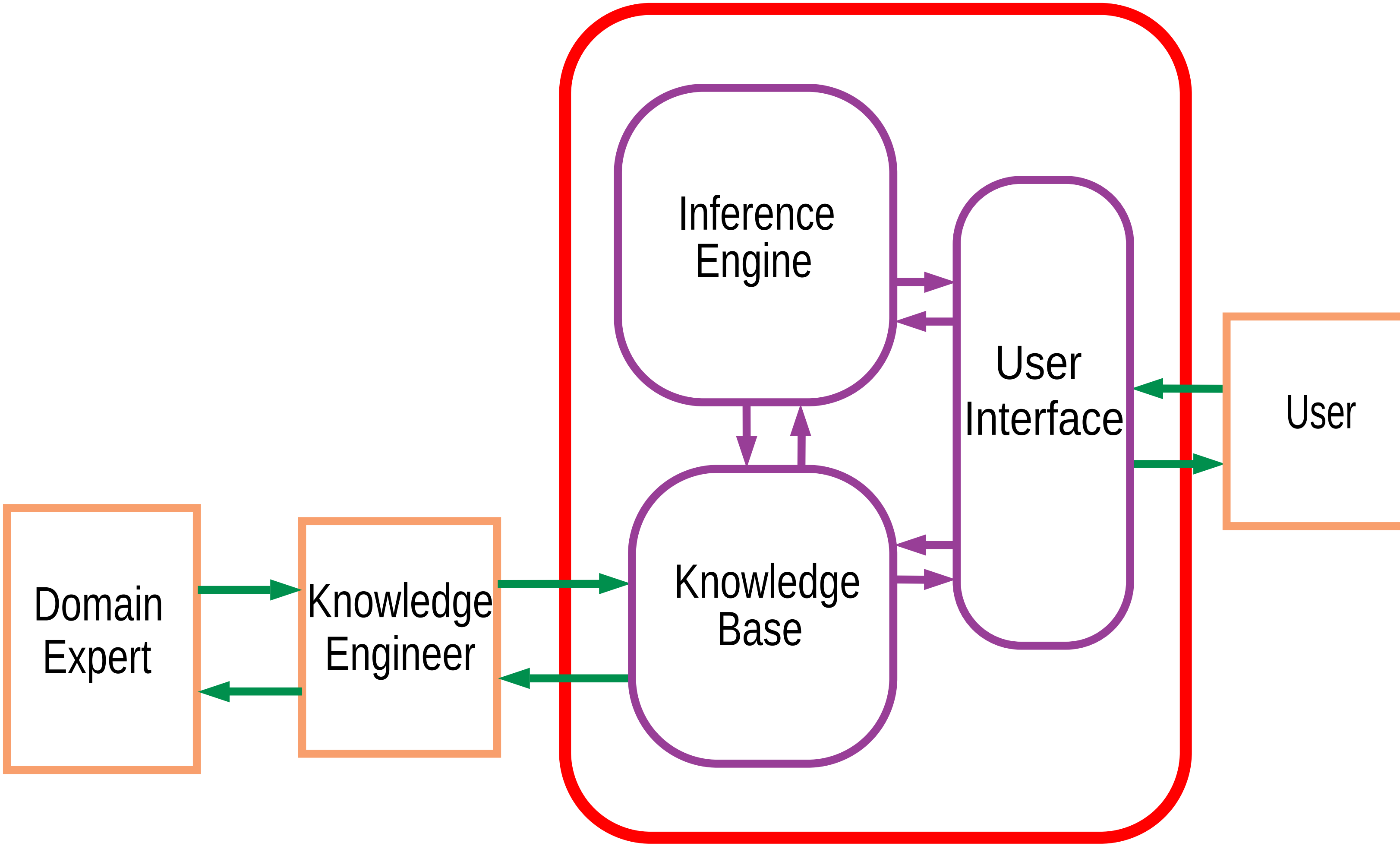
Knowledge Engineering

Overview:

- How representation and reasoning systems interact with humans.
- Roles of people involved in a RRS.
- Building RRSs using meta-interpreters.
- Knowledge-based interaction and debugging tools

Knowledge-based system architecture



Roles for people in a KBS

- Software engineers build the inference engine and user interface.
- Knowledge engineers design, build, and debug the knowledge base in consultation with domain experts.
- Domain experts know about the domain, but nothing about particular cases or how the system works.
- Users have problems for the system, know about particular cases, but not about how the system works or the domain.

Implementing Knowledge-based Systems

To build an interpreter for a language, we need to distinguish

- **Base language** the language of the RRS being implemented.
- **Metalanguage** the language used to implement the system.

They could even be the same language!

Implementing the base language

Let's use the definite clause language as the base language and the metalanguage.

- We need to represent the base-level constructs in the metalanguage.
- We represent base-level terms, atoms, and bodies as meta-level terms.
- We represent base-level clauses as meta-level facts.
- In the non-ground representation base-level variables are represented as meta-level variables.

Representing the base level constructs

- Base-level atom $p(t_1, \ldots, t_n)$ is represented as the meta-level term $p(t_1, \ldots, t_n)$.
- Meta-level term $oand(e_1, e_2)$ denotes the conjunction of base-level bodies e_1 and e_2.
- Meta-level constant *true* denotes the object-level empty body.
- The meta-level atom $clause(h, b)$ is true if "h if b" is a clause in the base-level knowledge base.

Example representation

The base-level clauses

$$connected\_to(l_1, w_0).$$

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$

$$lit(L) \leftarrow light(L) \land ok(L) \land live(L).$$

can be represented as the meta-level facts

$$clause(connected\_to(l_1, w_0), true).$$

$$clause(connected\_to(w_0, w_1), up(s_2)).$$

$$clause(lit(L), oand(light(L), oand(ok(L), live(L)))).$$

Making the representation pretty

- Use the infix function symbol "&" rather than *oand*.
  - instead of writing $oand(e_1, e_2)$, you write $e_1 \mathbin{\&} e_2$.
- Instead of writing $clause(h, b)$ you can write $h \Leftarrow b$, where $\Leftarrow$ is an infix meta-level predicate symbol.
  - Thus the base-level clause "$h \leftarrow a_1 \wedge \cdots \wedge a_n$" is represented as the meta-level atom $h \Leftarrow a_1 \mathbin{\&} \cdots \mathbin{\&} a_n$.

Example representation

The base-level clauses

$$connected\_to(l_1, w_0).$$

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$

$$lit(L) \leftarrow light(L) \wedge ok(L) \wedge live(L).$$

can be represented as the meta-level facts

$$connected\_to(l_1, w_0) \Leftarrow true.$$

$$connected\_to(w_0, w_1) \Leftarrow up(s_2).$$

$$lit(L) \Leftarrow light(L) \,\&\, ok(L) \,\&\, live(L).$$