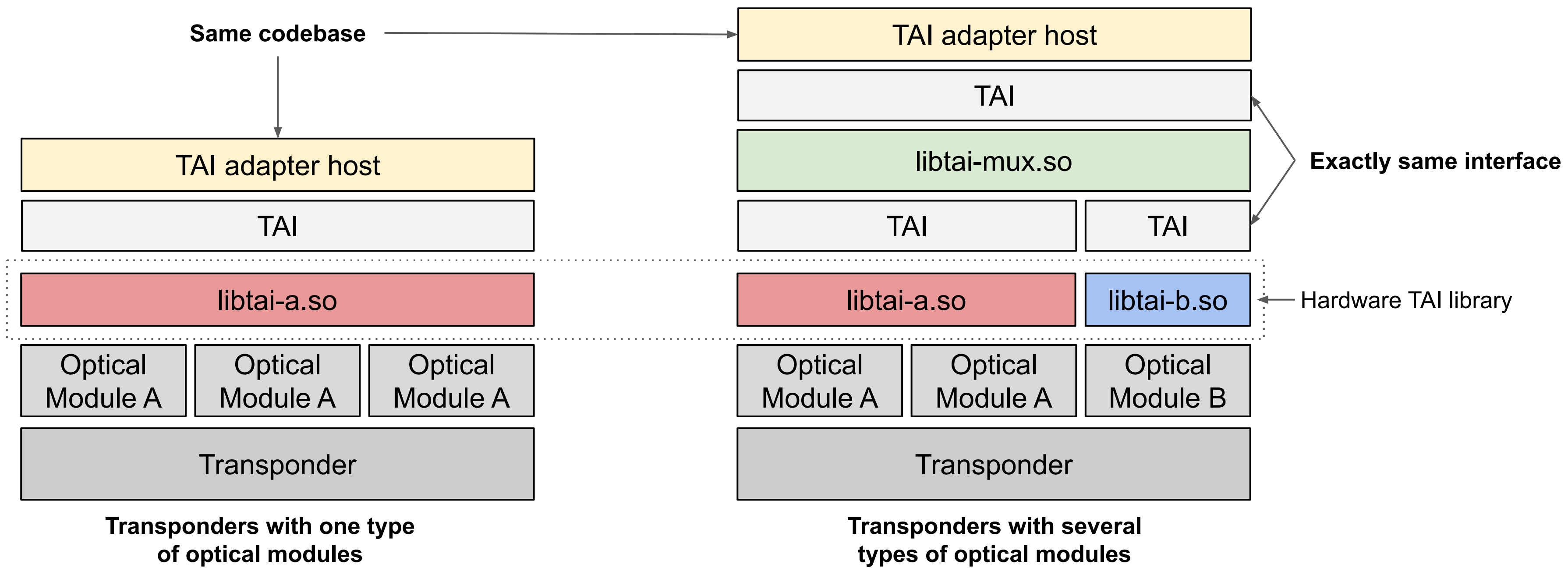
TAI multiplexer libtai-mux.so

Wataru Ishida

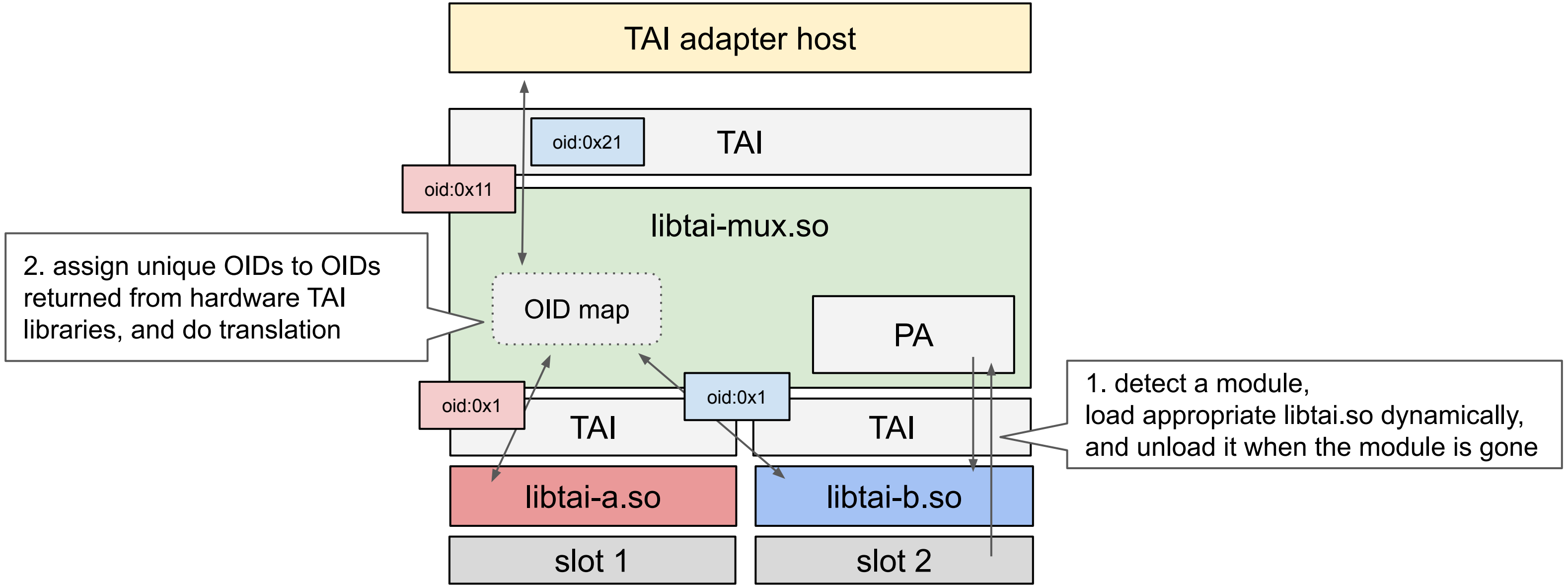
TAI multiplexer - libtai-mux.so

- TAI library to multiplex multiple TAI libraries
- It can be used for hardware which supports multiple types of optical module (e.g. Edgecore Cassini)
- Available here:
  - https://github.com/Telecominfraproject/oopt-tai-implementations/tree/master/tai_mux



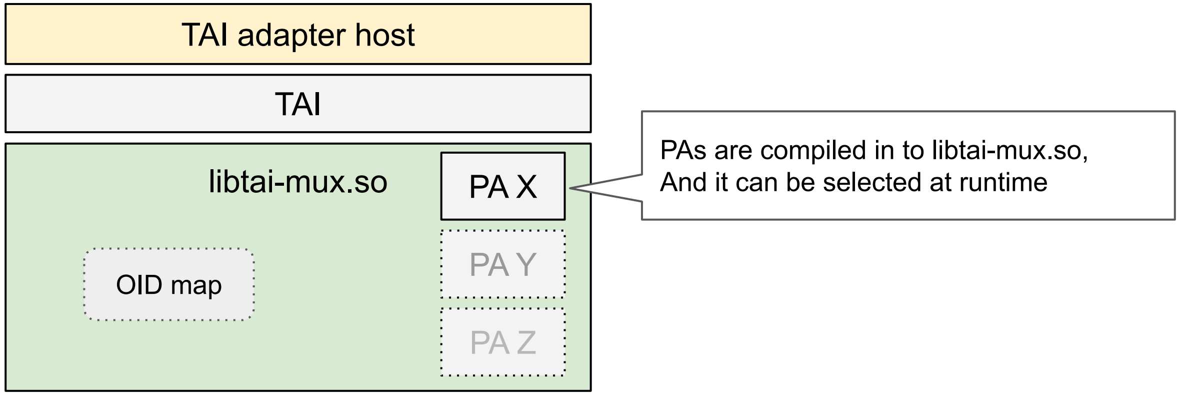
What libtai-mux.so does

1. Dynamic hardware TAI library loading/unloading
    - Platform Adapter (PA) detects modules and decides which hardware TAI library to use
2. Object ID (OID) mapping
    - Hardware TAI libraries could use the same object ID for different objects
    - libtai-mux.so manages Object ID map and ensures unique IDs are returned to TAI adapter host


TAI adapter host
oid:0x21
TAI
oid:0x11
libtai-mux.so
2. assign unique OIDs to OIDs returned from hardware TAI libraries, and do translation
OID map
PA
oid:0x1
oid:0x1
TAI
TAI
1. detect a module, load appropriate libtai.so dynamically, and unload it when the module is gone
libtai-a.so
libtai-b.so
slot 1
slot 2

Platform Adapter (PA)

- PA detects module insertion/removal, decides which hardware TAI library to load/unload based on their policy/configuration
- libtai-mux.so has modular design to support various types of PA
  - The methods to detect modules varies between OS and hardware
  - software/system vendors may develop their own PA
  - Users can select a PA at runtime by passing an env variable `TAI_MUX_PLATFORM_ADAPTER`
- Currently only static platform adapter is open sourced


TAI adapter host
TAI
libtai-mux.so
PA X
PA Y
PA Z
OID map
PAs are compiled in to libtai-mux.so, And it can be selected at runtime

static PA

- static PA is a PA which uses static configuration
- It doesn’t do module detection and blindly call module_presence() callback based on the configuration
- Configuration format is json. The key is location of the module and the value is the library to use for it
- By using the configuration below, libtai-a.so is used for modules whose location is 1,2,3,4, and libtai-b.so is used for modules whose location is 5,6,7,8

```
{
  “1”: “libtai-a.so”,
  “2”: “libtai-a.so”,
  “3”: “libtai-a.so”,
  “4”: “libtai-a.so”,
  “5”: “libtai-b.so”,
  “6”: “libtai-b.so”,
  “7”: “libtai-b.so”,
  “8”: “libtai-b.so”
}
```

/etc/tai/mux/static.json

Future roadmap

- Add Open Network Linux (ONL) platform adapter
  - We need to add dynamic module support to ONLP (Platform Abstraction Layer for ONL) first
  - ONL PA automatically detects which type (e.g. ACO card, DCO card etc..) of module is inserted and loads appropriate hardware TAI library.
    - With this, we can hot swap different types of module without any config modification!
- Add mechanizm to handle register access in libtai-mux.so/PA layer
  - Currently hardware TAI library needs to know how to do actual register access for particular platform (e.g Which bus to use i2c, MDIO or PCIe?)
  - It would be nice if libtai-mux.so can give an abstracted register access callbacks to hardware TAI library
    - This can increase the portability of hardware TAI library
    - Add entry to `tai_service_method_table_t` ?
    - How to handle hardware pins (e.g. tx_dis, mod_abs, mod_lopwr)?
    - How to handle hardware which uses several buses?