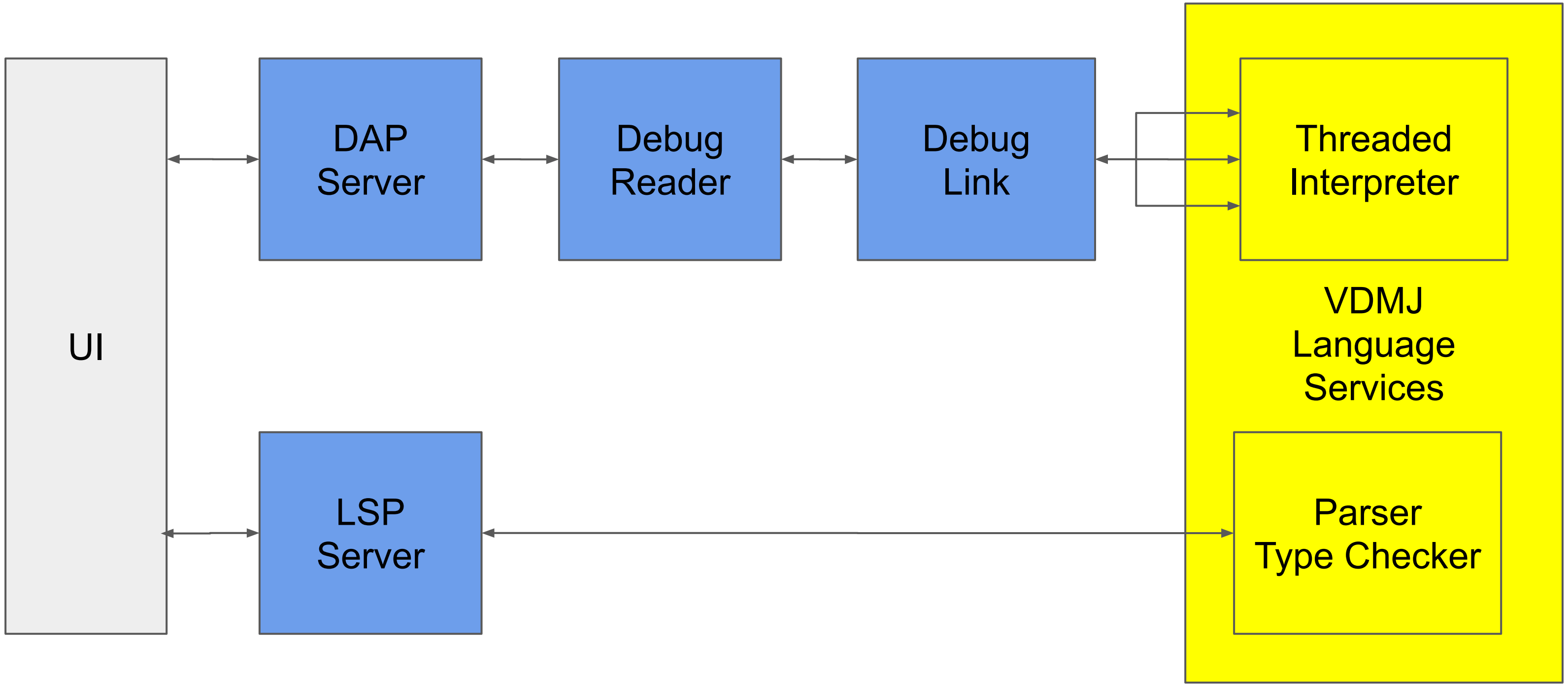
VDMJ Debugging

Nick Battle, July 2022

VDMJ Debugging



VDMJ Debugger

- Two distinct connections from the "IDE" to VDMJ
  - The main parse/typecheck/execute link
  - The debug link
- Corresponds to LSP and DAP protocols
- Debugging traps into a "DebugLink"
  - *DebugLink* interface defined by VDMJ
  - Implementation defined by the IDE and instantiated via a property, *vdmj.debug.link_class*
  - One instance of the debug link created (singleton)
  - Singleton used to communicate with all stopped threads
  - IDE uses a *DebugReader* thread to communicate with *DebugLink*

VDMJ Debugger

- *DebugLink* defines abstract methods for key thread events
- *stopped, breakpoint, tracepoint* called via Breakpoints
- *newThread/complete* called via *SchedulableThread body*
- *DebugLink* also allows abstract *DebugCommands*, like "STACK"

```
abstract public void newThread(CPUValue cpu);
abstract public void stopped(Context ctxt, LexLocation location, Exception ex);
abstract public void breakpoint(Context ctxt, Breakpoint bp);
abstract public void tracepoint(Context ctxt, Tracepoint tp);
abstract public void complete(DebugReason reason, ContextException exception)

protected DebugCommand readCommand(SchedulableThread thread)
protected void writeCommand(SchedulableThread thread, DebugCommand response)
```

VDMJ Debugger

- Every *INExpression* and *INStatement* has a *Breakpoint* field
- The *check* method is called on entry
- Set to a *Breakpoint* object by default
  - Allows global "pause" or "terminate" via UI
  - Checks single-step using Context *threadState*
  - Calls *DebugLink "stopped"* to stop on exceptions, else "breakpoint"
  - A stop forces other threads to call "stopped" too
- Can be set to a *Stoppoint*
  - Unconditionally stop at this point (ie. a user breakpoint)
  - Optionally allows hit counts and conditions
  - Calls *DebugLink "breakpoint"* to stop
- Can be set to a *Tracepoint*
  - Log that execution reached this point, but don't stop
  - Calls *DebugLink "tracepoint"*

VDMJ Debugger

- *DebugLink's waitForStop* method called by *DebugReader*
- Each call to *stopped* or *breakpoint* updates state
- When all threads are stopped, *waitForStop* returns to reader
- *DebugReader* then waits for IDE instructions
    - eg. set a new breakpoint or step or continue
    - Some commands sent to thread via *sendCommand*
    - Threads are waiting on *readCommand* in *stopped* method
    - Thread side command handled by a *DebugExecutor*
    - On continue, all stopped threads are sent *RESUME*
    - Then reader calls *waitForStop* again.