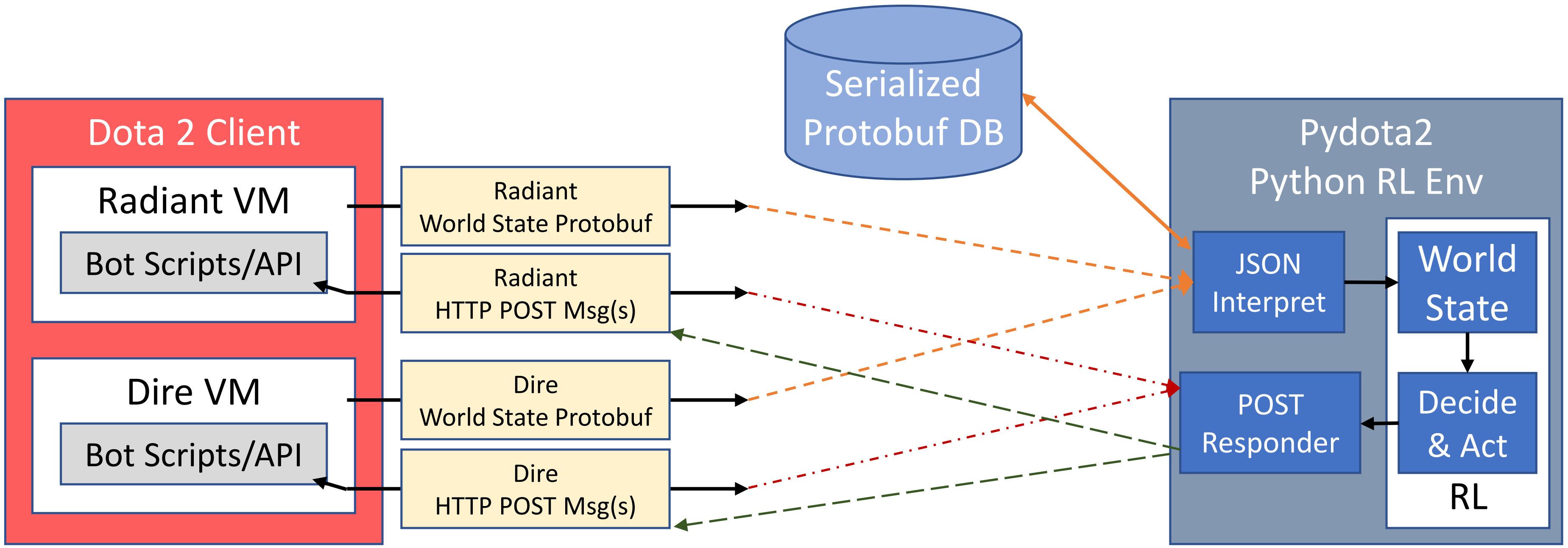
pydota2

Dota 2 Reinforcement Learning

Github: https://github.com/pydota2/pydota2

High-Level Design of pydota2 RL Environment



Dota 2 Client

Radiant VM

Bot Scripts/API

Dire VM

Bot Scripts/API

Radiant
World State Protobuf

Radiant
HTTP POST Msg(s)

Dire
World State Protobuf

Dire
HTTP POST Msg(s)

Serialized
Protobuf DB

Pydota2
Python RL Env

JSON
Interpret

World
State

POST
Responder

Decide
& Act

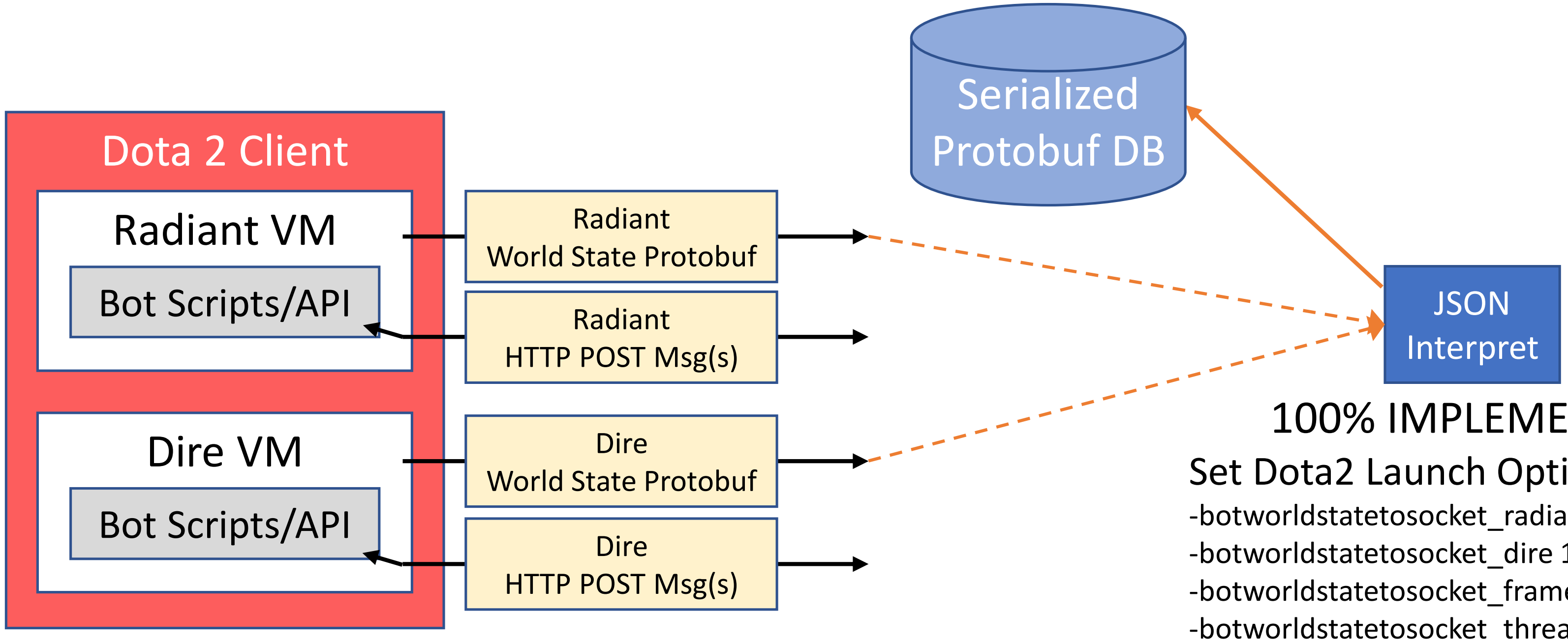
RL

Legend:

-·-·-▸ Polling post messages (using empty CreateHTTPRequest API)

◂- - - Replies to CreateHTTPRequest messages (actions to execute through Bot API)

- - -▸ ──▸ CMsgBotWorldState serialized protobuf frame dumps *pydota2/bin/proto_ingest.py*



100% IMPLEMENTED!

Set Dota2 Launch Options:

```
-botworldstatetosocket_radiant 12120
-botworldstatetosocket_dire 12121
-botworldstatetosocket_frames 10
-botworldstatetosocket_threaded
```

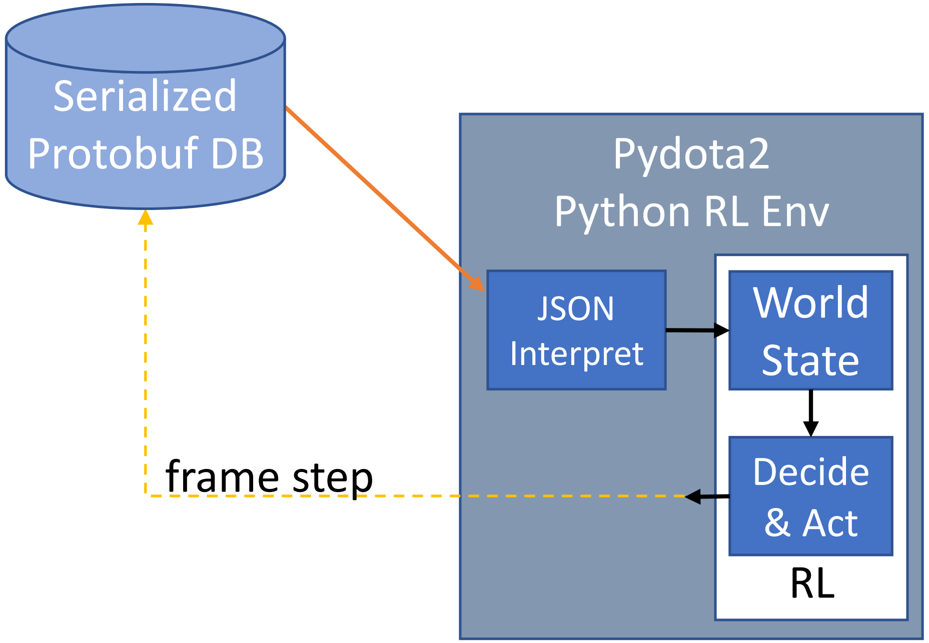
This is designed to just force the bot-match (or human(s) vs bots match) to dump the protobuf frames to the Database (which is really just a filesystem)

Files are stored in *pydota2/replays/YYYY_MM_DD_HHmm_<team>/*.bin*

Ultimately it is my hope that Valve will enable protobuf frame dumps for localized Dota2 replay files (and possibly even host a large protobuf replay DB of their own with thousands of games) *pydota2/bin/replay_actions.py*

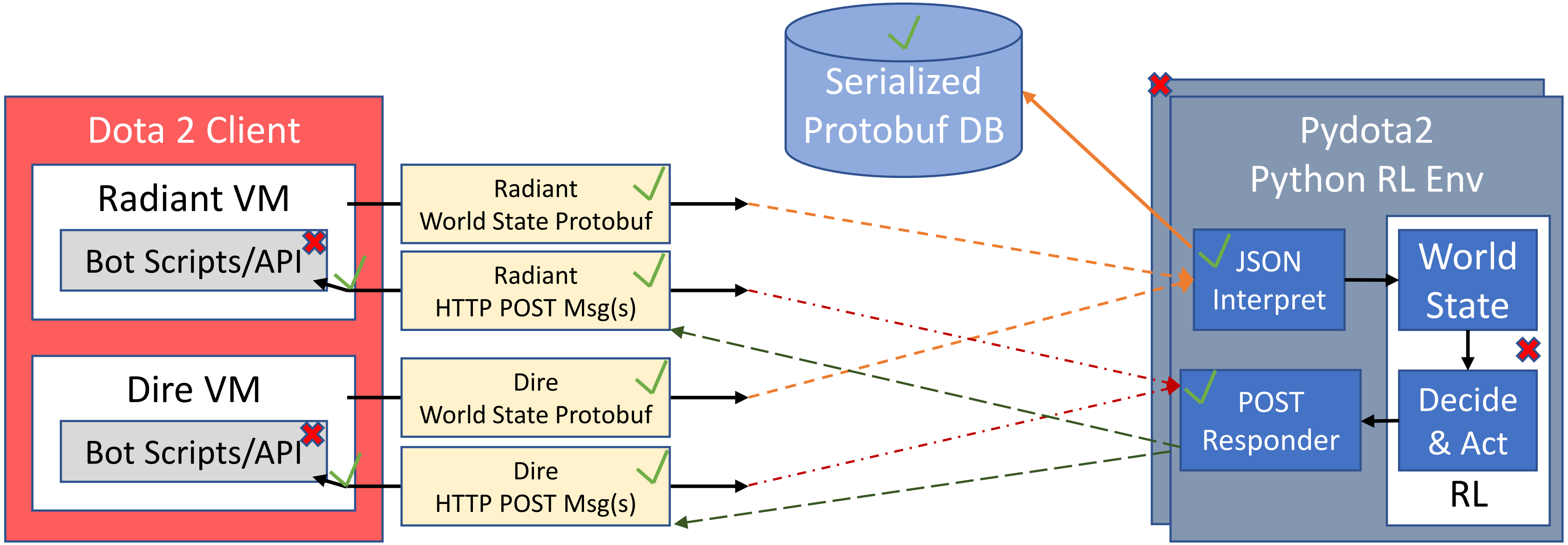
25% IMPLEMENTED

- Methods to ingest a replay file in steps is COMPLETED
- Method to map JSON data to World State is NOT YET STARTED
- Method to determine what “action” bot took is NOT YET STARTED
- Method to determine the “value”/”reward” of the action is NOT YET STARTED
- Replay multi-processing / threading could use improvement



This is designed to train our RL system as in a “supervised learning” approach.
We observe actions taken by bots in a game and mapping them against our World State. Then we short-circuit the Decide & Act state by pretending we are “learning” and thus take an arbitrary action (*aka* the action the bot/player took in game) and evaluate the result of that action as determined through evaluation of next step’s World State state to set a weight to the transition of state->action->reward->new_state and thus complete coverage of our system.

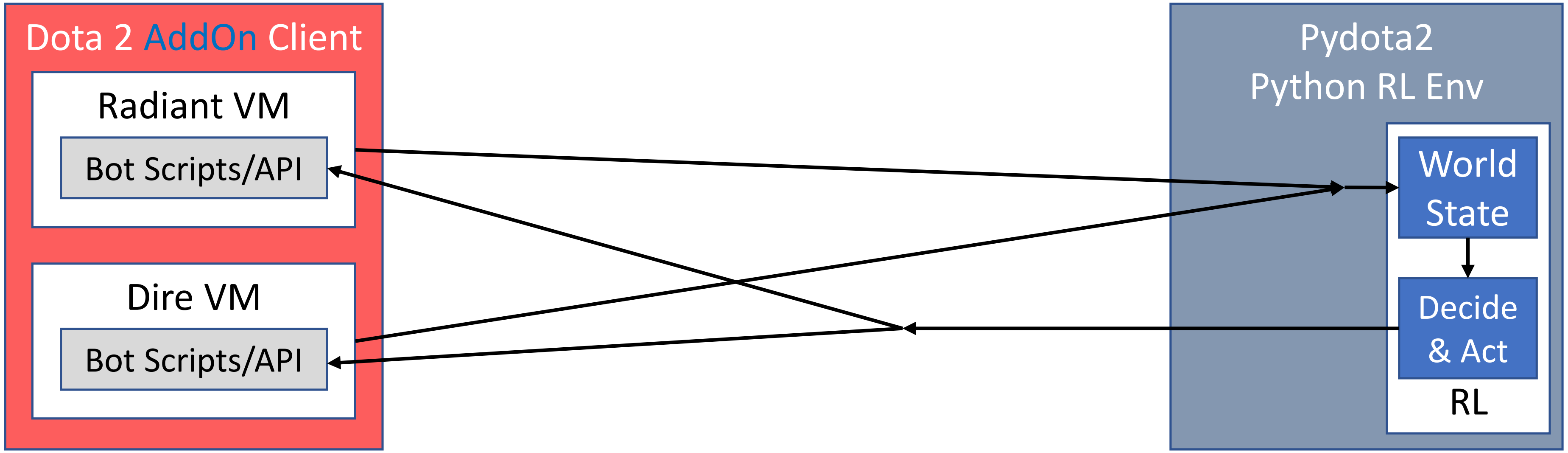
pydota2/bin/self_play.py & pydota2/bin/human_play.py



This is to allow self-play and human-play.

- In human-play the system will play just one side of the game (Radiant or Dire): meaning obtain only one side's protobuf data and connect to that side's POST messages
- In self-play the system will play both sides of the game by creating two processes, one for each side, and merging the improvements to policy & valuation graphs at end of game
- ✓ represent completed components, ✖ represent not started/completed components

NOT YET STARTED: *pydota2/bin/custom_game_training.py*



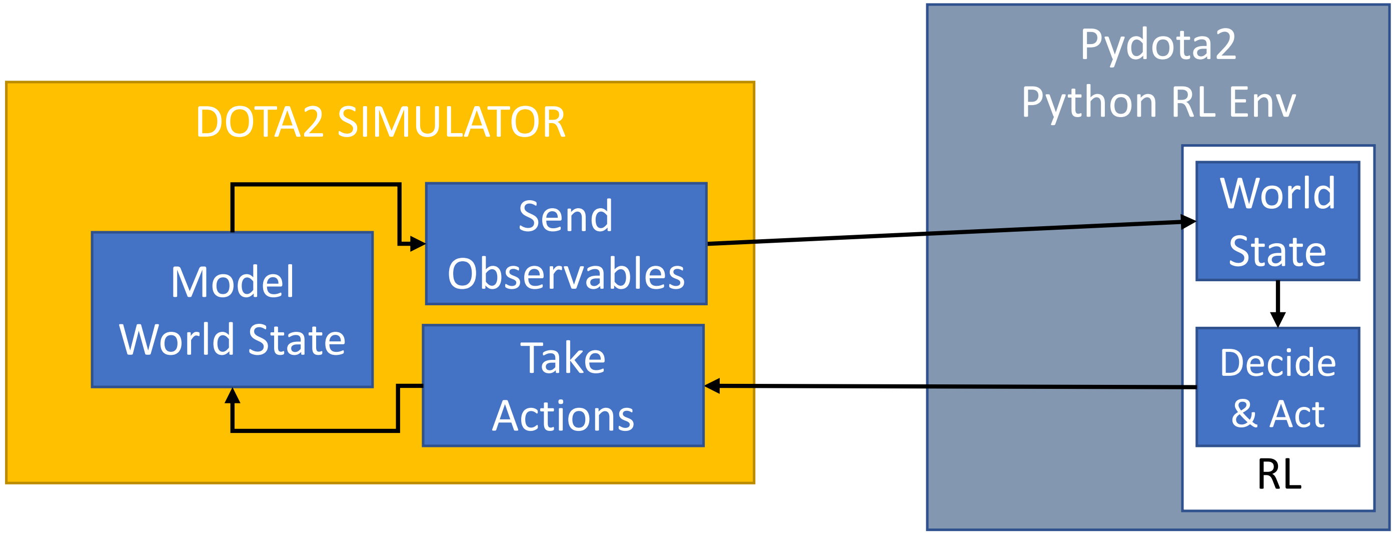
This is designed to create specific training-scenarios for our system to explore more quickly and in more exhaustive depth.

Example: creep blocking, specific ability/item use, team-fight dynamics, last-hitting, etc.

It is anticipated that the custom add-on environment, while using Dota2 data, will be incomplete in scope and time to the actual Dota2 game.

Benefits: quick resets, allows guided episodic learning, more controlled

NOT YET STARTED: *pydota2/bin/simulator.py*



This is designed to represent the Dota2 game through a simulator (say C++) and, like the custom_game_training mode, would be used to train specific scenarios.
Benefits: headless, easily parallelizable

DANGER: bad simulation of Dota2 will result in bad trained policy/value neural nets

So… which mode to use?

- All/Most of them will be needed
    - Dota2 has a frequent patch/update cycle
        - Ability/Item use-cases need to be re-trained
        - Map changes: positioning/pathing understanding will require re-training
    - Dota2 cannot be easily run in parallel
        - Not headless
        - Huge amount of rich information → huge state space… need to abstract it down
        - Games take a long time to complete → time is a resource… it’s not infinite

RL Thoughts: On Policy

- An approach would be to create multi-layered Neural Nets
  - High-Level “Objective” NN
    - Controls Team Desire to take Team Actions (e.g., Roshan, Defend a Lane, Push a Lane, Use Shrine, Check Rune, Split Push, Go Jungle, Use Global Ability) even if not all heroes will take part
    - Controls common Team Resources (e.g., Courier, Individual Hero’s purchase decisions, Smoke usage, Vision Needs through wards, Glyph use, Ping use)
  - Lane-Level “Objective” NN
    - Controls Role-based Hero Actions for a given “lane” (e.g., Ganking, Baiting, Zoning, Kill Wombo-Combos)
  - Hero NN
    - Controls localized environment actions (e.g., last-hitting, jungling, denying, neutral stacking/pulling, maintaining lane equilibrium, effective ability/item use)

RL Thoughts: On World State Representation

- Dota2 has a large hero/ability/item pool – ideally we train our Policy in agnostic methods
    - Abilities can be abstractly represented in terms of ArgumentTypes they take and cause/effect models
    - Items are effectively “abilities” and can be done similarly
    - Heroes can be represented by their health, mana, regen, bounding box size, attack speed & range, movement speed, turn rate, primary attribute, attribute gain per level, etc.
    - Heroes can be thought as having (or not having) “abilities” and “items”