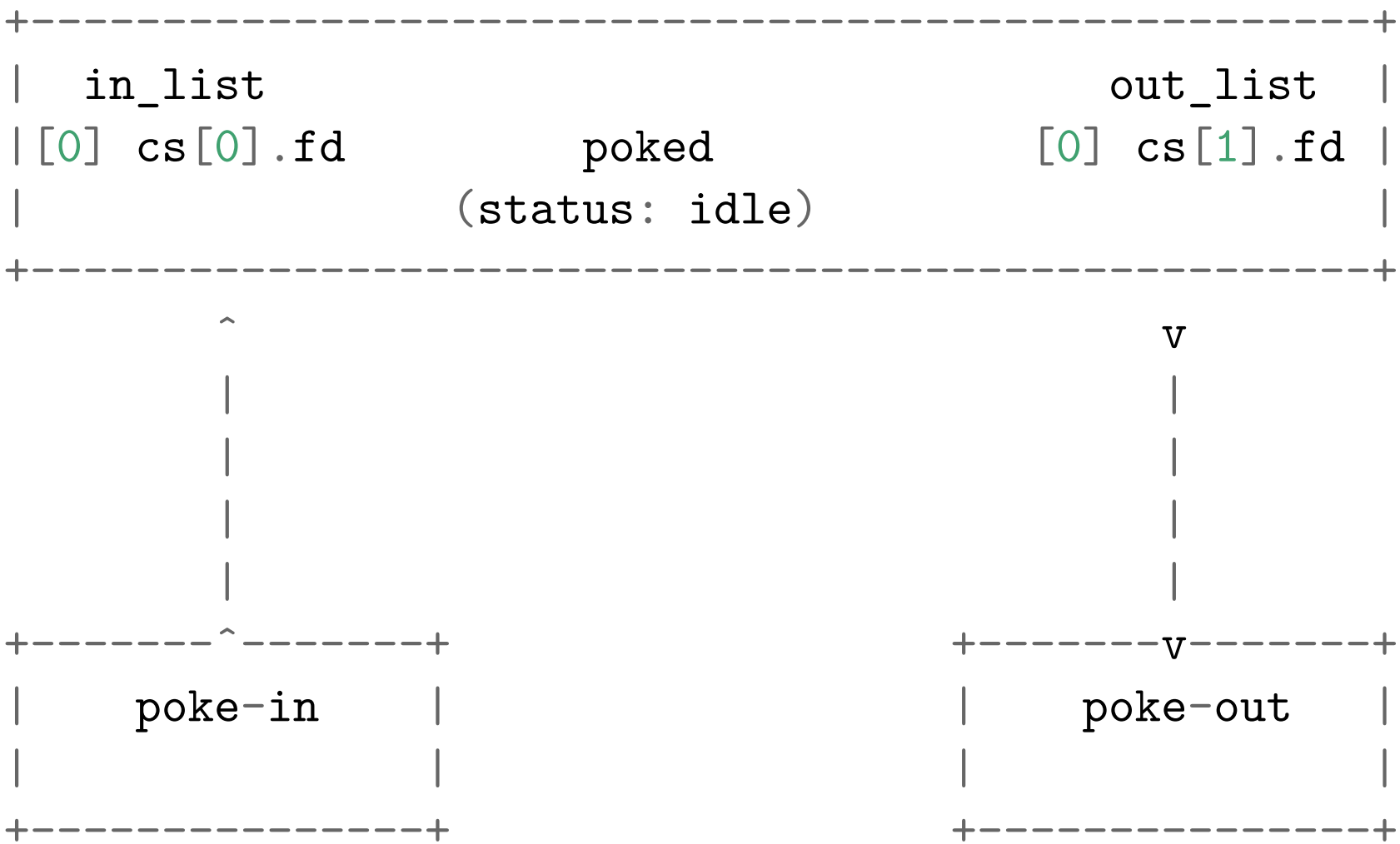
Example: poked & pokelet poke-in & pokelet poke-out



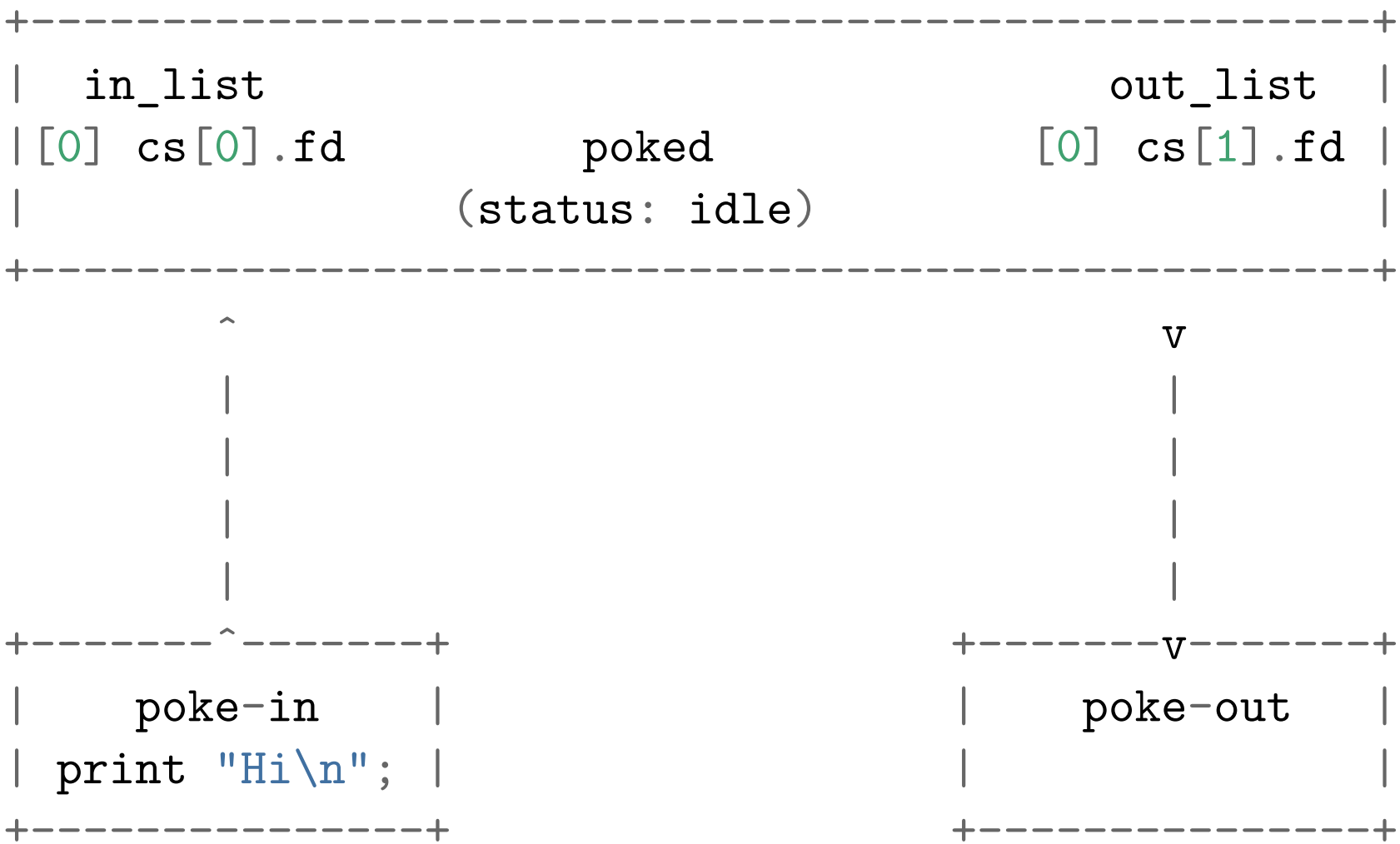
Example: poked & pokelet poke-in & pokelet poke-out



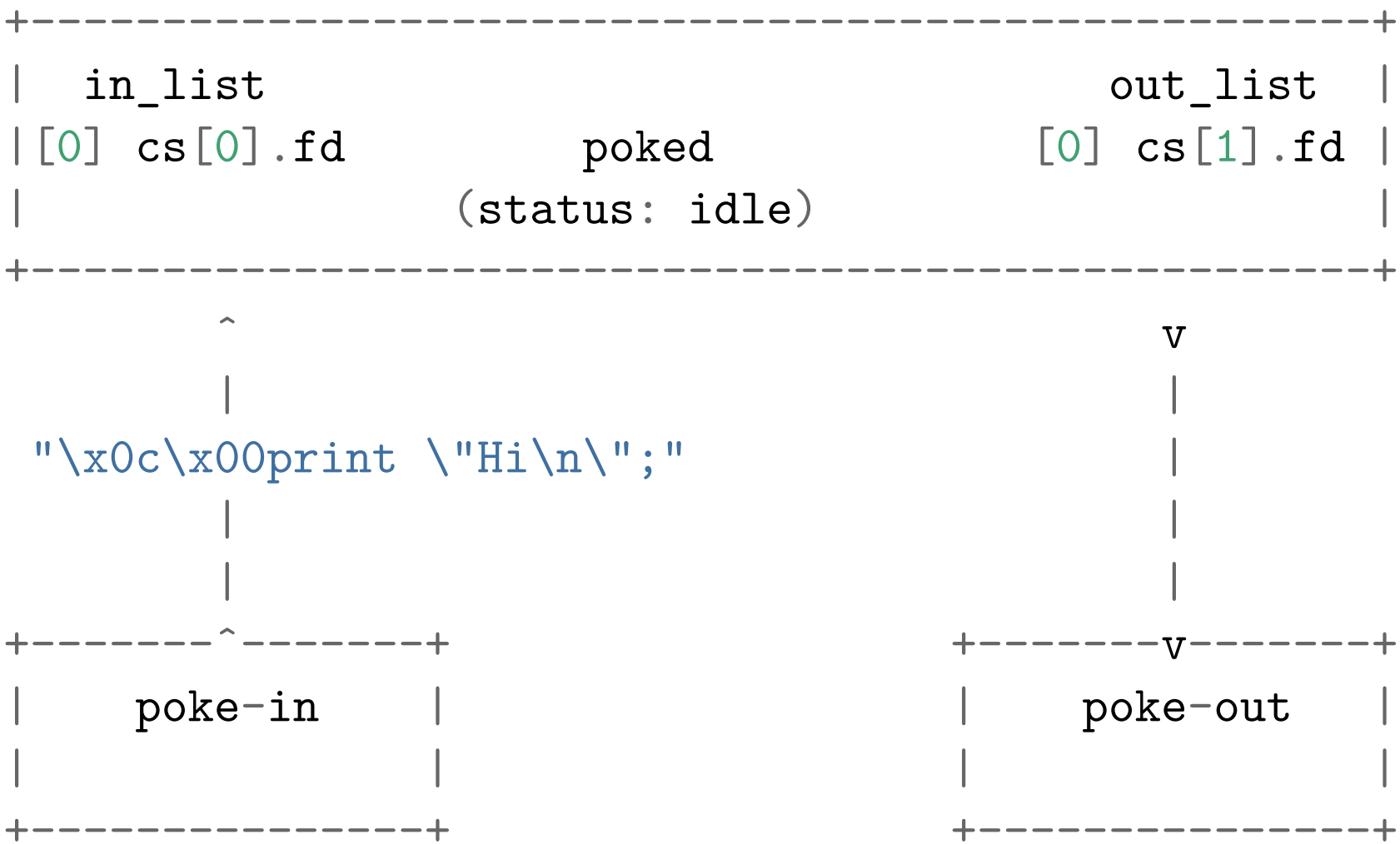
Example: poked & pokelet poke-in & pokelet poke-out



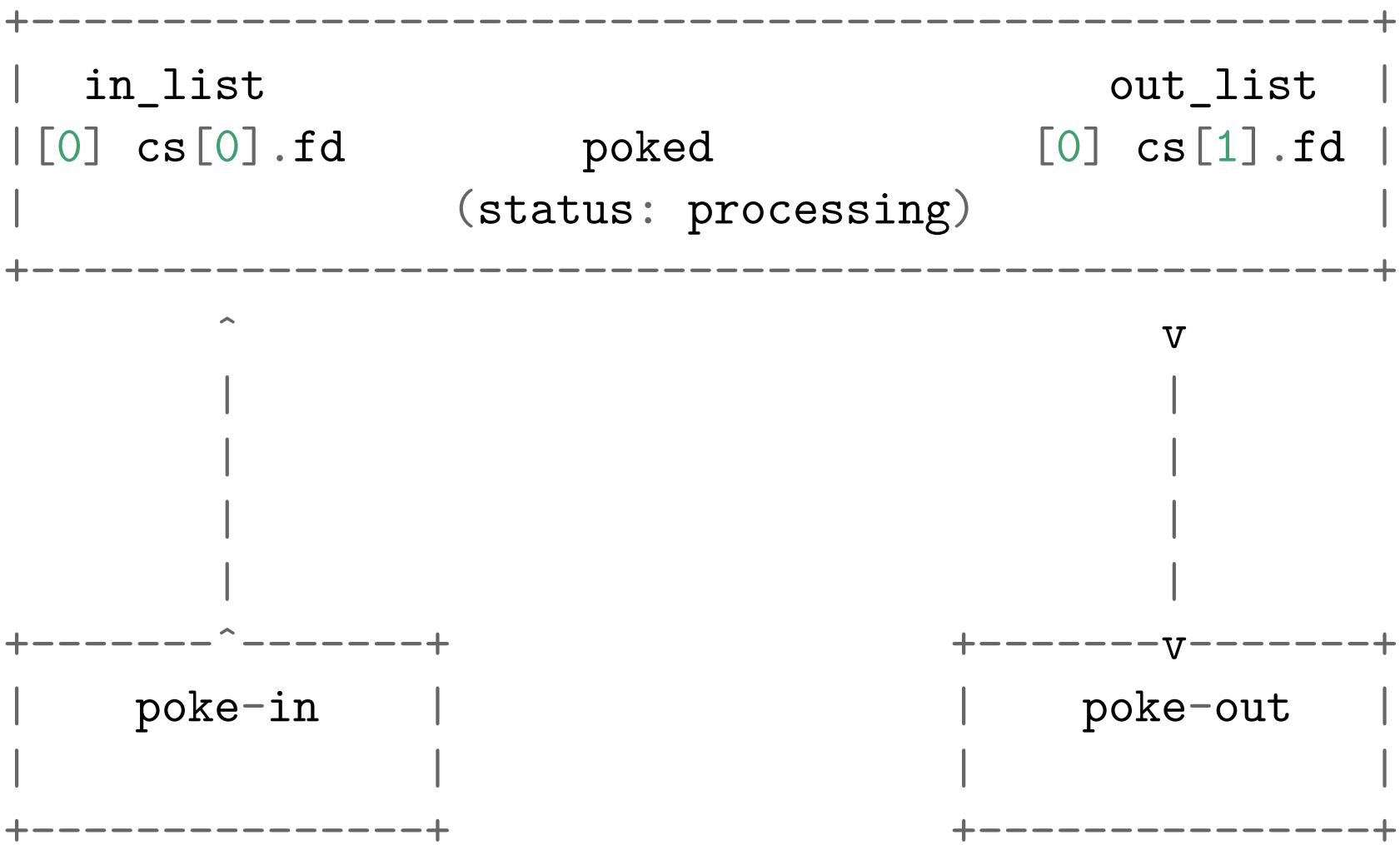
Example: poked & pokelet poke-in & pokelet poke-out



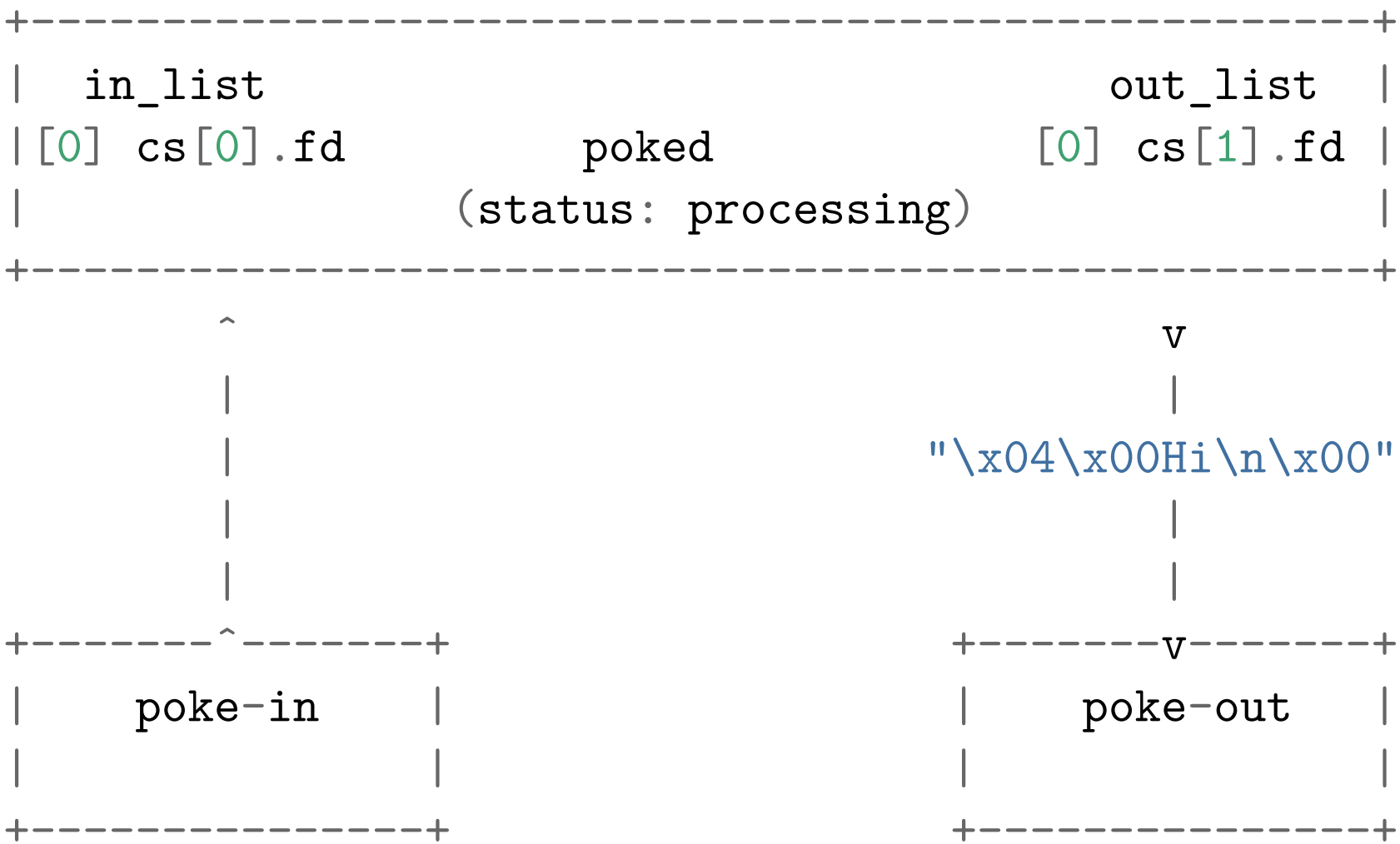
Example: poked & pokelet poke-in & pokelet poke-out



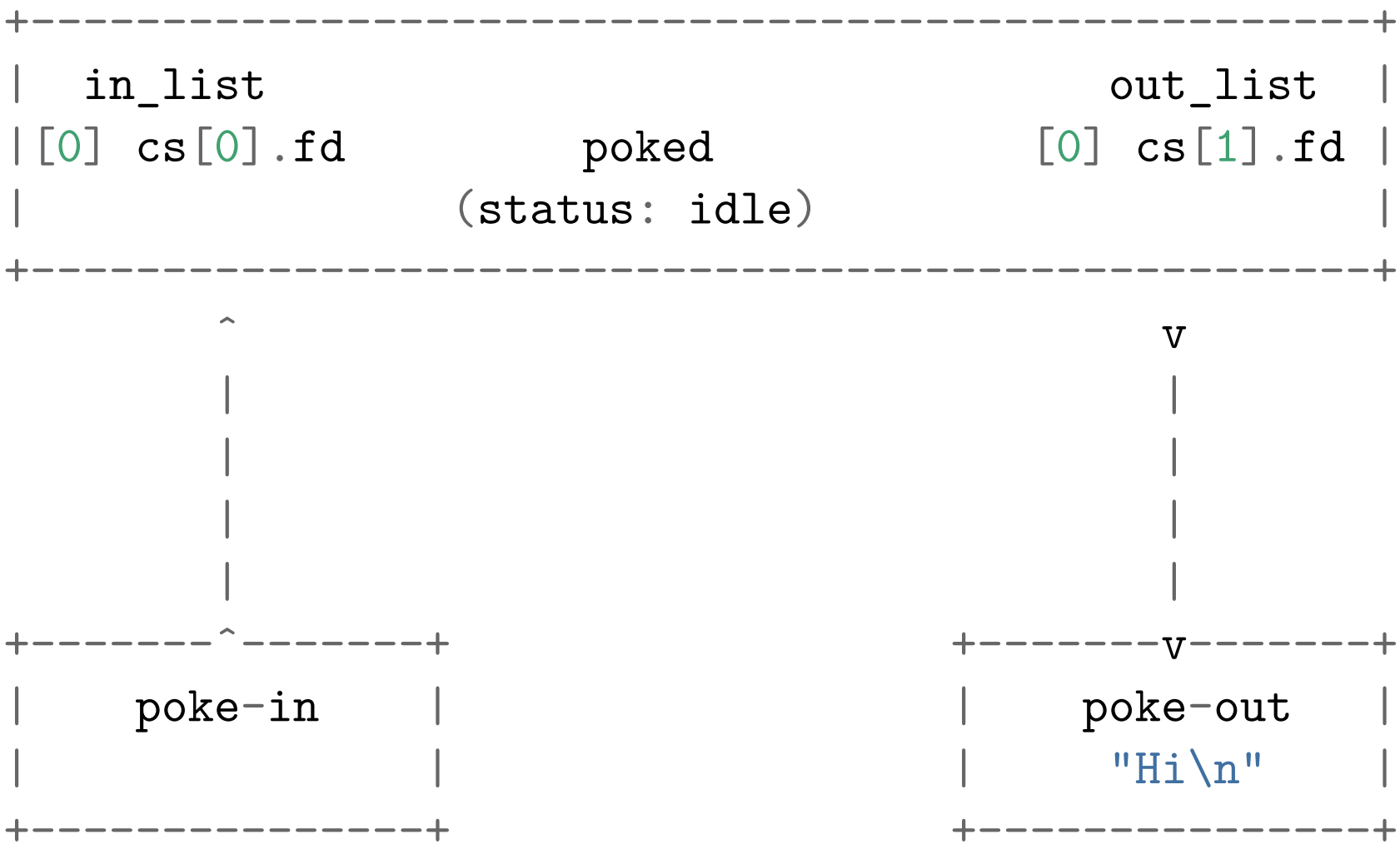
Example: poked & pokelet poke-in & pokelet poke-out



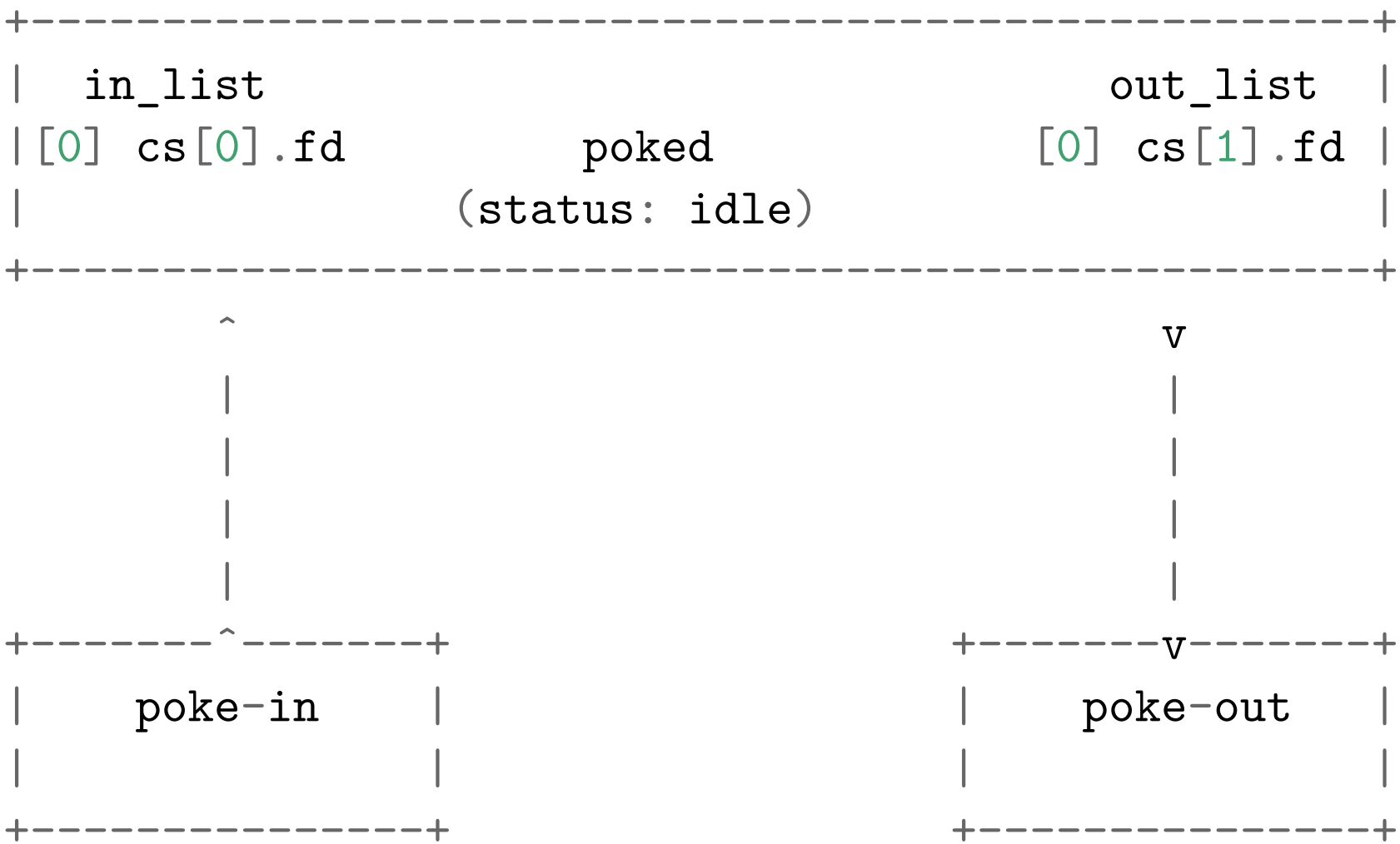
Example: poked & pokelet poke-in & pokelet poke-out



Example: poked & pokelet poke-in & pokelet poke-out



Example: poked & pokelet poke-in & pokelet poke-out



Questions?

Ask your questions about the example.

pokelet for terminal output channel (channel 1)

Part 1

```
import asyncio
import sys

async def main():
  (r, w) = await asyncio.open_unix_connection(
      'poked.ipc'
  )
  w.write(b'\x81')
  await w.drain()

  while True:
    # ...
```

pokelet for terminal output channel (channel 1)

Part 2

```
# ...
while True:
  lbytes = await r.readexactly(2)
  l = int.from_bytes(lbytes, byteorder='little')
  data = await r.readexactly(l)
  cmd, data = data[0], data[1:]
  if cmd == 1:  # New iteration
    iteration_n = int.from_bytes(
      data[:8], byteorder='little')
    print(f'\n//--- {iteration_n}')
  elif cmd == 2:  # Output
    output = data[:-1].decode('ascii')
    if output:
      print(output, end=''); sys.stdout.flush()
```

pokelet for terminal output channel (channel 1)

Part 3

```
# ...

asyncio.run(main())
```

Demo

- poked
- poke-out
- poke-repl
- poke-fuse
- poke-vu
- poke-treevu
- poke-disasm
- poke-websocket
- poke-plot
- poke-ide

Back to conclusion . . .

Automation as the Goal

General-purpose computers started as a **tool for automation**.

Automation as the Goal

General-purpose computers started as a **tool for automation**.

You give the recipe (the instructions) and computer will execute them.

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet
- Writing programs

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet
- Writing programs
- Debugging

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet
- Writing programs
- Debugging
- Reversing data formats or programs

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet
- Writing programs
- Debugging
- Reversing data formats or programs
- Exploring data (like scientific research, or program performance measurements)

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet
- Writing programs
- Debugging
- Reversing data formats or programs
- Exploring data (like scientific research, or program performance measurements)
- Computer-aided design (CAD)

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet
- Writing programs
- Debugging
- Reversing data formats or programs
- Exploring data (like scientific research, or program performance measurements)
- Computer-aided design (CAD)
- Editing photos/videos

Automation as the Goal

But nowadays, we don't use computers **only** for automation.

- Reading PDFs
- Watching videos
- Surfing internet
- Writing programs
- Debugging
- Reversing data formats or programs
- Exploring data (like scientific research, or program performance measurements)
- Computer-aided design (CAD)
- Editing photos/videos

At first sight, you might think that these tasks are not automation-friendly.

Automation as the Goal

If you see patterns, that's a sign for automation!

Automation as the Goal

If you see patterns, that's a sign for automation!

Even for tasks like editing images or videos which seems non-repetitive and "creative".

Automation as the Goal

Final words

Please,

- Don't hide your model behind GUI/TUI/Browser/etc.

Automation as the Goal

Final words

Please,

- Don't hide your model behind GUI/TUI/Browser/etc.
- Make them available as standalone models (the Smalltalk 80 definition of the word).

Automation as the Goal

Final words

Please,

- Don't hide your model behind GUI/TUI/Browser/etc.
- Make them available as standalone models (the Smalltalk 80 definition of the word).
- Make them **observable**.

Automation as the Goal

Final words

Please,

- Don't hide your model behind GUI/TUI/Browser/etc.
- Make them available as standalone models (the Smalltalk 80 definition of the word).
- Make them **observable**.
- Make them **controllable**.