

# Towards TLAPS IDE

Karolis Petrauskas

Vilnius University

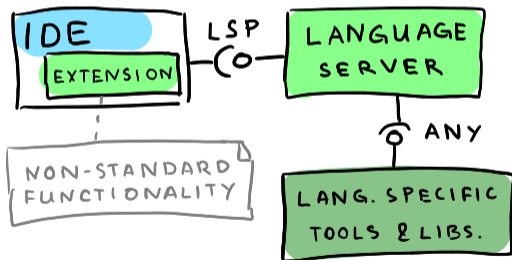
TLA<sup>+</sup> Community Event  
2024-09-10

## Why new IDE is needed for TLAPS?

- ▶ The Eclipse based TLA<sup>+</sup> Toolbox is not maintained anymore.
- ▶ The VSCode extension is the mainstream currently.
- ▶ It had no support for TLAPS.

## Why new IDE is needed for TLAPS?

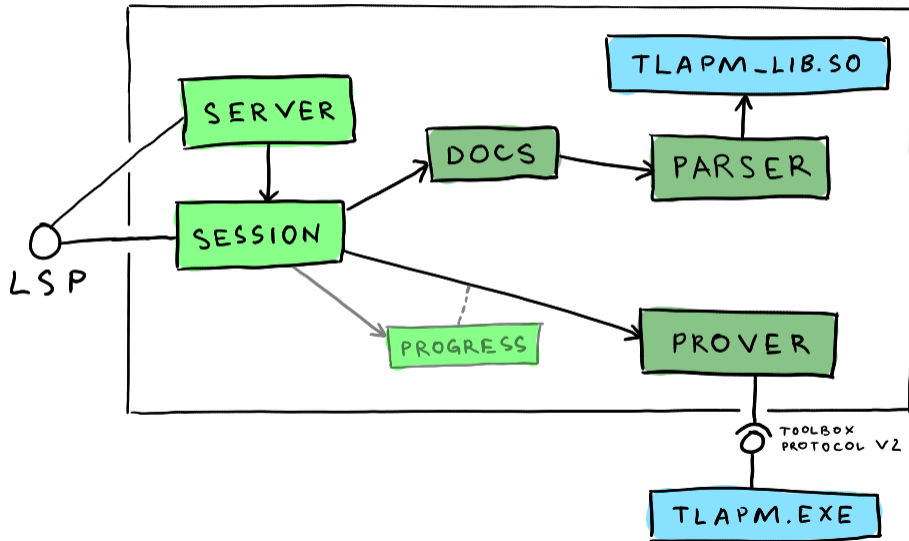
- ▶ The Eclipse based TLA<sup>+</sup> Toolbox is not maintained anymore.
- ▶ The VSCode extension is the mainstream currently.
- ▶ It had no support for TLAPS.



## TLAPS LSP supports:

- ▶ Command execution;
- ▶ Diagnostics;
- ▶ Code actions;
- ▶ Several extensions.

# Main structure of the LSP



## How to present the proof state?

# Proof state presentation

```

<2>6. CASE ServerRecvQRY
  <3> USE <2>6 DEF sendAndAck, updMsgs
  <3> PICK m \in msgs : ServerRecvQRY!(m) BY DEF ServerRe
  <3> SUFFICES ASSUME ~InSyncOrHaveMsgs!1' PROVE InSyncOr
  <3>1. CASE m.h_r = H(sValue) BY <3>1 DEF H
  <3>2. CASE m.h_r # H(sValue) BY <3>2 DEF H
  <3>3. QED BY <3>1, <3>2
<2>q. QED BY <2>1, <2>2, <2>3, <2>4, <2>5, <2>6 DEF Next
  
```

TLAPS checks more proof obligations than there are proof steps.

# Proof state presentation

```

<2>6. CASE ServerRecvQRY
  <3> USE <2>6 DEF sendAndAck, updMsgs
  <3> PICK m \in msgs : ServerRecvQRY!(m) BY DEF ServerRe
  <3> SUFFICES ASSUME ~InSyncOrHaveMsgs!1' PROVE InSyncOr
  <3>1. CASE m.h_r = H(sValue) BY <3>1 DEF H
  <3>2. CASE m.h_r # H(sValue) BY <3>2 DEF H
  <3>3. QED BY <3>1, <3>2
<2>q. QED BY <2>1, <2>2, <2>3, <2>4, <2>5, <2>6 DEF Next
  
```

We group them to proof steps for presentation. Take the "worst" state as an aggregate.

# Proof state presentation

```
<2>6. CASE ServerRecvQRY
  <3> USE <2>6 DEF sendAndAck, updMsgs
  <3> PICK m \in msgs : ServerRecvQRY!(m) BY DEF ServerRe
  <3> SUFFICES ASSUME ~InSyncOrHaveMsgs!1' PROVE InSyncOr
  <3>1. CASE m.h_r = H(sValue) BY <3>1 DEF H
  <3>2. CASE m.h_r # H(sValue) BY <3>2 DEF H
  <3>3. QED BY <3>1, <3>2
<2>q. QED BY <2>1, <2>2, <2>3, <2>4, <2>5, <2>6 DEF Next
```

Steps aggregated up to a module.



# Proof state markers



```
Example ⓘ
1 ---- MODULE Example ----
2 THEOREM \A a, b: a /\ b => a \/ b
3 <1>1. TAKE a, b
4 <1>2. HAVE a /\ b
5 <1>q. QED BY <1>1
6 =====
```

Example.tla > {} Example

✗	1	---- MODULE Example ----
✗	2	THEOREM \A a, b: a /\ b => a \/ b
✓	3	💡 <1>1. TAKE a, b
✓	4	<1>2. HAVE a /\ b
✗	5	<1>q. QED BY <1>1
	6	=====

**More Actions...**

💡 Check proofs on lines 2-5

## How to trace / explain a proof?

# Current proof step



File Edit Selection View Go Run Terminal Help

```
TLA+  
  
▼ CURRENT PROOF STEP  
Leaf proof step ✓ at Example.tla 5:3  
Obligation ✓ at 5:13  
  
✓ smt[time-limit: 5]  
  
ASSUME NEW CONSTANT a,  
      NEW CONSTANT b,  
      a /\ b  
PROVE  a \/ b
```

Shown for all steps. Helps in understanding and explaining proofs.

# Current proof step



File Edit Selection View Go Run Terminal Help

The screenshot shows a software interface for a TLA+ proof. At the top, a menu bar contains "File", "Edit", "Selection", "View", "Go", "Run", "Terminal", and "Help". Below the menu bar is a light gray panel. On the left side of this panel are three icons: a document icon, a magnifying glass icon, and a diamond icon with a plus sign. To the right of these icons, the text "TLA+" is displayed. Below "TLA+" is a dropdown arrow followed by the text "CURRENT PROOF STEP". Underneath that, the text "Structured proof" is followed by a red "x" and "at Example.tla 2:1". Below this, the text "Steps:" is followed by a green checkmark, the number "2", a red "x", and the number "1".

Overview of a structured proof step.

# Current proof step



File Edit Selection View Go Run Terminal Help

TLA+ ...

▼ **CURRENT PROOF STEP**

**Module** x at **Example.tla 1:1**

**Theorems:** 2 x 1

Overview of a module.

# Current proof step



File Edit Selection View Go Run Terminal Help

TLA+ ...

▼ CURRENT PROOF STEP

**Leaf proof step** × at Example.tla 5:3

**Obligation** × at 5:13

× smt[time-limit: 5]:  
(false)

× zenon[time-limit: 10]:  
(false)

× isabelle[auto; time-limit: 30]:  
(false)

ASSUME NEW CONSTANT a,  
NEW CONSTANT b,  
a  $\Rightarrow$  b  
PROVE a  $\vee$  b

**Obligation[aux]** × at 5:16

× smt[time-limit: 5]:  
(false)

× zenon[time-limit: 10]:  
(false)

× isabelle[auto; time-limit: 30]:  
(false)

TeX

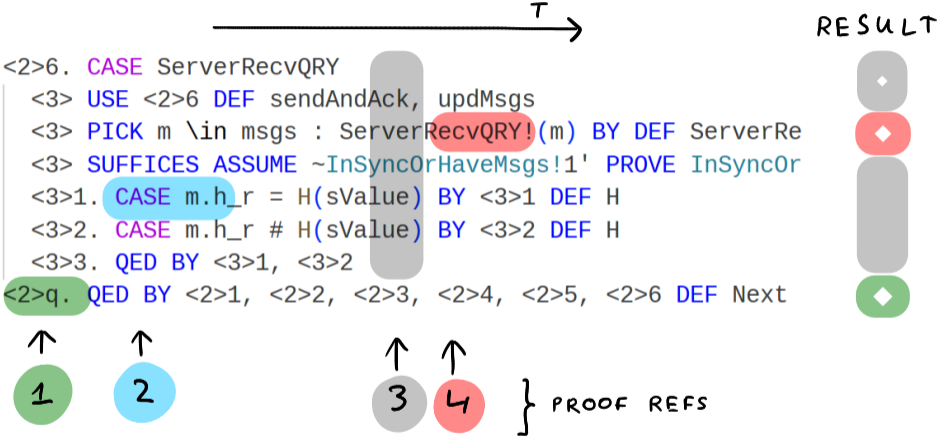
ASSUME NEW CONSTANT a,  
NEW CONSTANT b  
PROVE a  $\Rightarrow$  b

...

Auxiliary obligations  
only shown if failed.

**How to maintain edits and proof checks in parallel?**


# Accumulating proof check results



Within a single document version the proof results are overlaid by document lines.



# Retaining the proof state between edits

<pre> &lt;2&gt;6. CASE ServerRecvQRY   &lt;3&gt; USE &lt;2&gt;6 DEF sendAndAck,   &lt;3&gt; PICK m \in msgs : ServerR   &lt;3&gt; SUFFICES ASSUME ~InSyncOr   &lt;3&gt;1. CASE m.h_r = H(sValue)   &lt;3&gt;2. CASE m.h_r # H(sValue)   &lt;3&gt;3. QED BY &lt;3&gt;1, &lt;3&gt;2 &lt;2&gt;q. QED BY &lt;2&gt;1, &lt;2&gt;2, &lt;2&gt;3, </pre>		<pre> &lt;2&gt;6. CASE ServerRecvQRY   &lt;3&gt; USE &lt;2&gt;6 DEF sendAndAck,   &lt;3&gt; PICK m \in msgs : Server   &lt;3&gt; SUFFICES ASSUME ~InSyncOr   &lt;3&gt;0. TRUE \* TBD: ...   &lt;3&gt;1. CASE m.h_r = H(sValue)   &lt;3&gt;2. CASE m.h_r # H(FALSE)   &lt;3&gt;3. QED BY &lt;3&gt;1, &lt;3&gt;2 &lt;2&gt;q. QED BY &lt;2&gt;1, &lt;2&gt;2, &lt;2&gt;3, </pre>
---	--	--

# Retaining the proof state between edits

```

<2>6. CASE ServerRecvQRY
  <3> USE <2>6 DEF sendAndAck,
  <3> PICK m \in msgs : ServerR
  <3> SUFFICES ASSUME ~InSyncOr
  <3>1. CASE m.h_r = H(sValue)
  <3>2. CASE m.h_r # H(sValue)
  <3>3. QED BY <3>1, <3>2
<2>q. QED BY <2>1, <2>2, <2>3,
  
```

```

<2>6. CASE ServerRecvQRY
  <3> USE <2>6 DEF sendAndAck,
  <3> PICK m \in msgs : Server
  <3> SUFFICES ASSUME ~InSyncOr
  <3>0. TRUE \* TBD: ...
  <3>1. CASE m.h_r = H(sValue)
  <3>2. CASE m.h_r # H(FALSE)
  <3>3. QED BY <3>1, <3>2
<2>q. QED BY <2>1, <2>2, <2>3,
  
```

$$FP_{3,1} = FP'_{3,1}$$

# Retaining the proof state between edits

```

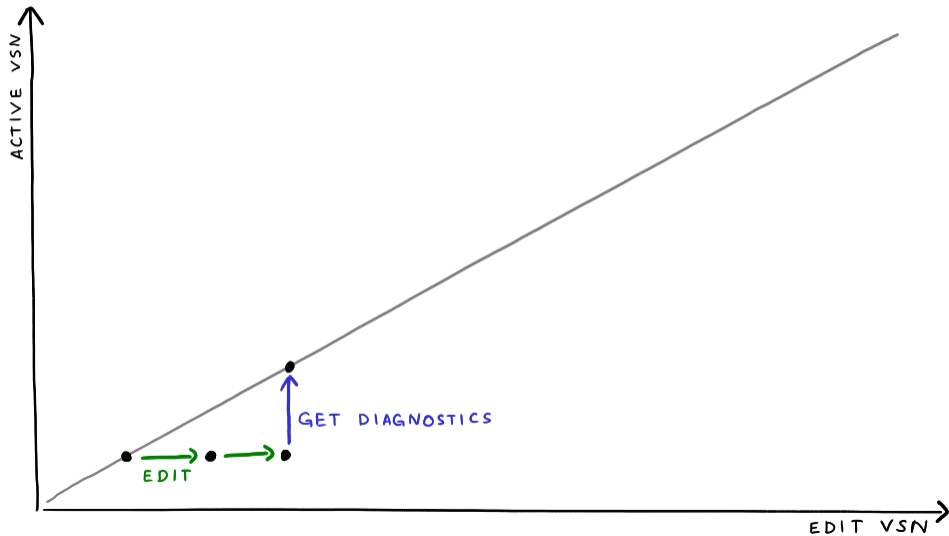
<2>6. CASE ServerRecvQRY
  <3> USE <2>6 DEF sendAndAck,
  <3> PICK m \in msgs : ServerR
  <3> SUFFICES ASSUME ~InSyncOr
  <3>1. CASE m.h_r = H(sValue)
  <3>2. CASE m.h_r # H(sValue)
  <3>3. QED BY <3>1, <3>2
<2>q. QED BY <2>1, <2>2, <2>3,
  
```

```

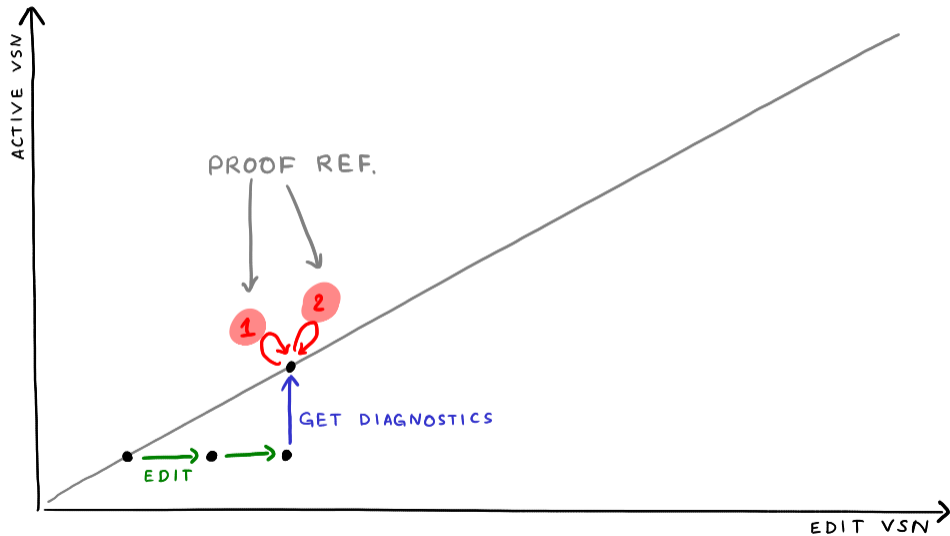
<2>6. CASE ServerRecvQRY
  <3> USE <2>6 DEF sendAndAck,
  <3> PICK m \in msgs : Server
  <3> SUFFICES ASSUME ~InSyncOr
  <3>0. TRUE \* TBD: ...
  <3>1. CASE m.h_r = H(sValue)
  <3>2. CASE m.h_r # H(FALSE)
  <3>3. QED BY <3>1, <3>2
<2>q. QED BY <2>1, <2>2, <2>3,
  
```

$FP_{3.2} \neq FP'_{3.2}$

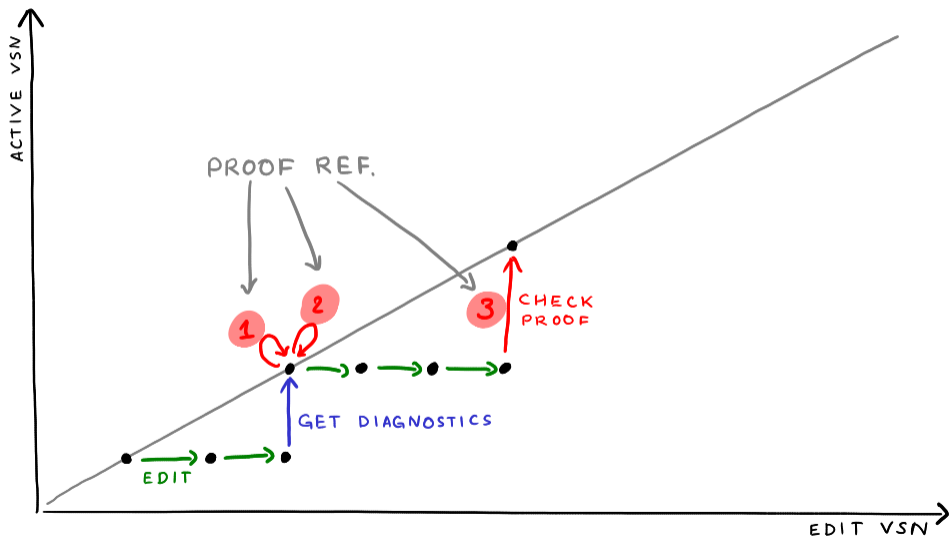
# Edits vs Diagnostics vs Proof checks



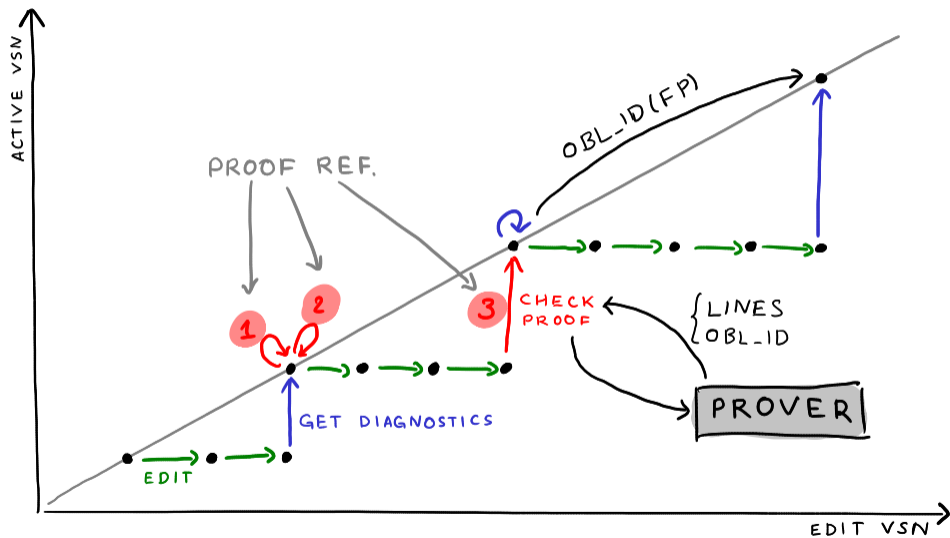
# Edits vs Diagnostics vs Proof checks



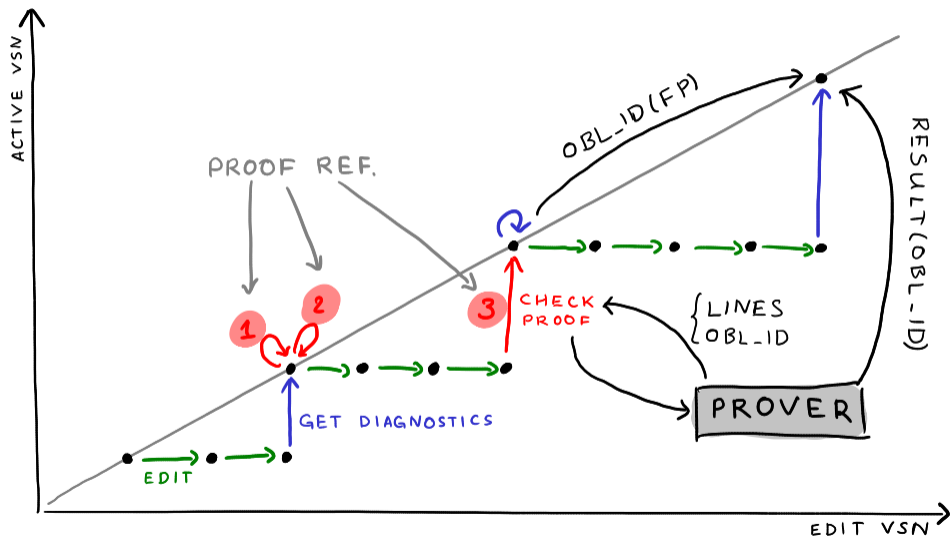
# Edits vs Diagnostics vs Proof checks



# Edits vs Diagnostics vs Proof checks



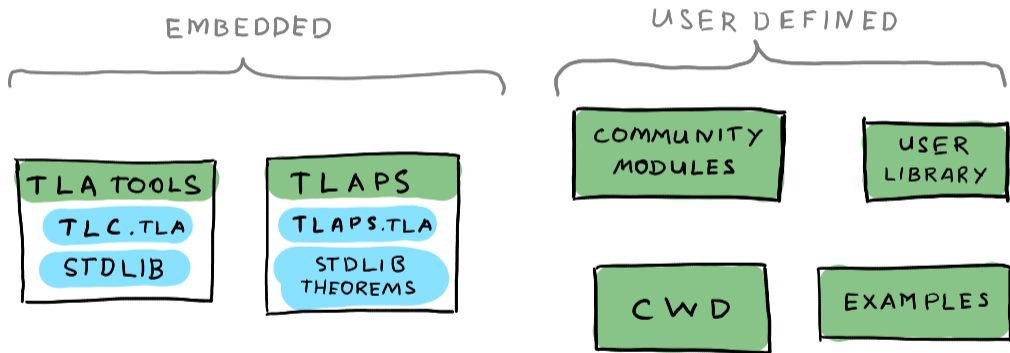
# Edits vs Diagnostics vs Proof checks



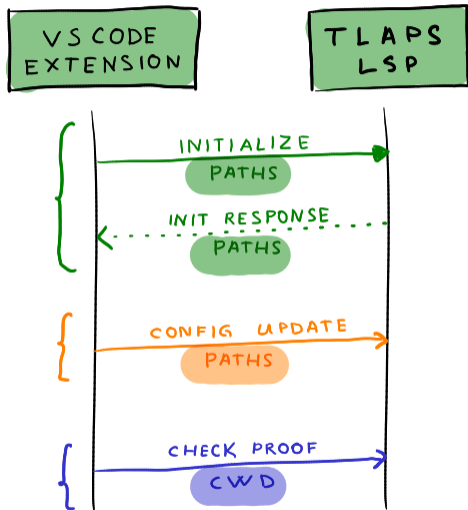


**Where to find the modules?**

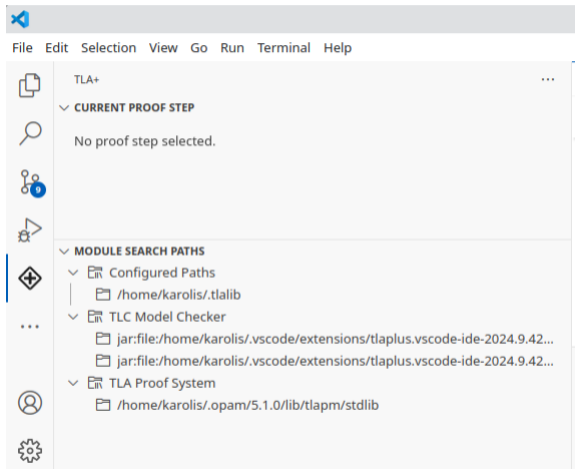
# There are several sources of modules



# TLA+ Extension synchronizes paths with TLAPS



# Paths are visible to the user



In the TLA+ side-panel a module search path view was introduced

- ▶ Modules could be shown here.
- ▶ Overrides / precedence presented explicitly.

# Where the paths should be configured?



User paths are currently configured in the VSCode extension settings.

- ▶ The tool-set should be IDE agnostic.
- ▶ We could introduce `.tlaplus` files and use them similarly as `.gitconfig` or `.tool-versions`.

## Extensions made to the TLAPM

## ► Toolbox Protocol V2.

- Planned provers are reported in V2 to estimate the proof termination.
- Switch added to specify the protocol version.

```
$ tlapm --toolbox 0 0 --toolbox-vsn 2 Example.tla
...
@!!BEGIN
@!!type:obligationprovers
@!!id:1
@!!provers:smt,zenon,isabelle
@!!END
...
```

# Extensions made to the TLAPM

## ▶ Process module from Stdin.

- ▶ Used by the LSP to check a module.
- ▶ Allows the user to check the document without saving it first.

```
$ tlapm --stdin Ephemeral.tla <<EOF
---- MODULE Ephemeral ----
THEOREM \A a, b: a /\ b => a \/ b
PROOF OBVIOUS
====
EOF
...
[INFO]: All 1 obligation proved.
```



# Extensions made to the TLAPM

- ▶ **Modules can be read from ZIP files.**
  - ▶ This was needed to access the modules from TlaTools.
  - ▶ Can be useful to simplify some testcases.

```
$ tlapm -I 'jar:file:tla2tools.jar!/tla2sany/StandardModules' \  
      --stdin Ephemeral.tla <<EOF  
---- MODULE Ephemeral ----  
EXTENDS Toolbox  
====  
EOF  
...  
[INFO]: All 0 obligation proved.
```

# Extensions made to the TLAPM



- ▶ **TLAPS can be interrupted by a signal.**
  - ▶ The process is interrupted if new proof check command is issued.

# Extensions made to the TLAPM



- ▶ **TLAPS can be interrupted by a signal.**
  - ▶ The process is interrupted if new proof check command is issued.
- ▶ **It runs now with a recent Isabelle version.**
  - ▶ The old version was the reason to fail on some recent hardware.

**What's next**

## Future plans

- ▶ **To cover the functions in TLA<sup>+</sup> Toolbox:**
  - ▶ Proof decomposition commands.
  - ▶ Proof step re-numbering command.



- ▶ **To cover the functions in TLA<sup>+</sup> Toolbox:**
  - ▶ Proof decomposition commands.
  - ▶ Proof step re-numbering command.
  
- ▶ **Ongoing developments (student projects):**
  - ▶ Suggest definitions to expand.
  - ▶ Find / suggest related facts by a pattern.
  - ▶ Evaluate the usability and accessibility systematically.

## Future plans

- ▶ **To cover the functions in TLA<sup>+</sup> Toolbox:**
  - ▶ Proof decomposition commands.
  - ▶ Proof step re-numbering command.
  
- ▶ **Ongoing developments (student projects):**
  - ▶ Suggest definitions to expand.
  - ▶ Find / suggest related facts by a pattern.
  - ▶ Evaluate the usability and accessibility systematically.
  
- ▶ **Ideas for improvements:**
  - ▶ Install it from the VSCode. [TLAPM release](#) is needed for that.
  - ▶ Improve proof navigation and visualization. E.g. [links to definitions](#), [tooltips](#).
  - ▶ Show expanded formulas in a foldable way. [To make large formulas more readable](#).
  - ▶ And more.