

# Information Flow Control Made Simple(r)

Hemant Gouni  
gouni008@umn.edu

# **Why bother with information flow?**

(Why is it cool? Why did I do any of this?)

# The part we care about:

**confidentiality**

**integrity**



**proofs** (>1mloc)

**kernel** (~10kloc)

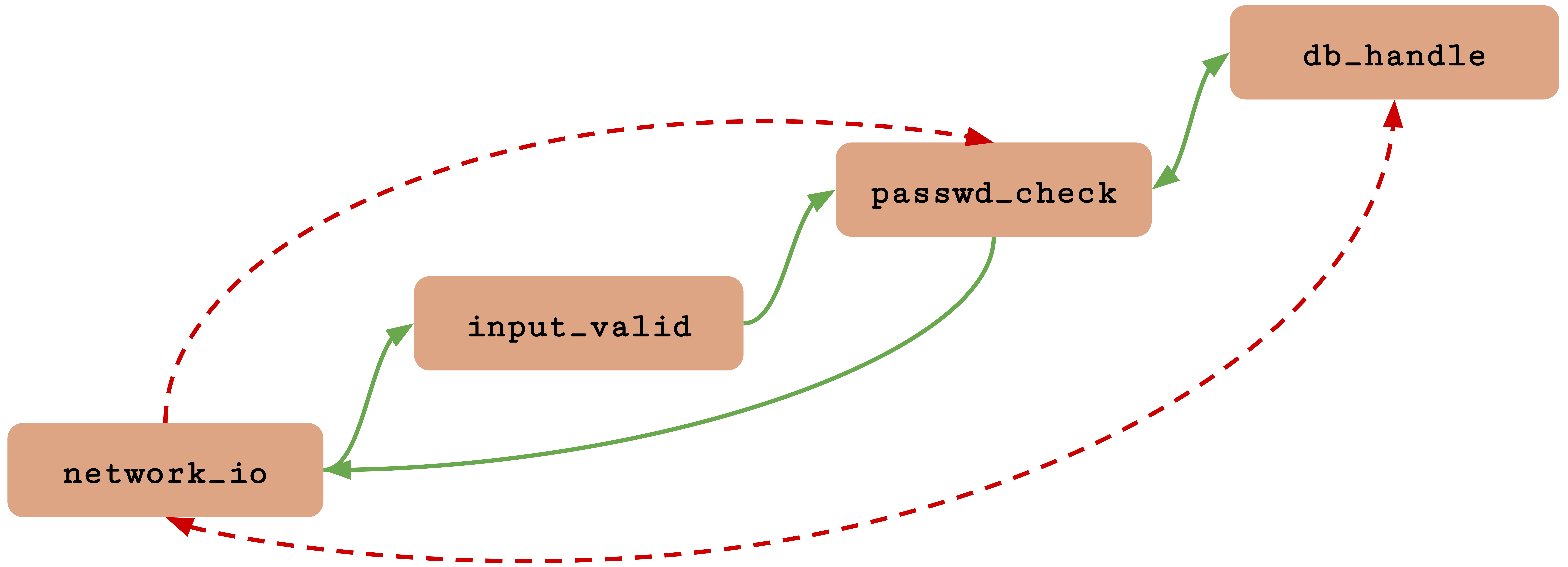
**The part we care about:**

**confidentiality**

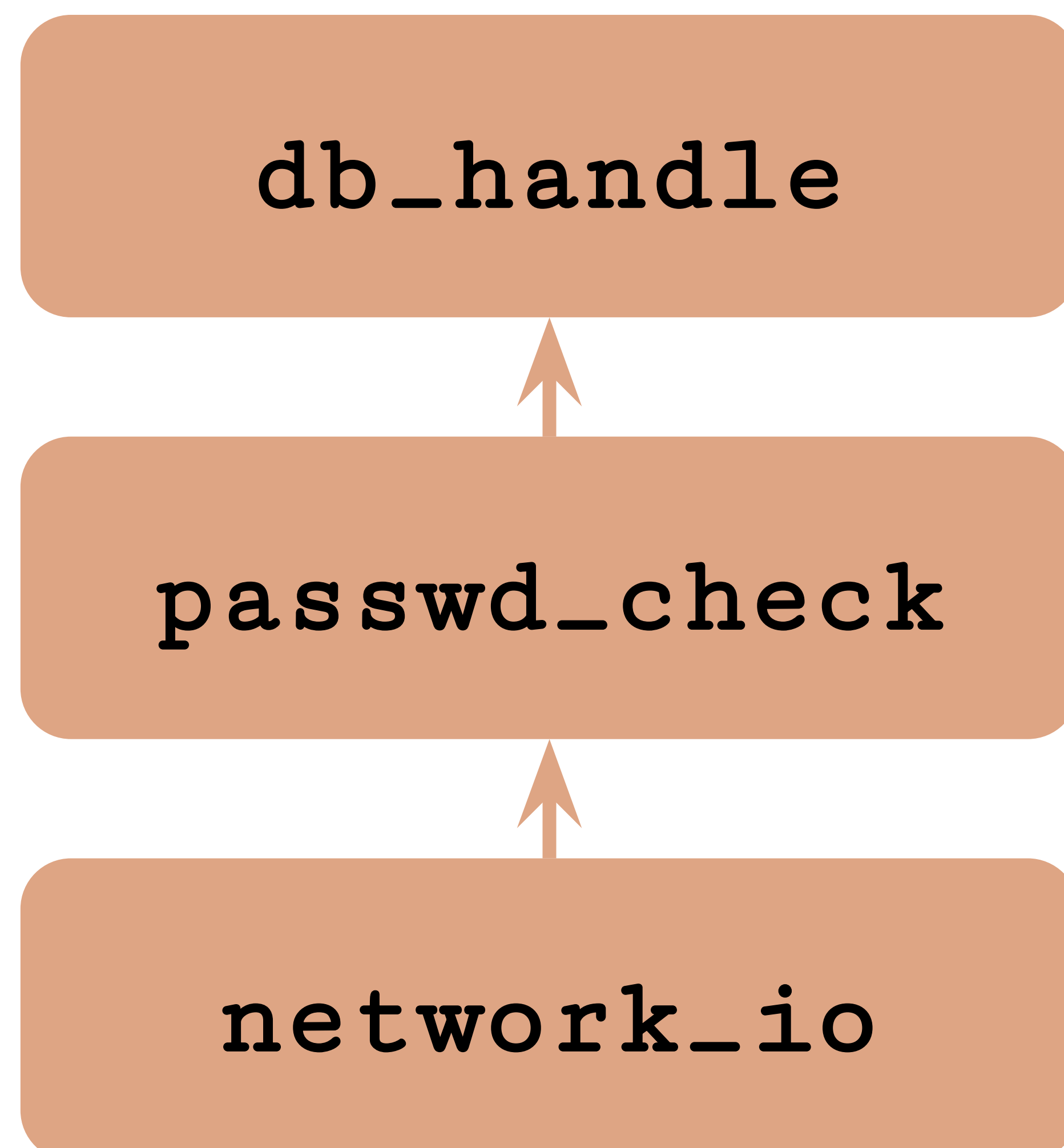
**integrity**

**Information Flow Control!**

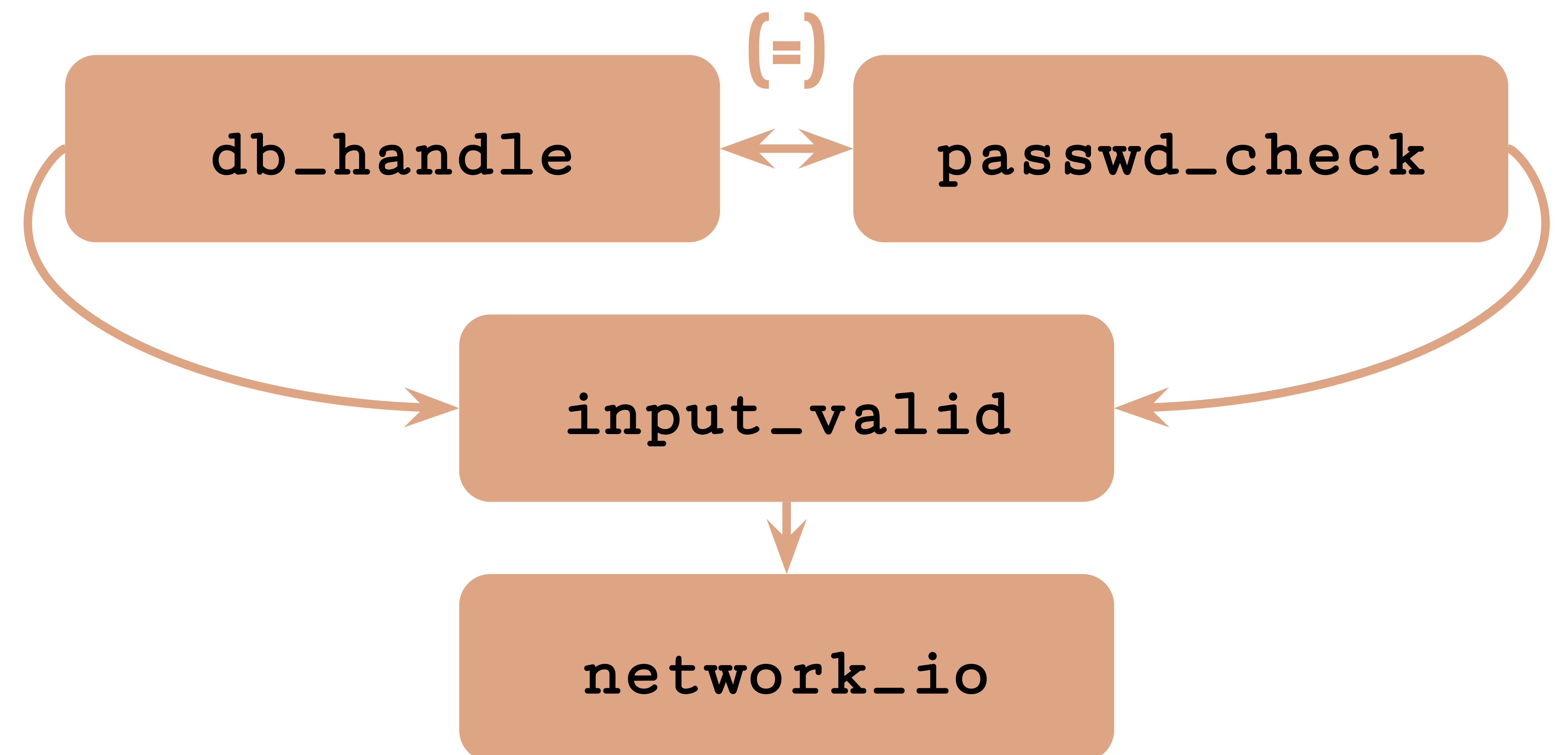
**Why doesn't IFC see more use?**

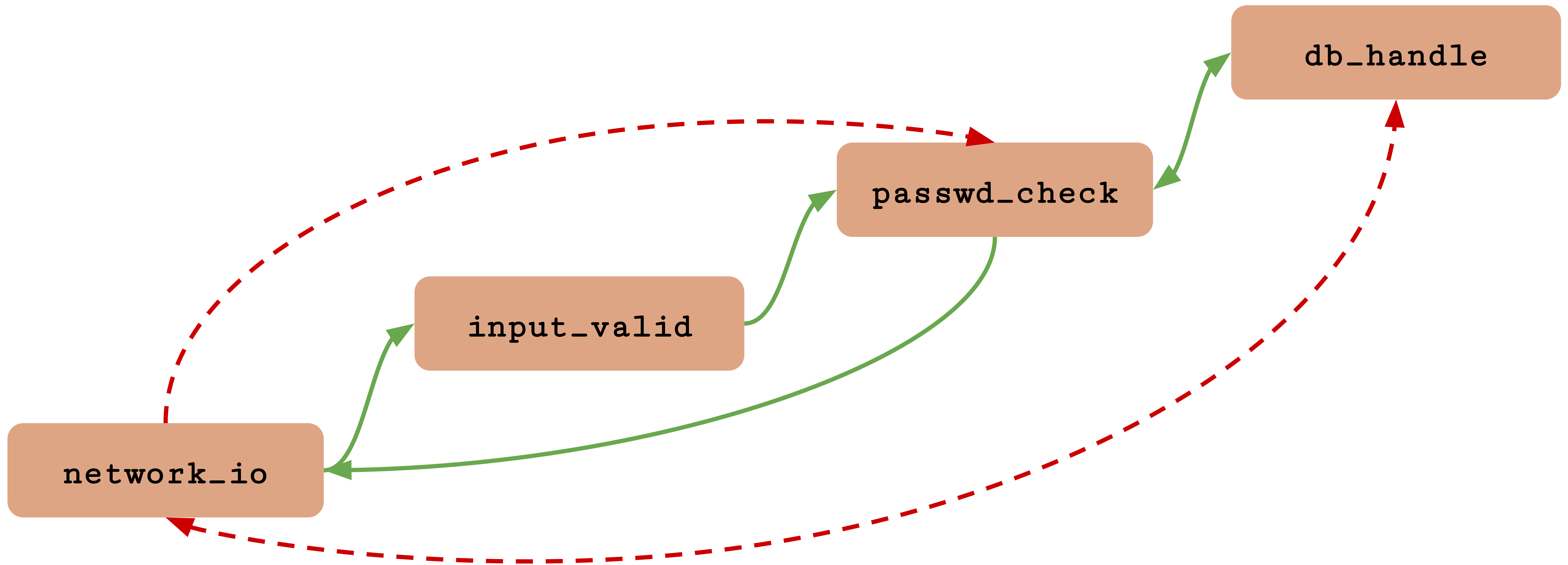


**confidentiality:**



**integrity:**

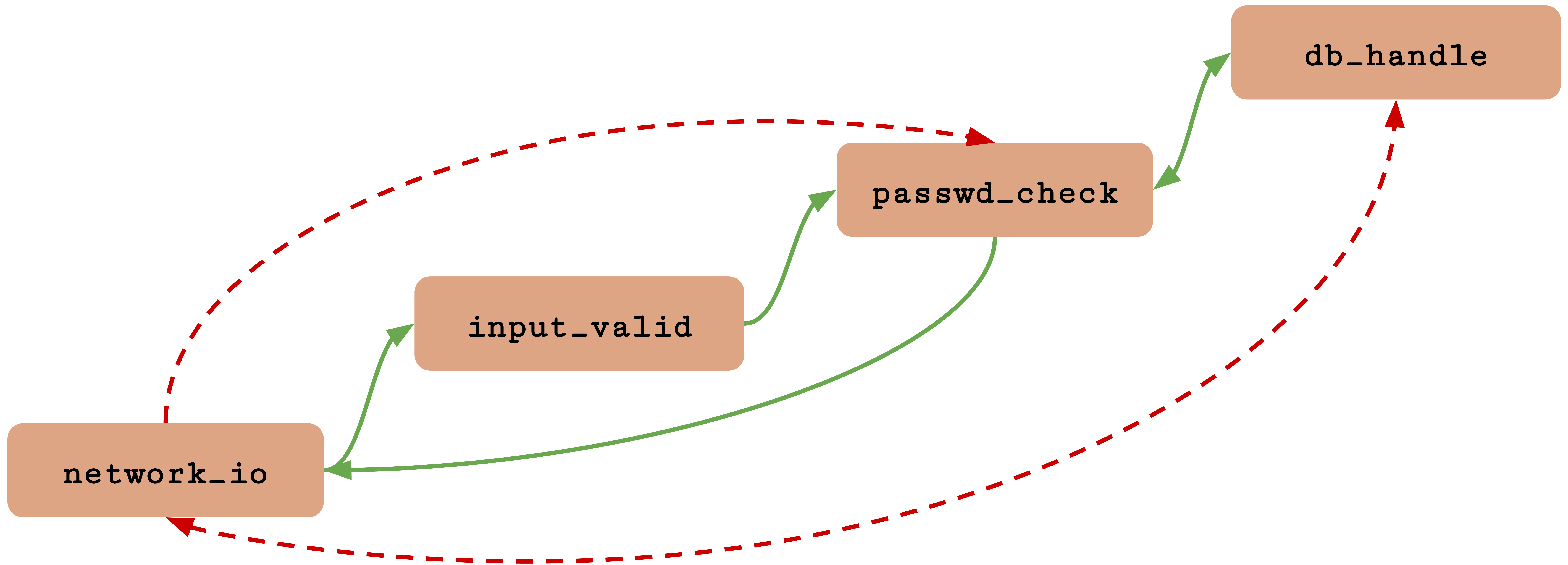




```
flow mod network_io ->! mod passwd_check;
```

```
flow mod network_io ->! mod db_handle;
```

```
flow mod db_handle ->! mod network_io;
```



declassify  
endorse

allow



# An observation on software architecture



Database  
connectors

Connection  
multiplexing

State  
management

HTTP routing

```
rocket::config::  
TlsConfig::key()
```



3rd party  
services

Caching

Room and user  
management

Federating

Reactions

Notifications

```
conduit::Database  
.globals.keypairs()
```

## A simple flow rule

```
let sec = ("password123", "1.1.1.1");  
let sock = Socket::new(...);
```

```
flow sec ->! sock;
```

## Overriding flow rules

```
{  
  flow sec.snd -> sock;  
  sock.bind(sec.snd);  
}
```

## Error!

```
sock.send(sec.snd);  
          ^^^^  
          |---- error: flow from  
                sec.snd to sock  
                with rule  
                sec ->! sock
```

A flow rule is  
declared

The rule is overridden

sec.snd flows to sock

sec.snd can't flow to  
sock here, error!

# Case study: Rocket



# This is where Rocket sets up TLS

```
flow self.config.tls.key ->! fn io!();  
flow self.config.tls.key ->! *;
```

...

```
let mut listener: Either<TcpListener, TlsListener> =  
    Left(TcpListener::bind(addr).await.map_err(ErrorKind::Bind)?);
```

```
if self.config.tls_enabled() {  
    if let Some(ref config) = self.config.tls  
        with flow self.config.tls.key -> config {  
  
        let conf = config.to_native_config().map_err(ErrorKind::Io)?  
            with flow self.config.tls.key -> conf;  
  
        flow self.config.tls.key -> listener;  
  
        listener =  
            Right(TlsListener::bind(addr, conf).await.map_err(ErrorKind::Bind)?);  
    }  
}
```

```
listener = allow listener;
```

...

**self.config.tls.key  
contains raw key data**

```
flow self.config.tls.key ->! fn io!();  
flow self.config.tls.key ->! *;
```

...

```
let mut listener: Either<TcpListener, TlsListener> =  
    Left(TcpListener::bind(addr).await.map_err(ErrorKind::Bind)?);
```

```
if self.config.tls_enabled() {  
    if let Some(ref config) = self.config.tls  
        with flow self.config.tls.key -> config {  
  
        let conf = config.to_native_config().map_err(ErrorKind::Io)?  
            with flow self.config.tls.key -> conf;  
  
        flow self.config.tls.key -> listener;  
  
        listener =  
            Right(TlsListener::bind(addr, conf).await.map_err(ErrorKind::Bind)?);  
    }  
}
```

```
listener = allow listener;
```

...

```
flow self.config.tls.key ->! fn io!();
flow self.config.tls.key ->! *;
```

...

```
let mut listener: Either<TcpListener, TlsListener> =
    Left(TcpListener::bind(addr).await.map_err(ErrorKind::Bind)?);
```

```
if self.config.tls_enabled() {
    if let Some(ref config) = self.config.tls
        with flow self.config.tls.key -> config {
```

```
        let conf = config.to_native_config().map_err(ErrorKind::Io)?
            with flow self.config.tls.key -> conf;
```

```
        flow self.config.tls.key -> listener;
```

```
        listener =
            Right(TlsListener::bind(addr, conf).await.map_err(ErrorKind::Bind)?);
```

```
    }
```

```
}
```

```
listener = allow listener;
```

...

**Bind a variable which initially holds a TCP socket**

**Check that TLS is enabled**

**Extract TLS config**

**Create a TLS socket**

Prevents flows to  
side-effecting functions

\* prevents flows to  
all variables

```
flow self.config.tls.key ->! fn io!();  
flow self.config.tls.key ->! *;
```

...

```
let m ... TcpListener, TlsListener> =  
L ... (addr).await.map_err(ErrorKind::Bind)?);
```

```
if self.config.tls_enabled() {  
    if let Some(ref config) = self.config.tls  
        with flow self.config.tls.key -> config {
```

```
        let conf = config.to_native_config().map_err(ErrorKind::Io)?  
            with flow self.config.tls.key -> conf;
```

```
        flow self.config.tls.key -> listener:
```

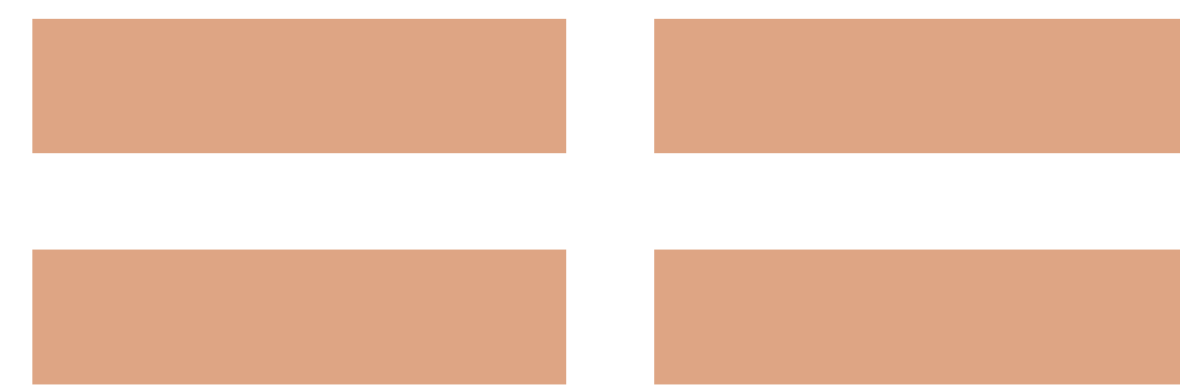
```
        listener  
            R ... await.map_err(ErrorKind::Bind()?);  
    }  
}
```

```
listener = allow listener;
```

...

with flow inserts the flow policy  
between binding and initialization

```
let conf;  
flow self.config.tls.key -> conf;  
conf = ...;
```



```
let conf = ...  
  with flow self.config.tls.key -> conf;
```



```
flow self.config.tls.key ->! fn io!();
flow self.config.tls.key ->! *;
```

...

```
let mut listener: Either<TcpListener, Listener> =
  Left(TcpListener::bind(addr, p_err(ErrorKind::Bind)?));
```

**These are all overrides**

```
if self.config.tls_enabled {
  if let Some(ref config) = self.config.tls
    with flow self.config.tls.key -> config {
```

```
    let conf = config.to_native_config().map_err(ErrorKind::Io)?
      with flow self.config.tls.key -> conf;
```

```
    flow self.config.tls.key -> listener;
```

```
    listener =
      Right(TlsListener::bind(addr, conf).await.p_err(ErrorKind::Bind)?);
  }
```

**This call is permitted**

```
listener = allow listener;
```

**We allow any further flows here**

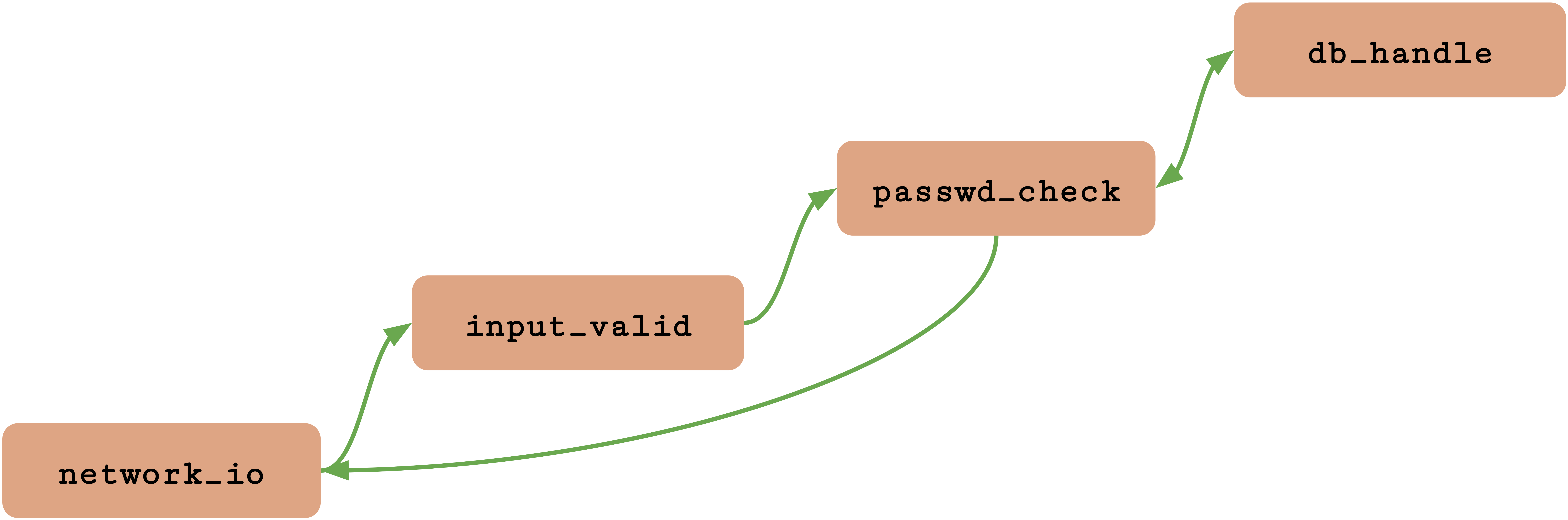
...

# **Weaknesses**

**Normalizing the use of 'coercions'**

**No user studies!**

**Lattices might be neat!**



# **Weaknesses**

**Normalizing the use of 'coercions'**

**No user studies!**

**Lattices might be neat!**

# **Weaknesses**

**Normalizing the use of 'coercions'**

**No user studies!**

**Lattices might be neat!**

# **Green's Cognitive Dimensions of Notations**

**Progressive evaluation**

**Consistency**

**Closeness of mapping**

**Hard mental operations**

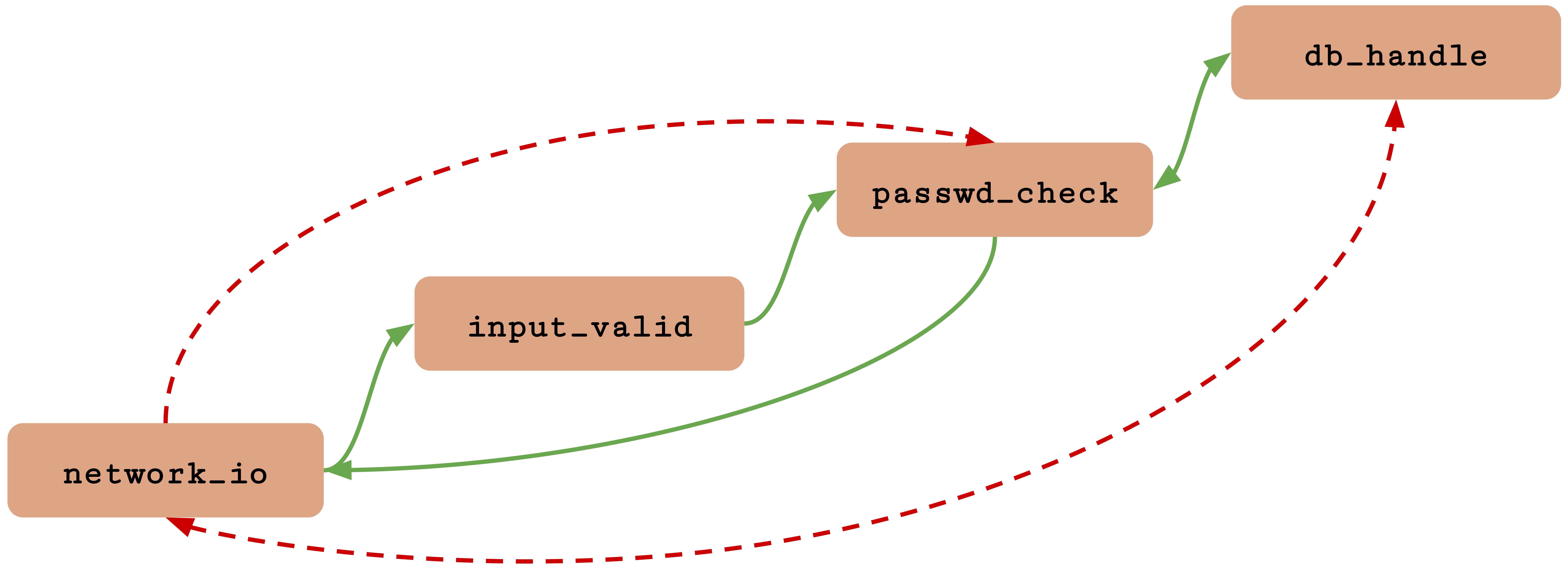
## **Progressive evaluation**

**"How easy is it to evaluate and obtain feedback on an incomplete solution?"**

# Consistency

**"After part of the notation has been learned, how much can be guessed?"**





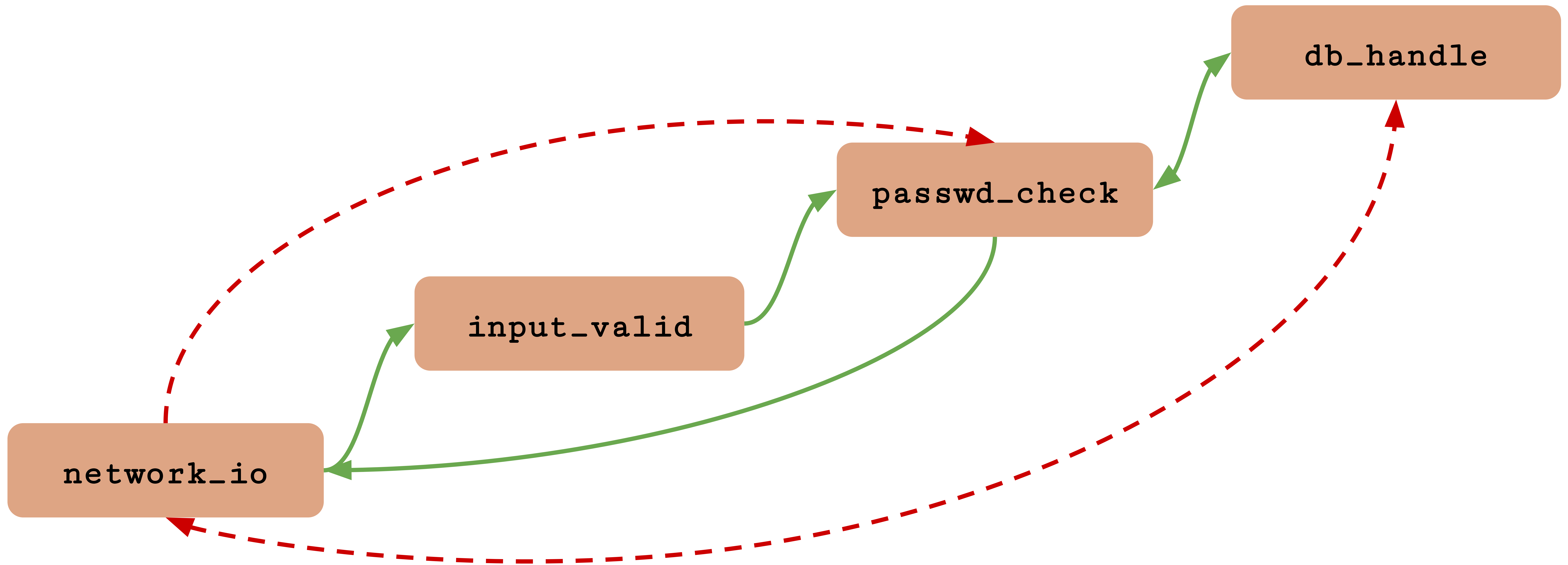
```
flow mod network_io ->! mod passwd_check;
```

```
flow mod network_io ->! mod db_handle;
```

```
flow mod db_handle ->! mod network_io;
```

## **Closeness of mapping**

**"How closely does the notation correspond to the problem world?"**



```
flow mod network_io ->! mod passwd_check;
```

```
flow mod network_io ->! mod db_handle;
```

```
flow mod db_handle ->! mod network_io;
```

## **Hard mental operations**

**"How much hard mental processing lies at the notational level, rather than the semantic one?"**

# What's next?

**Soundness**

**User studies**

**Implementation**

# **What's next?**

**Soundness**

**User studies**

**Implementation**

# What's next?

Soundness

User studies

Implementation

# Thanks

**IFC is neat, and it's worth finding ways to make it more accessible and pleasant to use**

**User-facing lattices do more harm than good**