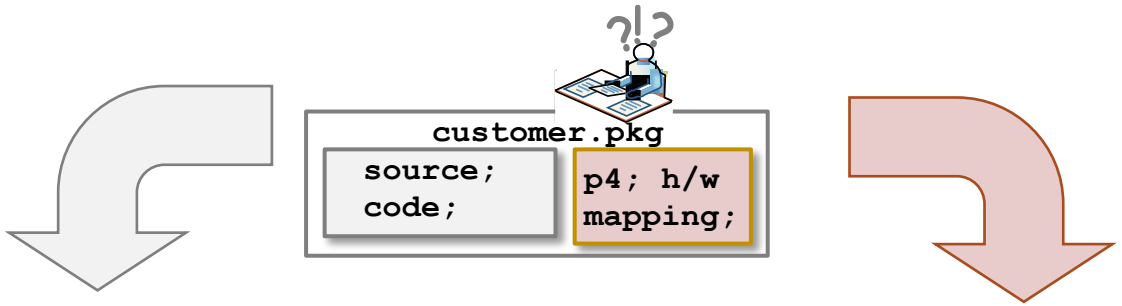
P4 Designer

Pratap Pellakuru, Sharmila S, Michael Orr, Bob Beard
Intel

Problem Statement

- **P4 popularity is growing and there is a demand to extend support to multiple architectures**
- **There is not necessarily a 1-1 mapping between P4 elements and hardware blocks**
- **BIG gap between being a "P4 Programmer" and a "P4 Programmer for a particular product"**
  - Developer needs to know inner workings of hardware at the detailed/nuanced level
  - Developer must tell the compiler how to map P4 to hardware blocks
- **RESULT: Even a knowledgeable P4 programmer with experience in another packet-processor has a significant learning period to become productive for another product**
- **Customer Experience: Even after the learning period, confidence levels are low while deploying P4**

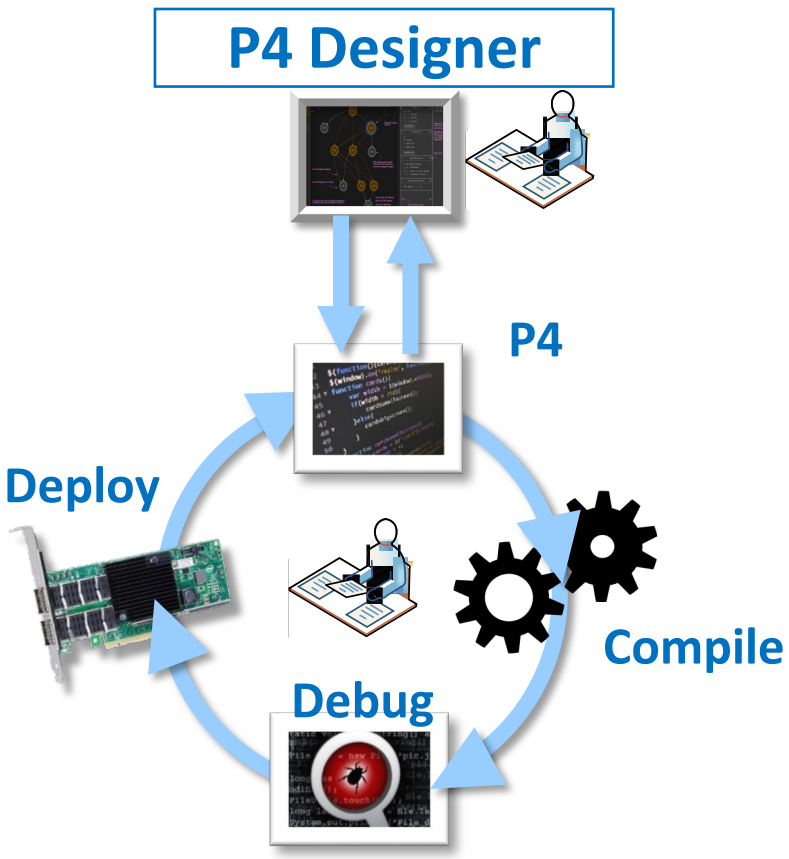


Writing P4 requires deep understanding of hardware architecture.

- Block order, parallelism
- Information in built-in metadata
- Architectural capabilities per each block
- Packet modification and drop semantics
- Recirculation behavior
- Many, many more

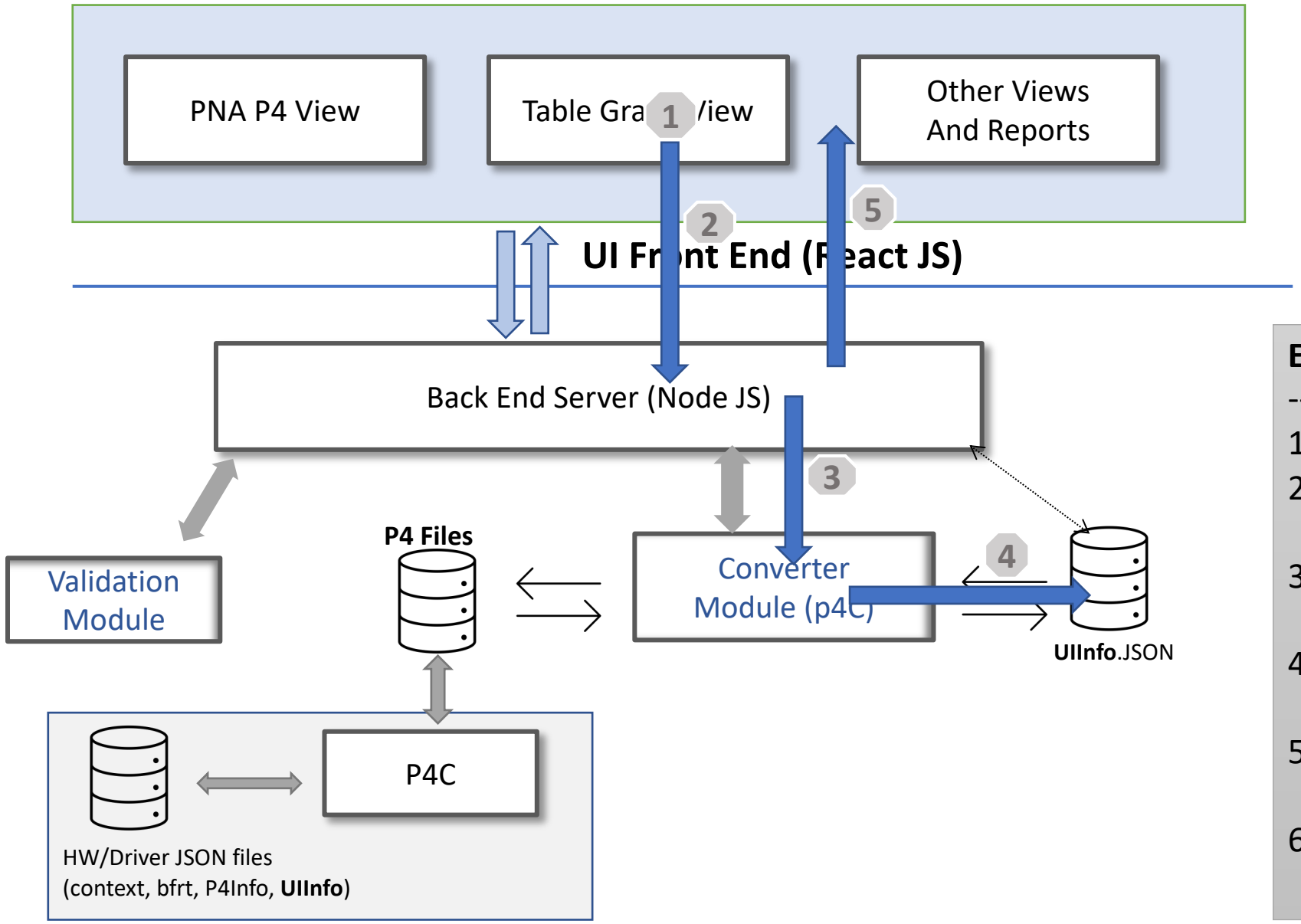
Compiler needs lot of help to figure out the exact hardware mapping

P4 Designer Requirements



- All vendors have the same problem. Win the ease-of-Use battle
  - Make it easy for customer to navigate P4 to hardware mapping
  - Hiding hardware details from P4 designers as much as possible
  - Protect customers from necessary changes to compiler for hardware mapping and improve backward compatibility
- Visual representation and advantages
  - Visually displaying logical table to hardware mapping and other relevant hardware details per node or block. Finding that information through P4 files isn't trivial for customer or P4 designer
  - Auto generate P4 code while restoring rest of the P4 from the GUI modifications
  - Validate mapping and inter dependencies early in the process
  - Make relevant parts of P4 for a table or parser node editable from UI

Architecture – Loading existing P4 (P4 → GUI)

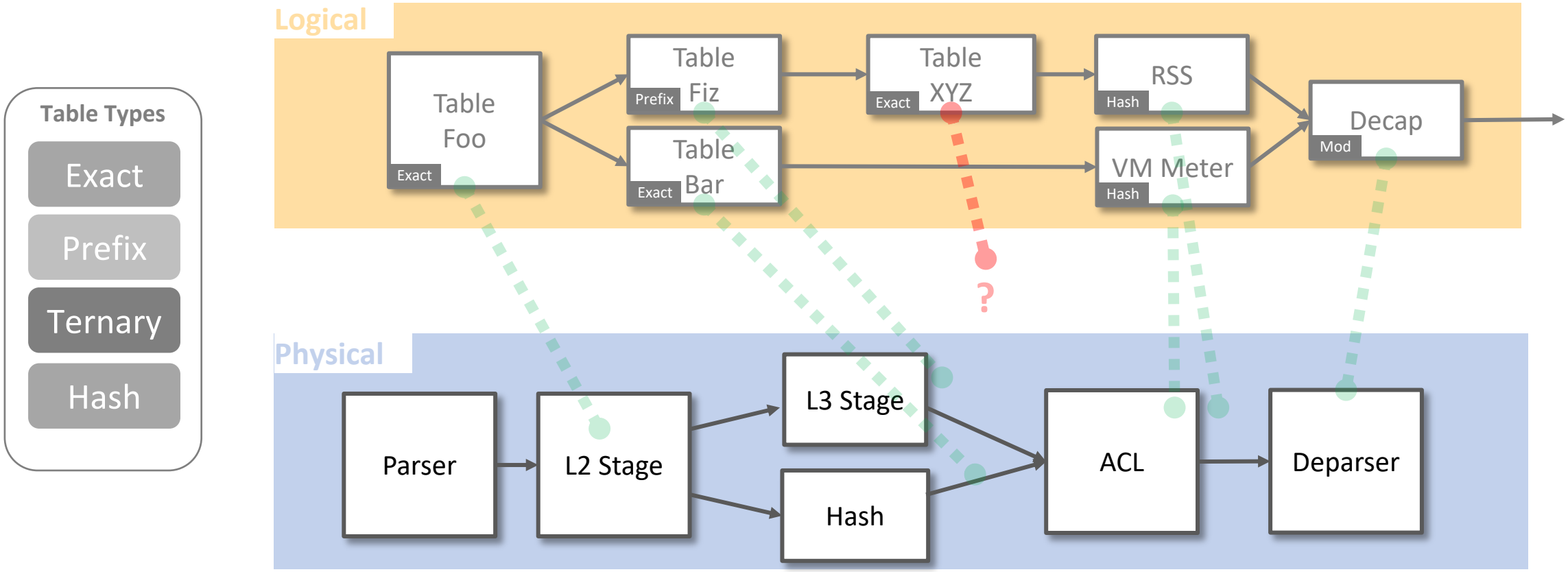


Event flow for Loading P4

---

1. Select P4 from UI
2. Front End sends request to Back End (GET)
3. Back End invokes converter module (P4C)
4. Converter Module generates JSON
5. Back End sends JSON to Front End
6. Front End to parse and display

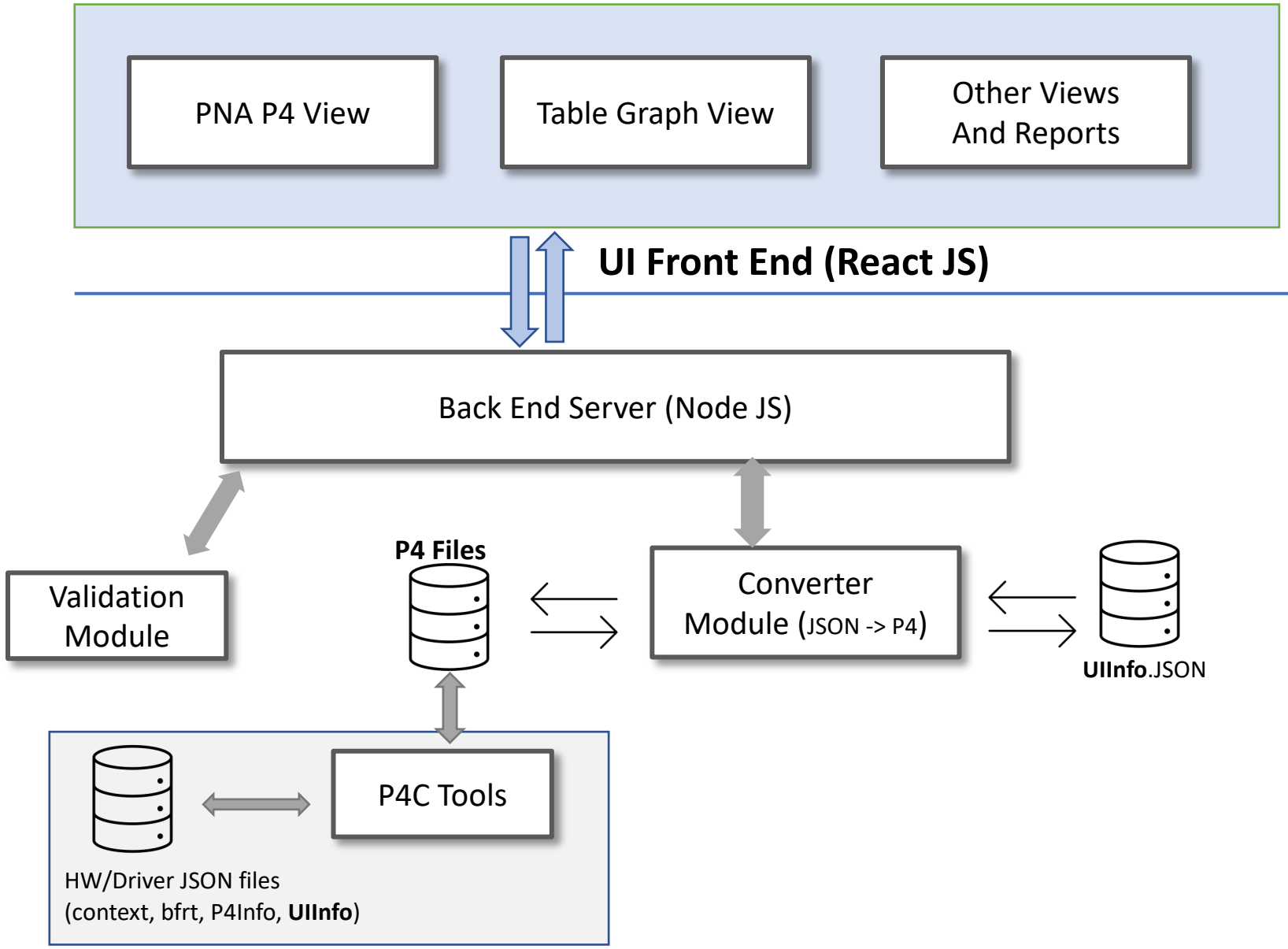
Demo Example: Mapping Workloads to Hardware



Thank You

Backup

Generate/Edit/Restore P4 (GUI → P4)



Event flow for Generating P4

---

1. Add or edit an entry in UI
2. Front End updates JSON
3. Front End sends update to Back End (POST)
4. Back End invokes converter module
5. Converter Module generates P4 from JSON
6. Back End sends P4 to Front End
7. Front End to parse and display

Parser Graph