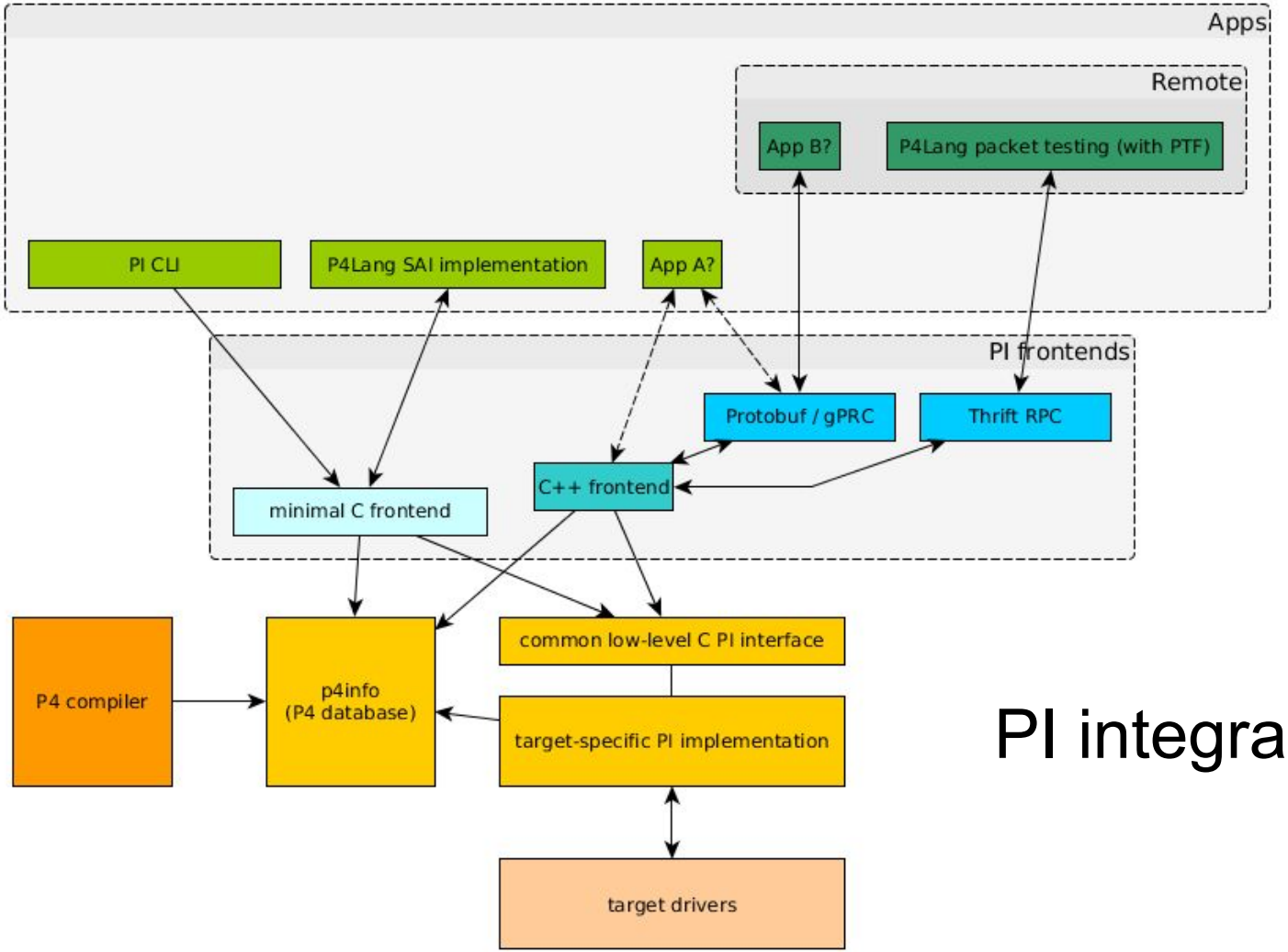
PI API

A program-independent P4 API

PI in 3 points

- No driver recompilation needed when the P4 program is updated and, under certain conditions, no app recompilation needed
- A low-level efficient C interface, which can be easily extended for user-friendliness, sanity-checking…
- A common API implemented by all P4-compatible targets for portability of controllers and apps

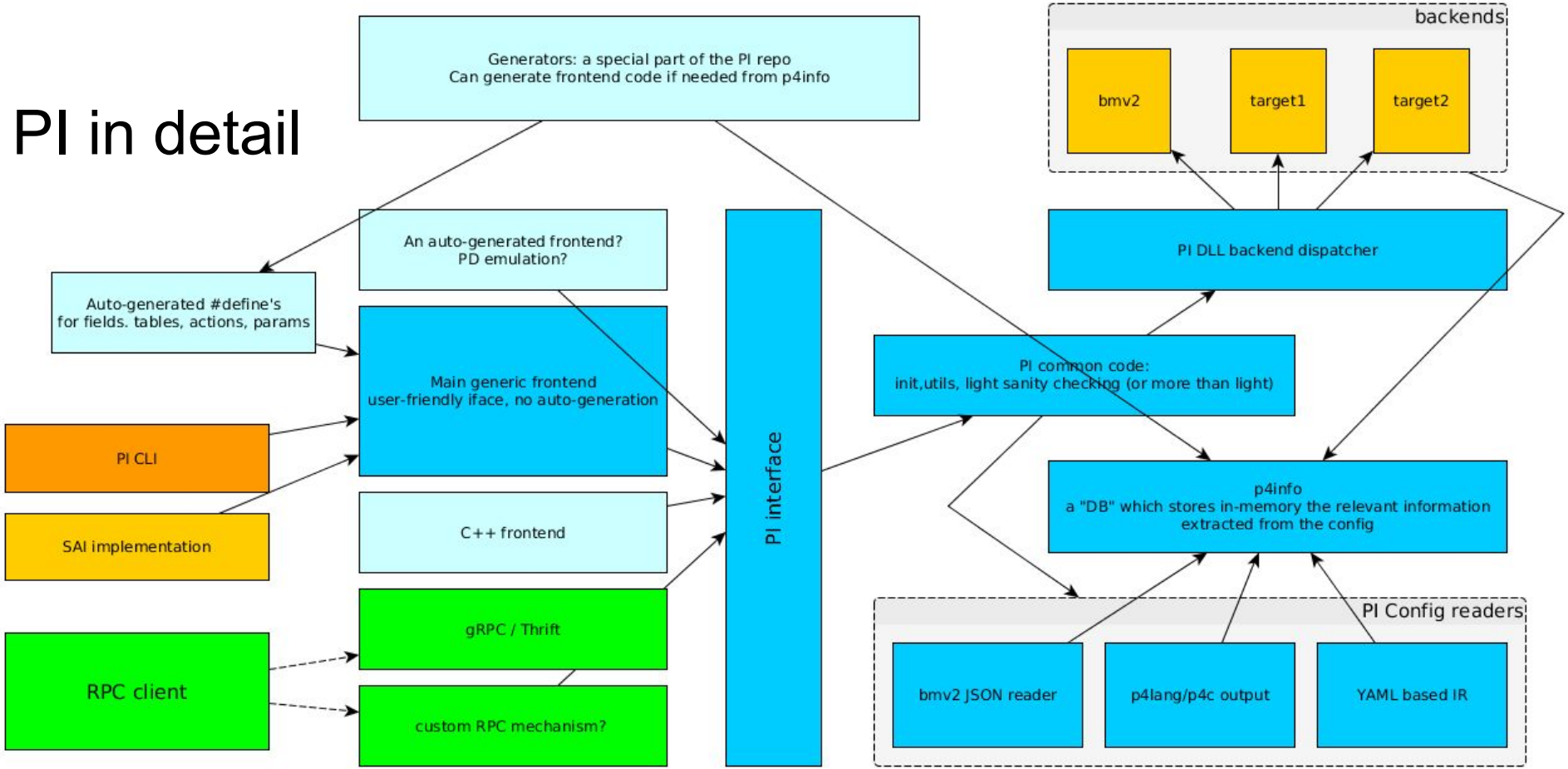


PI integration

Proof of concept, what we have

- PI able to process P4 compiler output to extract relevant information (e.g. match fields of each table) into the `p4info` P4 database
- PI implementation for P4Lang software switch (bmv2) supports adding and removing entries
- PI CLI supports `table_add` and `table_delete` to add and remove table entries
- A simple, efficient C++ PI frontend for C++ apps which need to be linked with `libpi`

PI in detail


Generators: a special part of the PI repo
Can generate frontend code if needed from p4info
backends
bmv2
target1
target2
An auto-generated frontend?
PD emulation?
PI DLL backend dispatcher
Auto-generated #define's
for fields. tables, actions, params
Main generic frontend
user-friendly iface, no auto-generation
PI common code:
init,utils, light sanity checking (or more than light)
PI CLI
SAI implementation
C++ frontend
PI interface
p4info
a "DB" which stores in-memory the relevant information
extracted from the config
gRPC / Thrift
PI Config readers
RPC client
custom RPC mechanism?
bmv2 JSON reader
p4lang/p4c output
YAML based IR

Some PI methods

```
pi_status_t pi_table_entry_add(
    pi_dev_id_t dev_id, pi_p4_id_t table_id, const pi_match_key_t *match_key,
    const pi_table_entry_t *table_entry, int overwrite, pi_entry_handle_t *entry_handle);

pi_status_t pi_table_entry_delete(
    pi_dev_id_t dev_id, pi_p4_id_t table_id, pi_entry_handle_t entry_handle);

pi_status_t pi_table_default_action_set(
    pi_dev_id_t dev_id, pi_p4_id_t table_id, const pi_table_entry_t *table_entry);

pi_status_t pi_table_entries_fetch(
    pi_dev_id_t dev_id, pi_p4_id_t table_id, pi_table_fetch_res_t **res);

pi_status_t pi_table_default_action_get(
    pi_dev_id_t dev_id, pi_p4_id_t table_id, pi_table_entry_t *table_entry);
```

The PI frontends

- The match key (`pi_match_key_t`) and action data (`pi_action_data_t`) use an efficient representation which is standardized and known to the backend implementation, but not very application-friendly
- Frontends are more convenient to use, and represent a small amount of code
- Different frontends for different use cases (instead of one size fits all)
- Can be auto-generated or not

C++ frontend: a simple example

```
#include "PI/frontends/cpp/tables.h"
#include "pi_fe_defines_p4.h"  // auto-generated #define's for P4 objects

pi_p4info_t *p4info = …
unique_ptr<pi::MatchTable> ipv4_table(new pi::MatchTable(p4info, PI_P4_TABLE_IPV4_LPM));

int add_route(uint32_t prefix, int pLen, uint32_t nhop, uint16_t port,
              pi_entry_handle_t *handle) {
  pi::error_code_t rc = 0;
  pi::MatchKey match_key(p4info, PI_P4_TABLE_IPV4_LPM);
  rc |= match_key.set_lpm(PI_P4_FIELD_IPV4_DSTADDR, prefix, pLen);
  pi::ActionData action_data(p4info, PI_P4_ACTION_SET_NHOP);
  rc |= action_data.set_arg(PI_P4_ACTIONP_SET_NHOP_NHOP_IPV4, nhop);
  rc |= action_data.set_arg(PI_P4_ACTIONP_SET_NHOP_PORT, port);
  rc |= ipv4_table->entry_add(match_key, PI_P4_ACTION_SET_NHOP, action_data, true, handle);
  Return rc;
}

pi_entry_handle_t handle;
// 10.0.0.0/8 => set_nhop(nhop=10.0.0.11, port=9);
assert(!add_route(0x0a000000, 8, 0x0a00000b, 9, &handle);
assert(!ipv4_table->entry_delete(handle));
```

The auto-generated `#define`'s

pi_fe_defines_p4.h ->

```
// FIELDS
#define PI_P4_FIELD_STANDARD_METADATA_INGRESS_PORT 0x4000000
#define PI_P4_FIELD_STANDARD_METADATA_PACKET_LENGTH 0x4000001  // …
#define PI_P4_FIELD_ETHERNET_DSTADDR 0x4000008  // …
#define PI_P4_FIELD_IPV4_VERSION 0x400000b  //…

// ACTIONS AND ACTION PARAMETERS
#define PI_P4_ACTION__DROP 0x1000000
#define PI_P4_ACTION_SET_NHOP 0x1000001
#define PI_P4_ACTIONP_SET_NHOP_NHOP_IPV4 0x3000100
#define PI_P4_ACTIONP_SET_NHOP_PORT 0x3000101  // …

// TABLES
#define PI_P4_TABLE_IPV4_LPM 0x2000000
#define PI_P4_TABLE_FORWARD 0x2000001
#define PI_P4_TABLE_SEND_FRAME 0x2000002  // …
```

The auto-generated `#define`'s

- Auto-generated from the information passed by the P4 compiler
- No executable code, just preprocessor macros (or alternatively enums) for convenience
- Alternatively, ids can be obtained at runtime from the name string, and cached by the application for performance (the auto-generated header file is not needed any more!)
- When adding or removing tables to a P4 program, or when modifying the control flow, there is no need to recompile or even stop the application, the old ids remain valid.
- When modifying a table or an action signature, the application needs to be updated and recompiled, but not the PI + drivers!