Terrestrial Verification Methodology

A “Down-To-Earth” Approach to Developing UVM VIPs

Bryan Morris, P.Eng.
Ciena Corp.

April 21, 2017
Canada

Agenda

- Patterns, Patterns… everywhere
- Façade Pattern
- Façade : ‘Terrestrial’ Verification Methodology (TVM)
- Future Work…
- Q&A

I’m a big fan of…

Design Patterns

Pattern	.. Huh?
Factory	Dynamically change classes instantiated
Callback	Add functionality via hooks
State	Finite State Machine (dynamic)
Chain of Responsibility	Handles sequence of transformations
Command	Dynamically change behaviour

This Year's Pattern *du Jour?*

Façade

Façade

Wikipedia Definition:

A **façade** (/fəˈsɑːd/)[1] is generally one exterior side of a building, usually, but not always, the front.

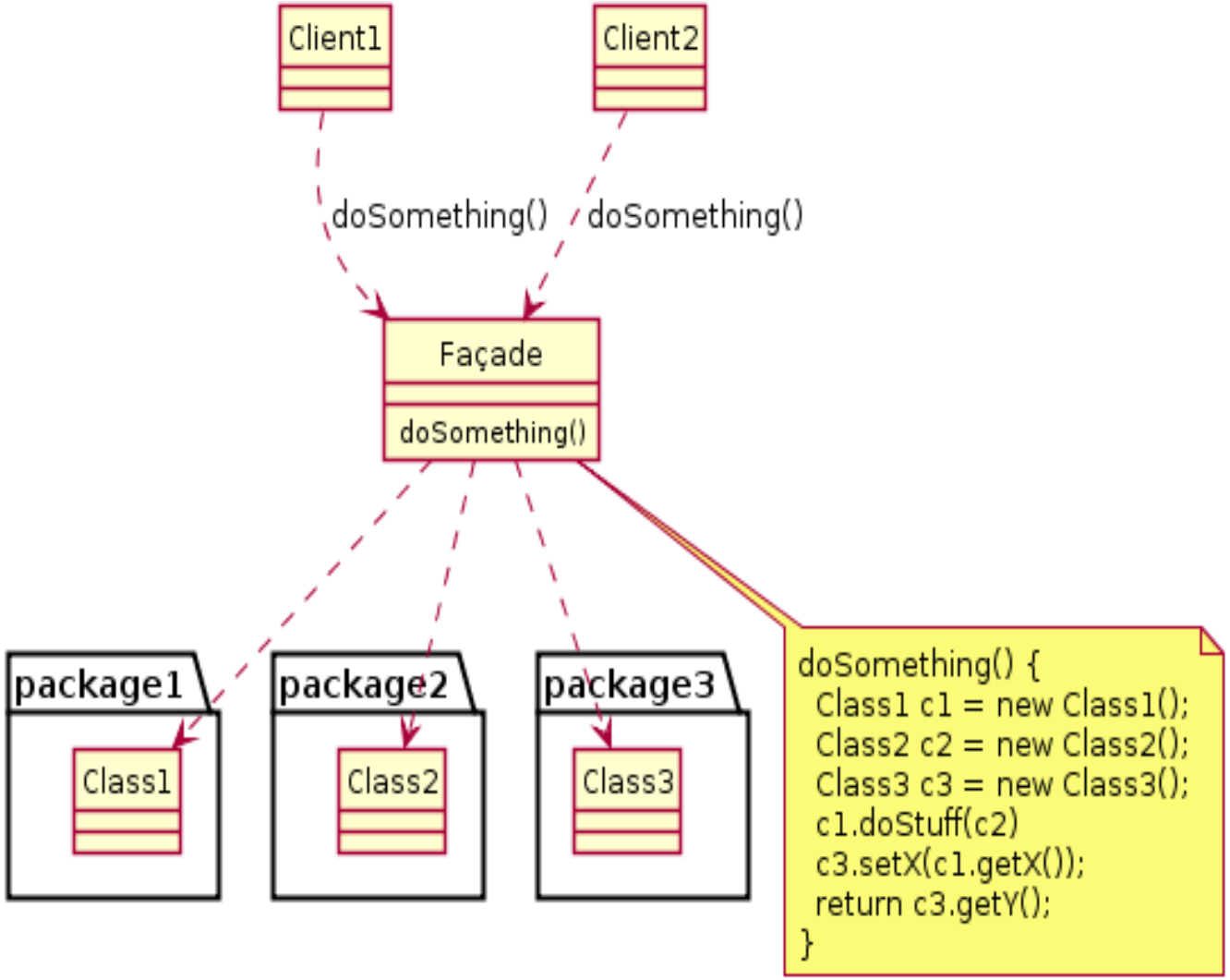
In architecture, the façade of a building is often the most important aspect from a design standpoint, as it sets the tone for the rest of the building

Façade Pattern

Goal:

Adds a simplifying front-end for multiple collaborating classes

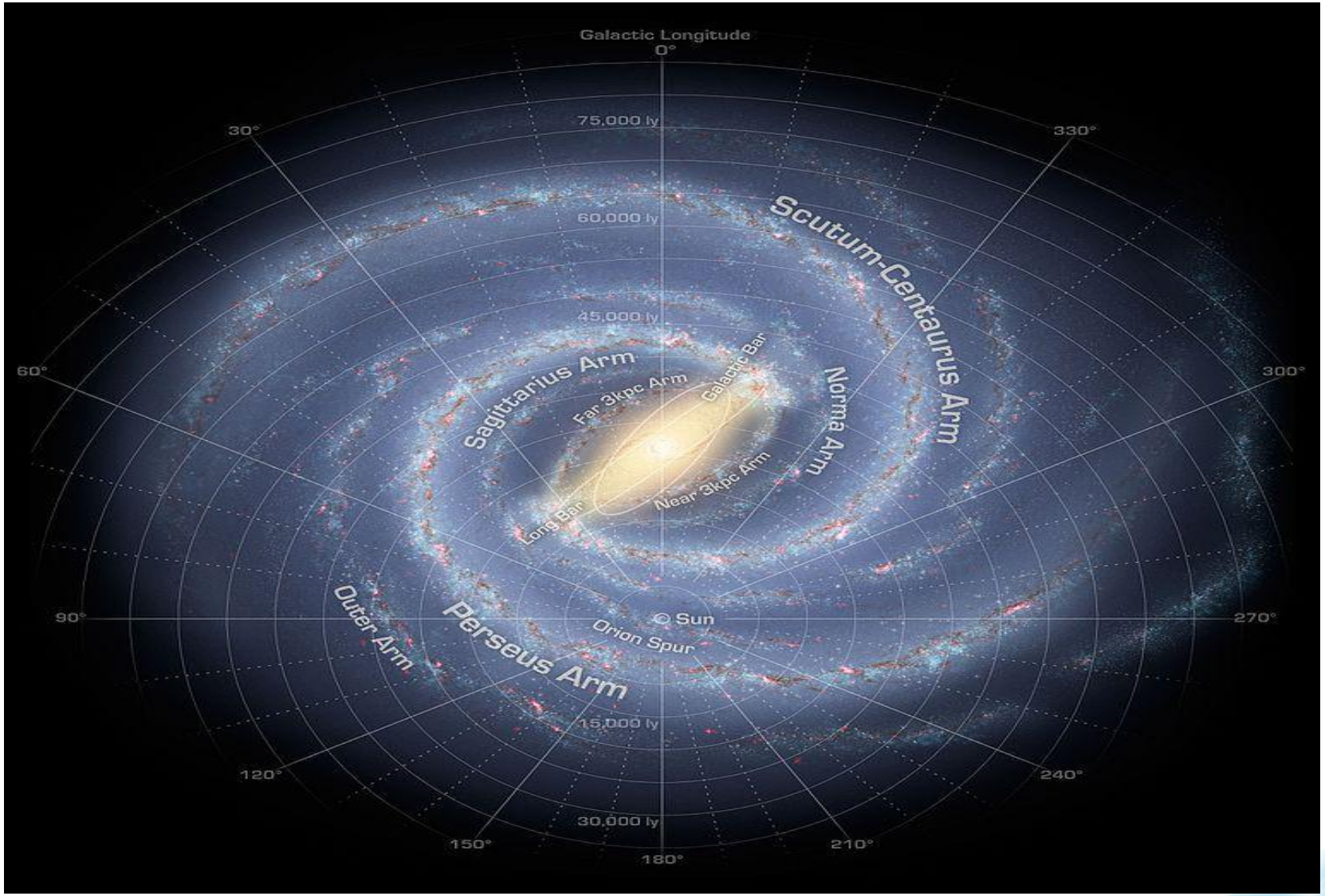
Façade Pattern



Terrestrial Verification Methodology

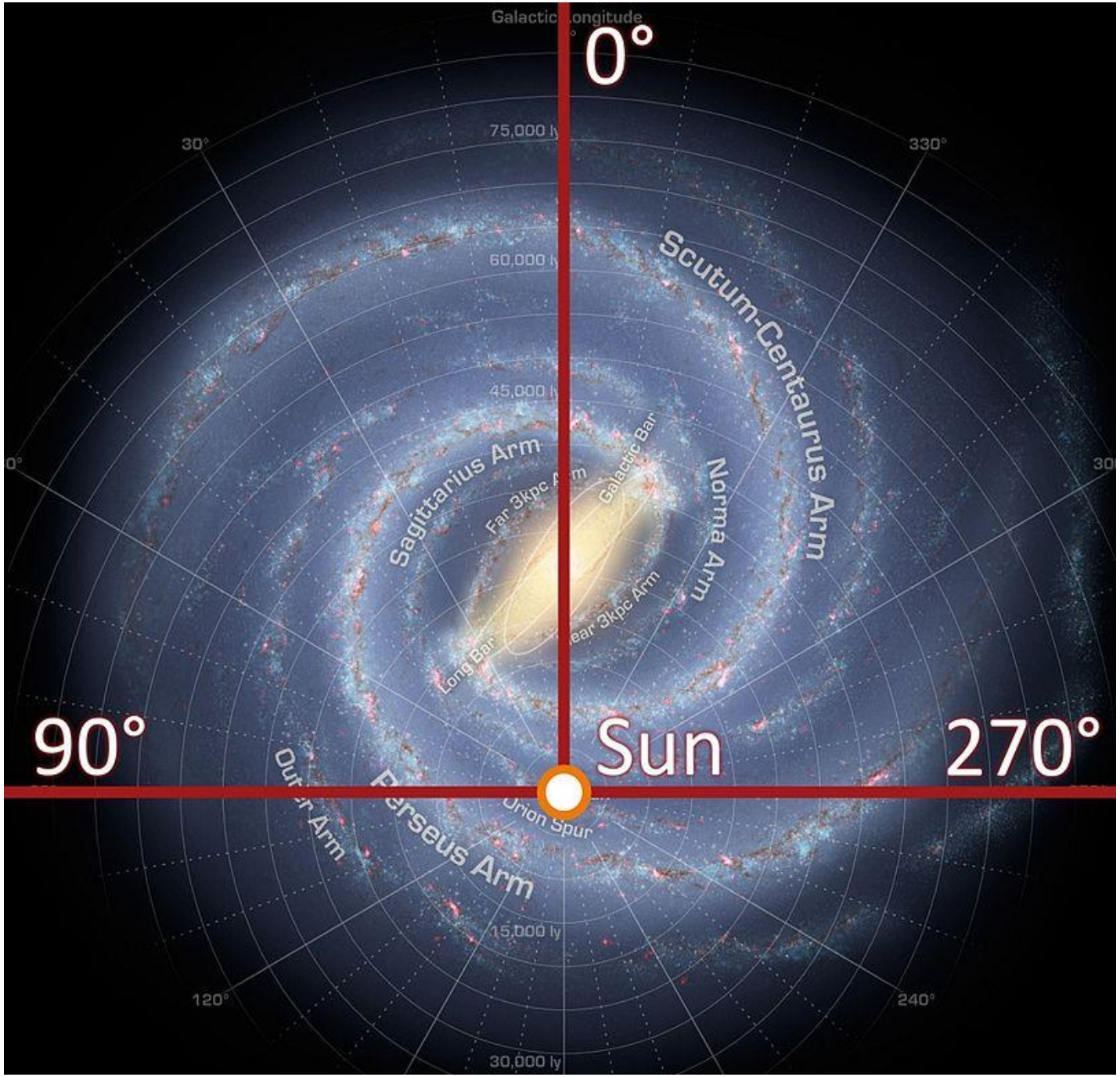
Application of Façade Pattern

Universal
Verification Methodology

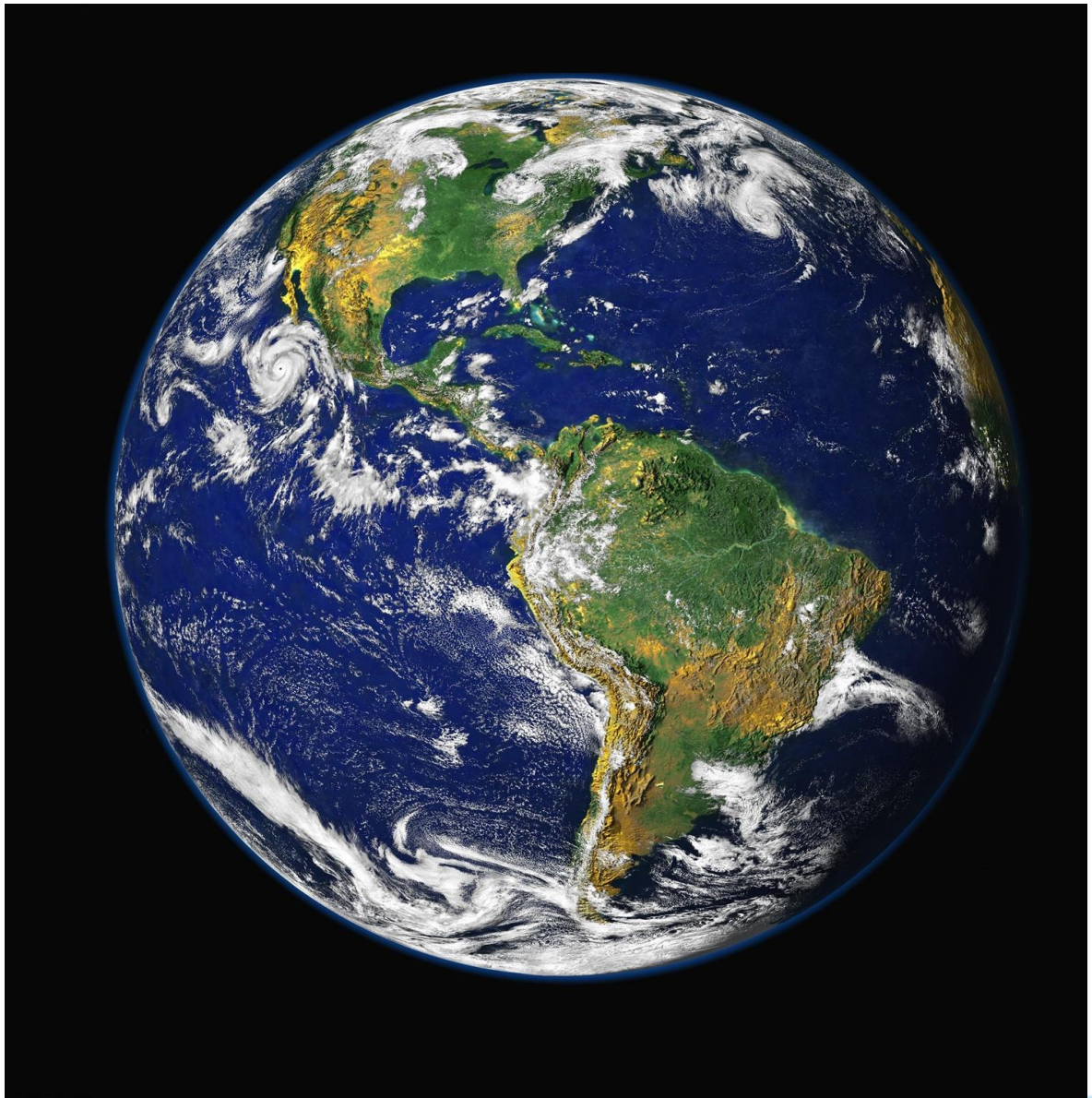


I've *never* said that the name was *pretentious*… nope, not me.

Now…


Galactic Longitude
0°
75,000 ly
30°
330°
60,000 ly
45,000 ly
Scutum-Centaurus Arm
Sagittarius Arm
Far 3kpc Arm
Galactic Bar
Norma Arm
Near 3kpc Arm
Long Bar
90°
Sun
270°
Outer Arm
Perseus Arm
Orion Spur
15,000 ly
120°
240°
30,000 ly

Terrestrial
Verification Methodology



It’s goals are a little more *down to earth*.

Someone needs to…

ciena Experience. Outcomes.

snug Synopsys Users Group Canada

What About UVM?

From a slide-deck presentation on PSS

- **The Bad**
  - Non DV & Designer Engineers are not familiar with System Verilog & UVM
    - Overly complicated and hard to debug
    - Need to be an expert in UVM to create a simple directed test

Someone needs to… look up the word:

ciena
Experience. Outcomes.

snug
Synopsys Users Group
Canada

What About UVM?

- Excellent for Block/IP Level Verification,
  - Does not scale to System Level Verification, Only Solves “Checking” Portability Problem

Someone needs to… look up the word: “irony”

ciena Experience. Outcomes.

snug Synopsys Users Group Canada

What About UVM?

- Excellent for Block/IP Level Verification,
  - Does not scale to System Level Verification, Only Solves “Checking” Portability Problem

Engineering needs.

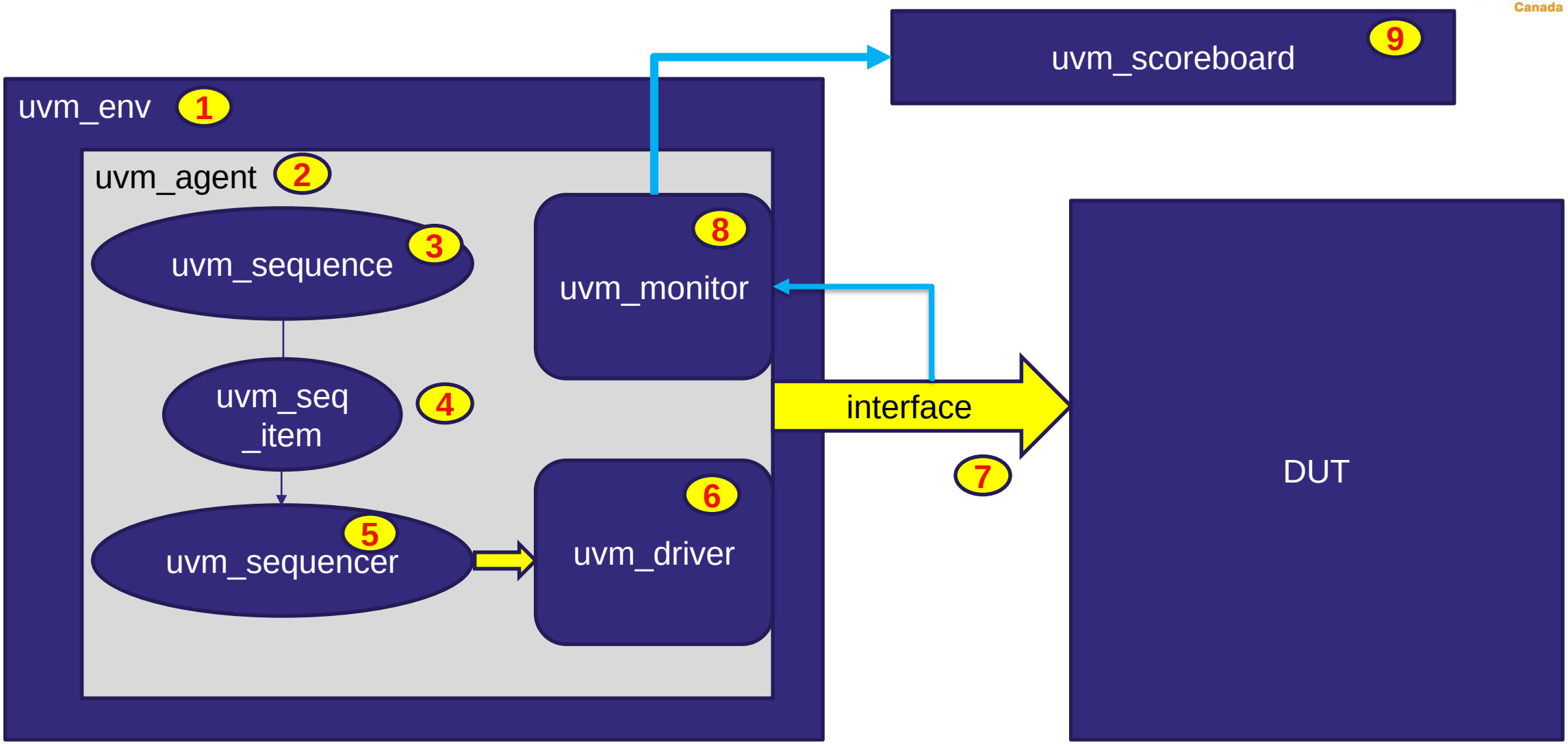
Terrestrial Verification Methodology (TVM)

Goal

Simplify the use-case for an agent's driver, monitor and scoreboard.

… intended for *simple* agents

Usual UVM Sequence Item *Flow*



Sorry…

… my head just exploded

Do we *really* need...

NINE collaborating classes???

Uh…well… *actually*….

YES.

Why? SOLID

S Single Responsibility Princi

O Open/Closed Principle

L Liskov Substitution Principle

I Interface Segregation Principle

D Dependency Inversion Principle

Class should have only a single

Open for extension;

Replaceable with

Many specific interfaces are better than 1 general

Depend on abstractions, not concretions

UVM

Encourages adherence to the SOLID principles.

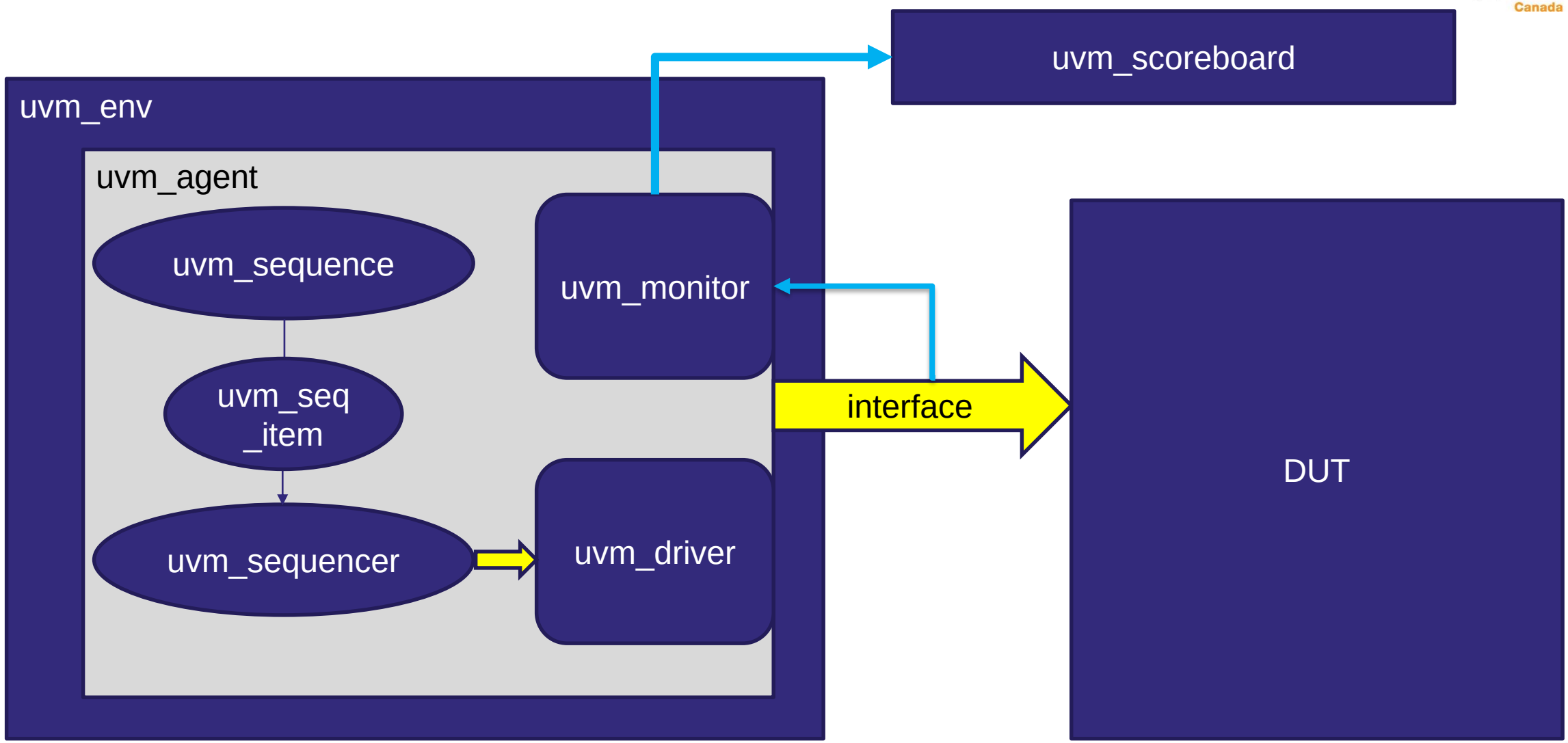
- S each class is responsible for one thing
- O inheritance
- L Factory Pattern
- I API in each is consistent
- D encourages abstraction

But *sometimes* you just want it to be simpler…

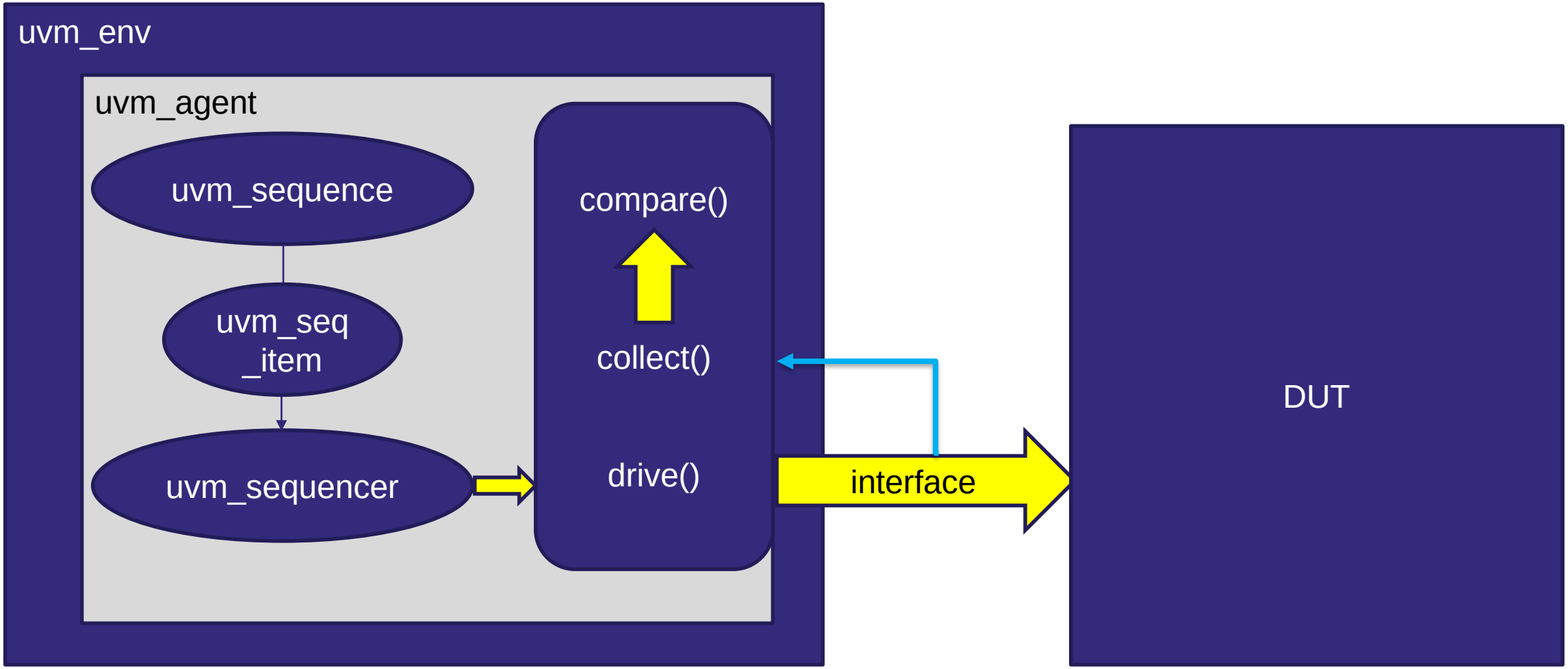
Use cases:

- Simple VIPs that will likely not change: proprietary serial interfaces
- Simple agent for RTL unit testing

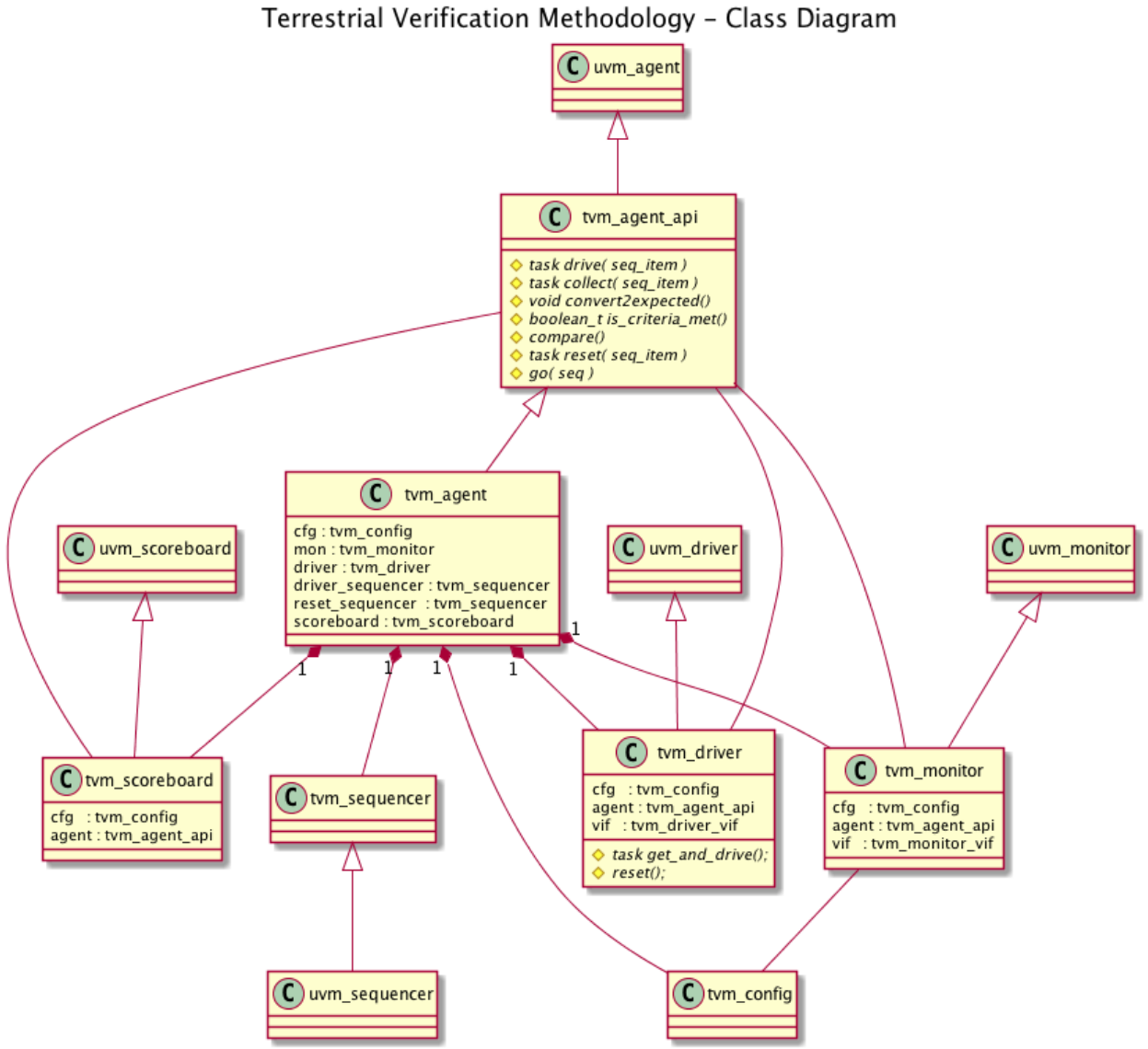
Usual UVM Sequence Item *Flow*



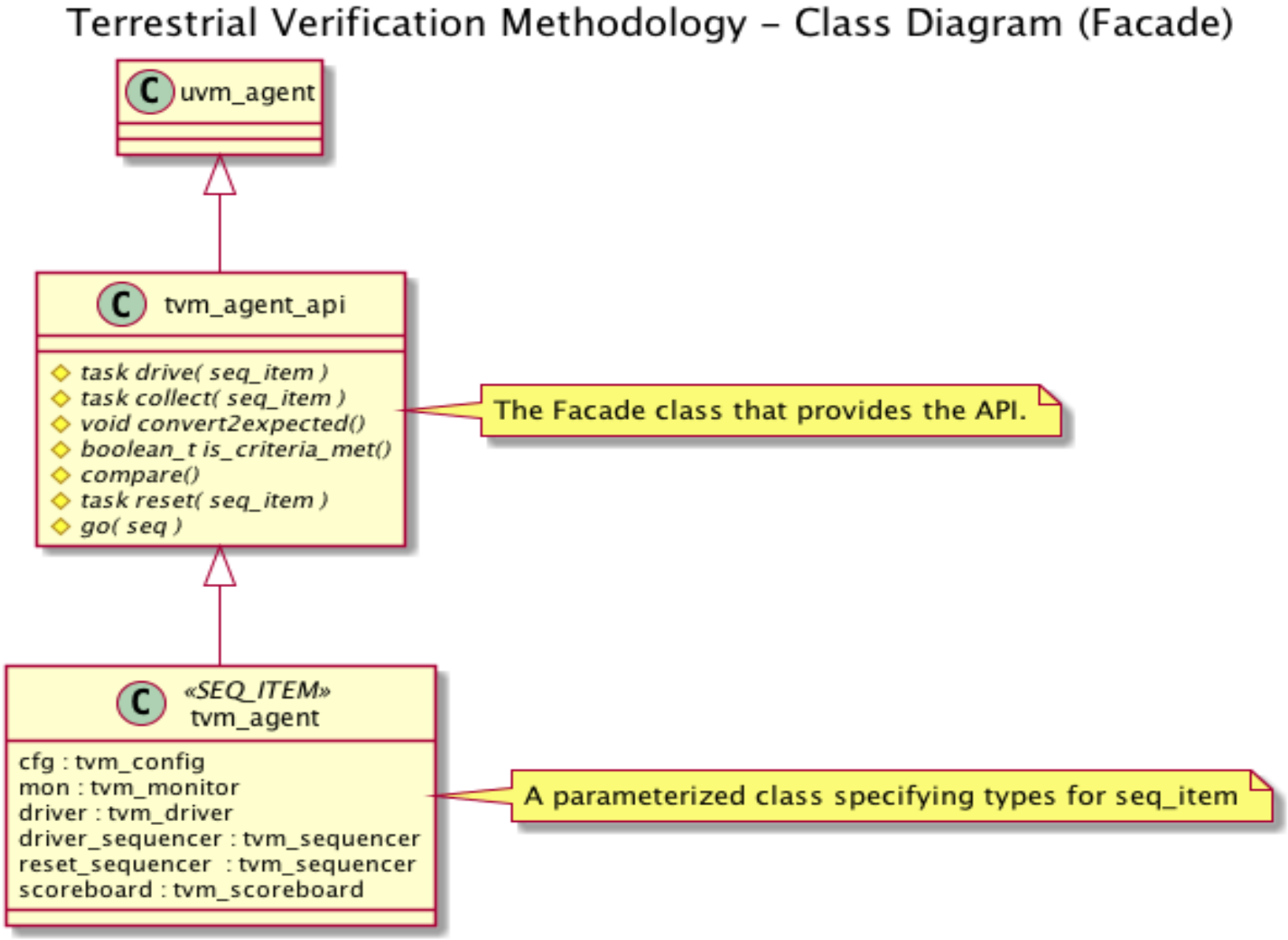
TVM: *Globbing* driving, monitoring & scoreboarding



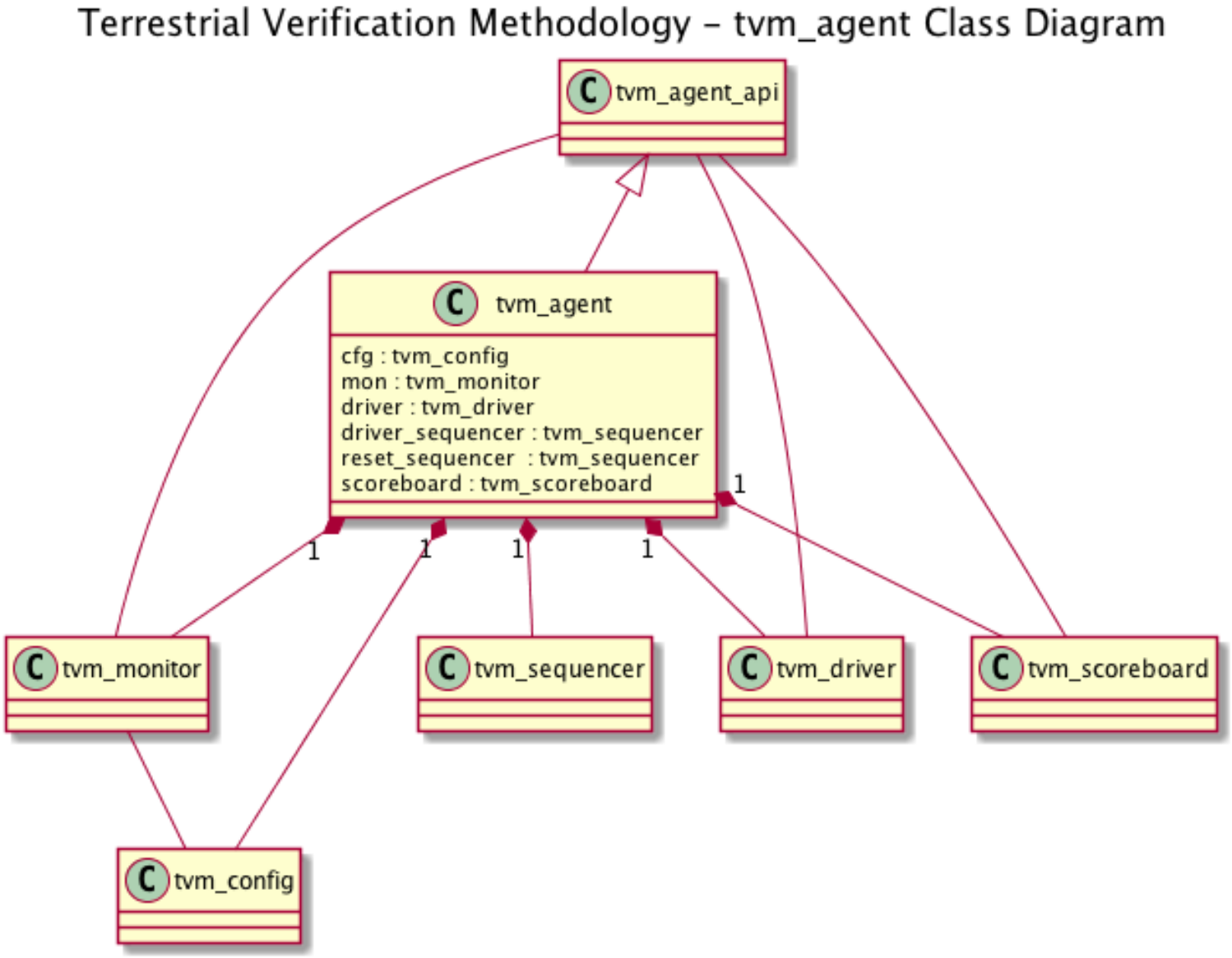
TVM : Entire Class UML Eye-Chart



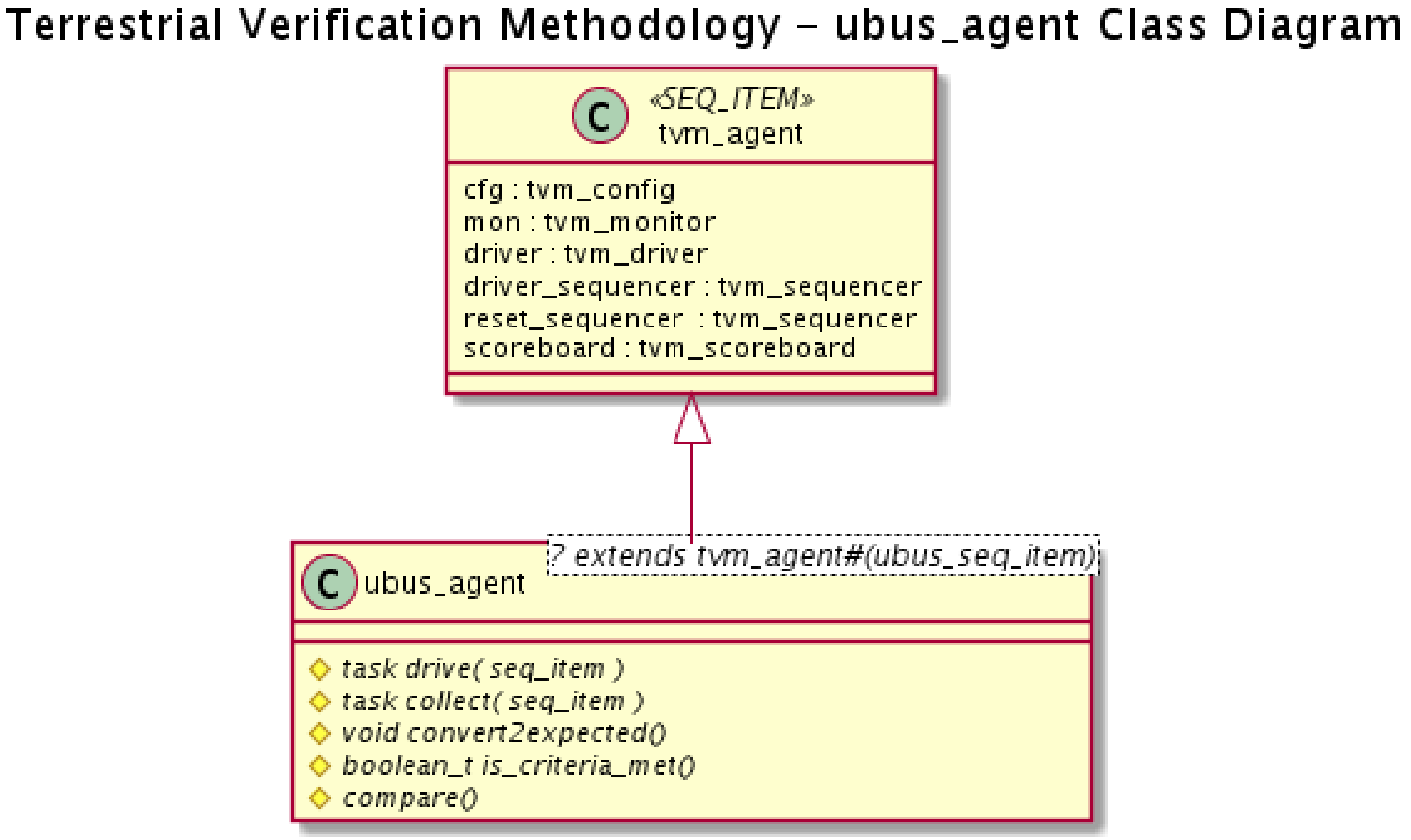
TVM : API – The Heart of It All



TVM Agent: If you build it they will come



TVM Example: ubus_agent



TVM Agent: Build and Connect

- Build phase:
  - Creates the underlying `tvm_driver`, `tvm_monitor`, `tvm_scoreboard`

- Connect phase:
  - Connects sequencer to driver
  - Connects monitor to scoreboard
  - etc.

TVM Agent: Incrementally Evolutionary

Evolutionary: Built on standard UVM classes

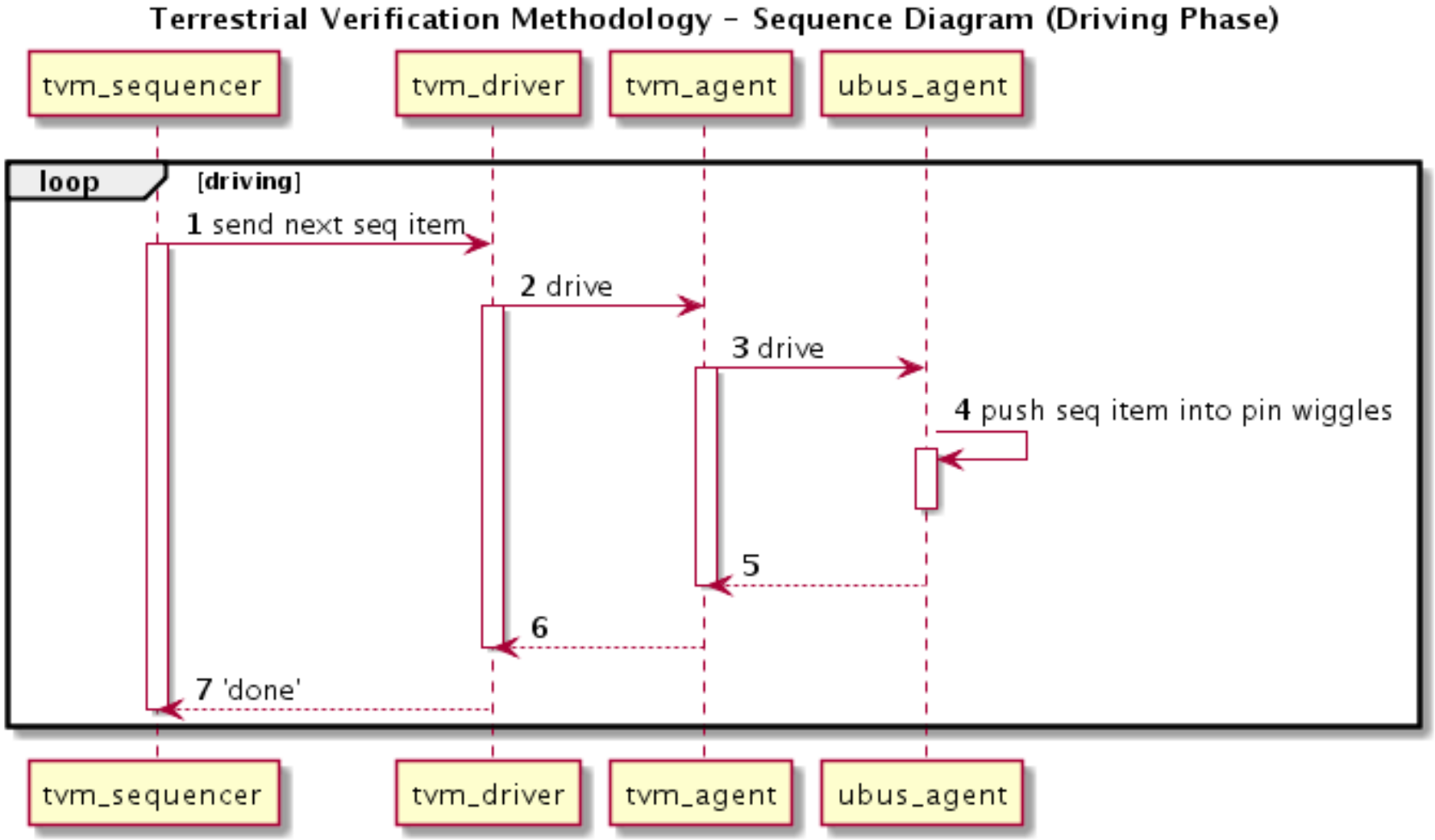
- Can use factory to swap in more complicated driver, monitor, scoreboard
- `tvm_scoreboard` has built-in in-order, HOL comparison
- Can derive `tvm_agent` (or any `tvm_*` class) from your company- or project-layer UVM class.

TVM Agent: Run Phase API

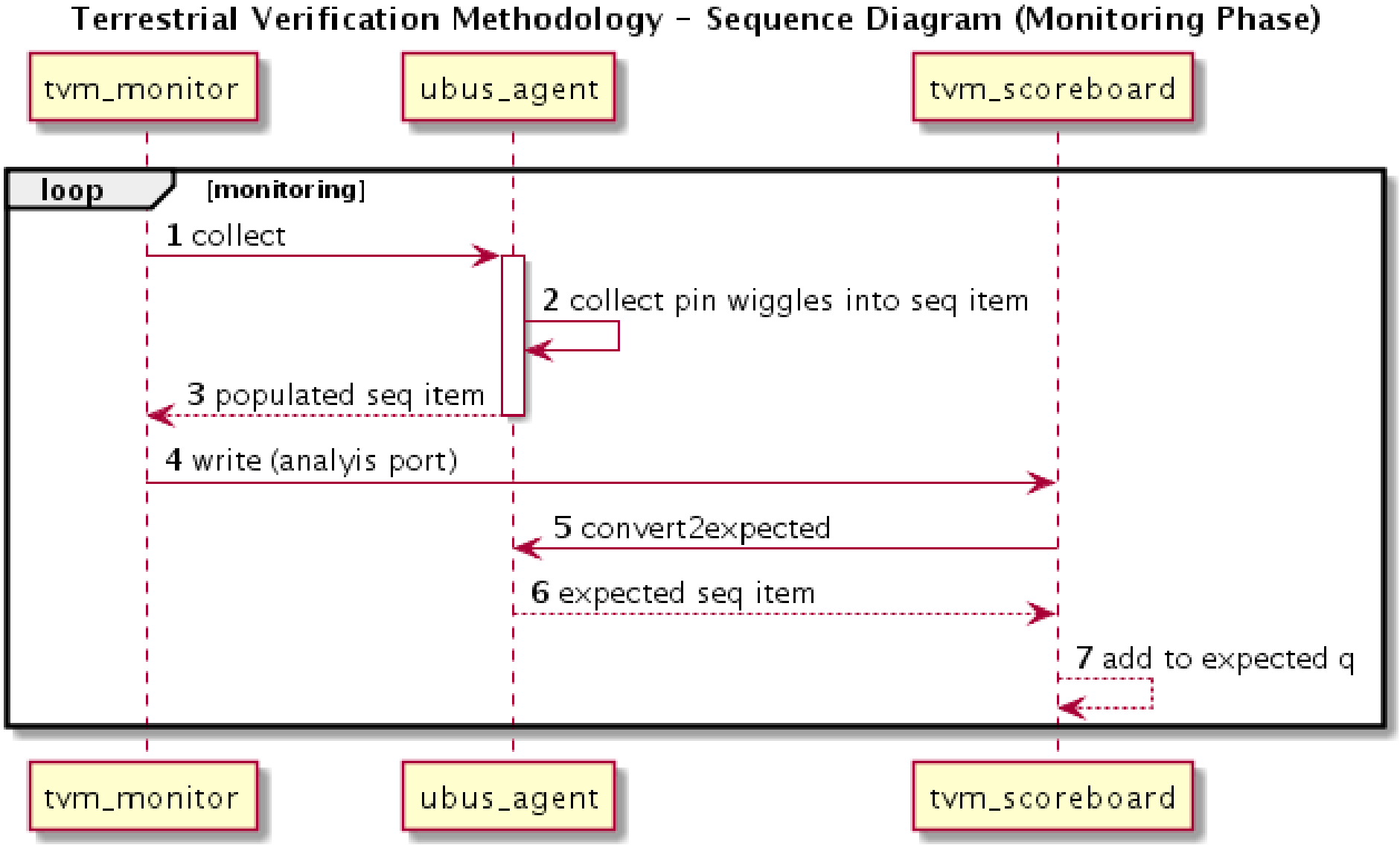
- Driver
- Monitor
- Score board

`drive`	transfers a seq item (from your sequence) to pin wiggles
`collect`	collects pin wiggles and create a seq item, pushes it onto its analysis port
`convert2expected`	aka predictor: accepts input seq item and converts to *expected* seq item
`is_criteria_met`	when *actual* seq item received, do you have everything you need to run the compare? e.g,. Is there something in the *expected* q?
`compare`	does *actual* == *expected* (override if necessary)

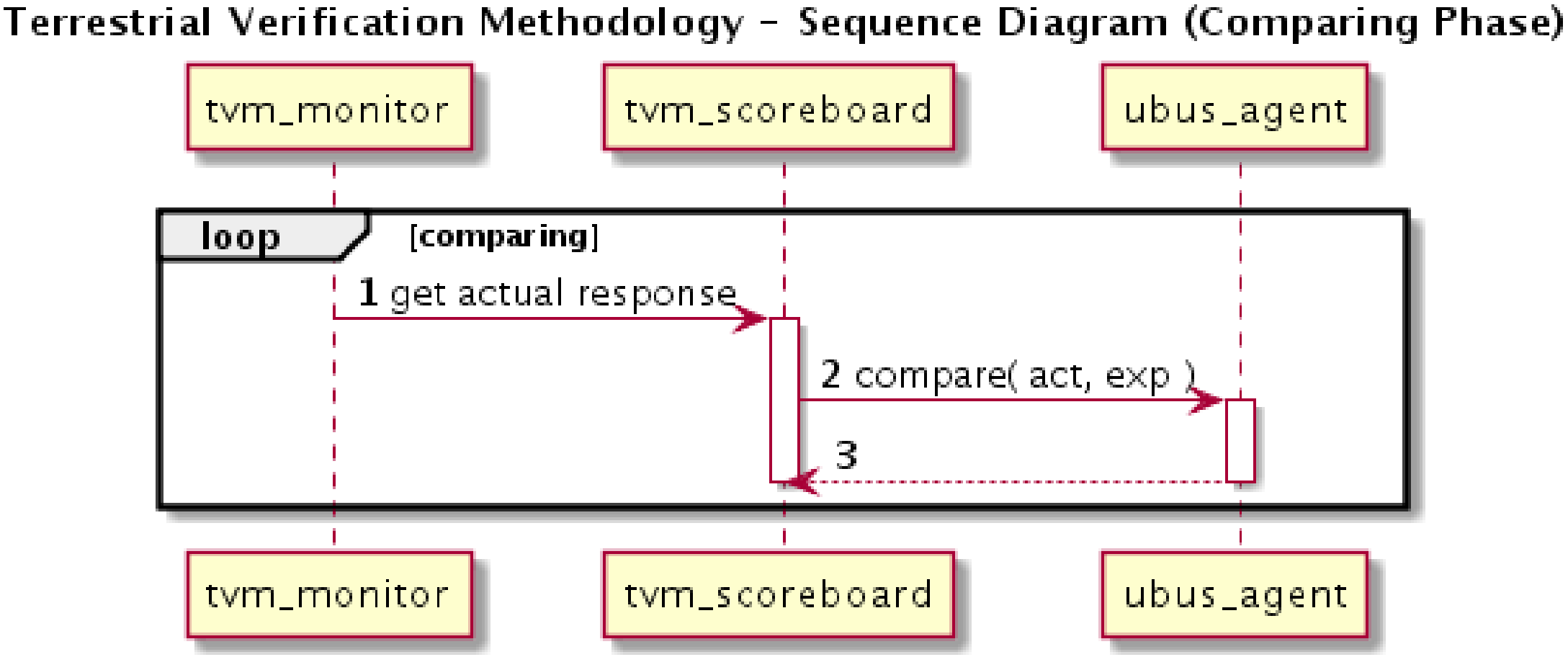
TVM : Driving


Terrestrial Verification Methodology – Sequence Diagram (Driving Phase)
tvm_sequencer
tvm_driver
tvm_agent
ubus_agent
loop
[driving]
1 send next seq item
2 drive
3 drive
4 push seq item into pin wiggles
5
6
7 'done'
tvm_sequencer
tvm_driver
tvm_agent
ubus_agent

TVM : Monitoring



TVM : Scoreboarding



Compare & Contrast: TVM vs UVM

UVM	TVM
Complicated (nine classes)	Less complicated (down to six…and counting)

Compare & Contrast: TVM vs UVM

UVM	TVM
Complicated (nine classes)	Less complicated (down to six…and counting)
Good adherence to SOLID principles	Missing the S and I – OLD??

Compare & Contrast: TVM vs UVM

UVM	TVM
Complicated (nine classes)	Less complicated (down to six…and counting)
Good adherence to SOLID principles	Missing the S and I – OLD??
Can use factory override	Ditto

Compare & Contrast: TVM vs UVM

UVM	TVM
Complicated (nine classes)	Less complicated (down to six…and counting)
Good adherence to SOLID principles	Missing the S and I – OLD??
Can use factory override	Ditto
Need to understand all the UVM plumbing/ un-interesting boilerplate code.	The ‘boilerplate’ code is taken care of (connecting sequencer to driver; monitor to scoreboard)

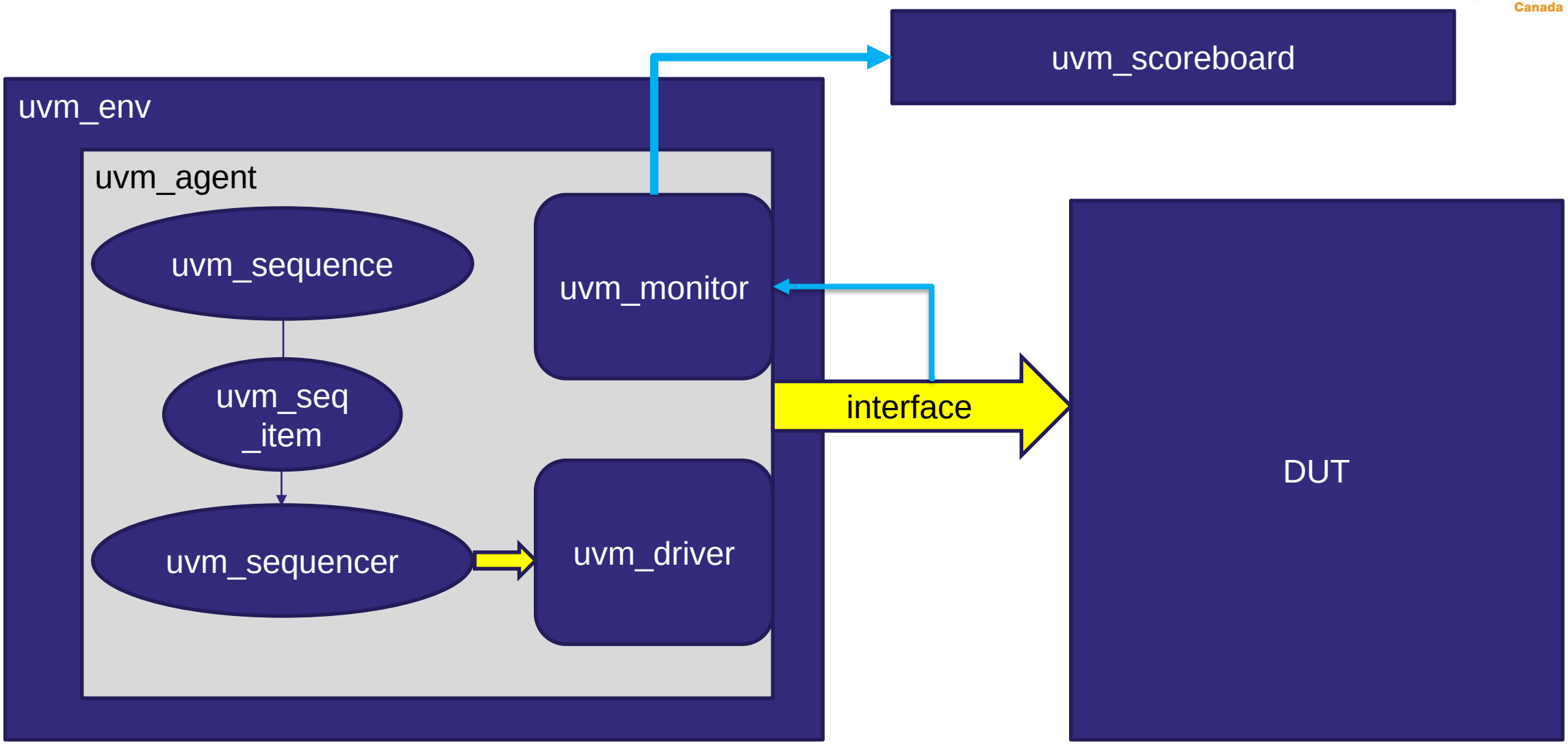
Compare & Contrast: TVM vs UVM

UVM	TVM
Complicated (nine classes)	Less complicated (down to six…and counting)
Good adherence to SOLID principles	Missing the S and I – OLD??
Can use factory override	Ditto
Need to understand all the UVM plumbing/ un-interesting boilerplate code.	The ‘boilerplate’ code is taken care of (connecting sequencer to driver; monitor to scoreboard)
Can be created from a company / project UVM layer	Yup. Ditto

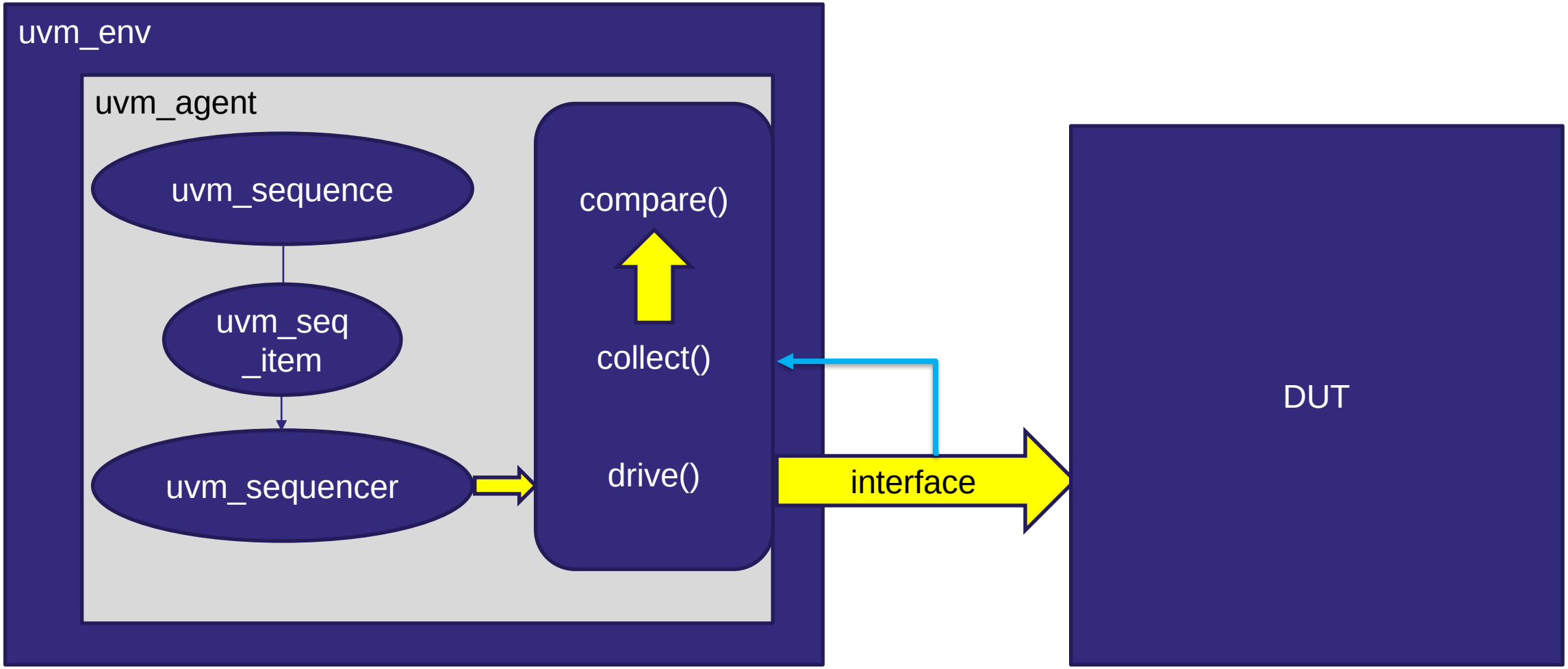
Compare & Contrast: TVM vs UVM

UVM	TVM
Complicated (nine classes)	Less complicated (down to six…and counting)
Good adherence to SOLID principles	Missing the S and I – OLD??
Can use factory override	Ditto
Need to understand all the UVM plumbing/ un-interesting boilerplate code.	The ‘boilerplate’ code is taken care of (connecting sequencer to driver; monitor to scoreboard)
Can be created from a company / project UVM layer	Yup. Ditto
Scale from Block to System.	It should… but if you probably want to create a complete UVM VIP at this point

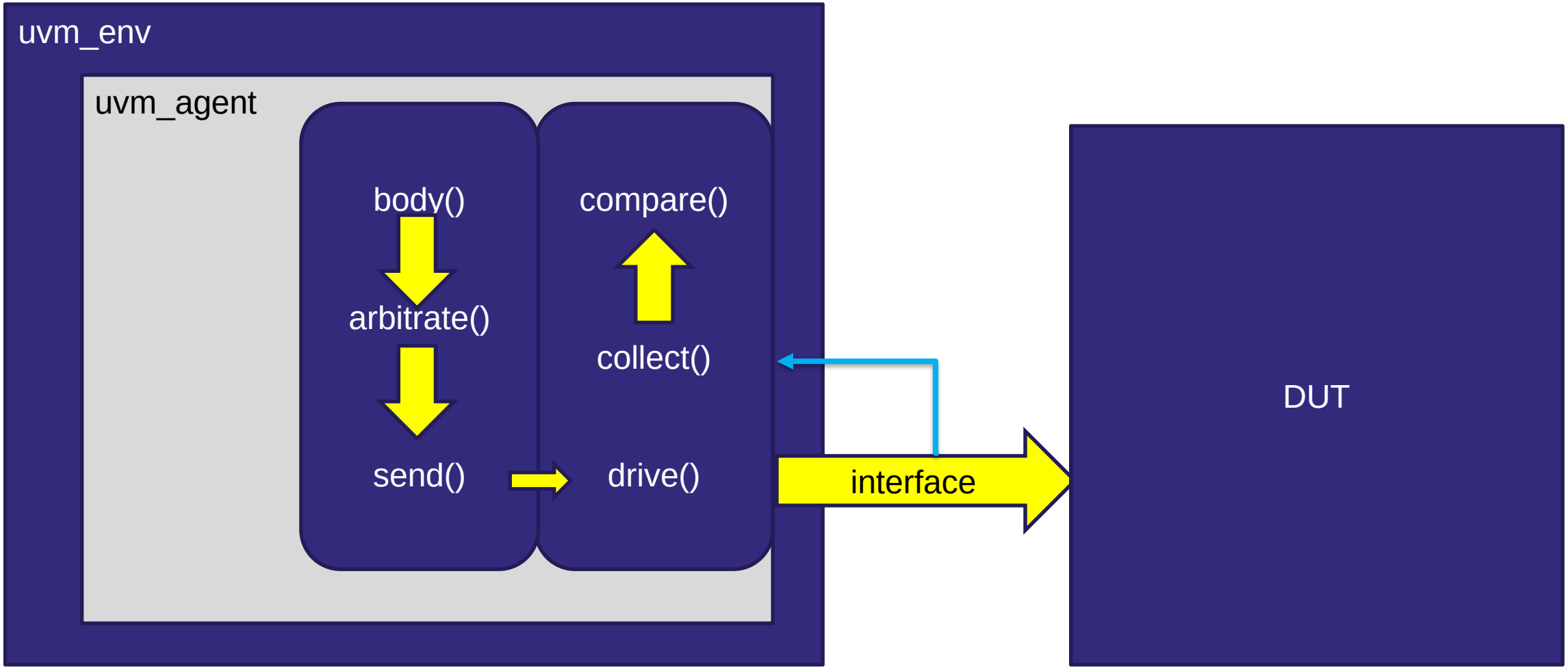
UVM *Now*


uvm_scoreboard
uvm_env
uvm_agent
uvm_sequence
uvm_seq
_item
uvm_sequencer
uvm_monitor
uvm_driver
interface
DUT

TVM *Now*


uvm_env
uvm_agent
uvm_sequence
uvm_seq
_item
uvm_sequencer
compare()
collect()
drive()
interface
DUT

TVM Future? *Globbing* seq_item gen



Conclusions

- Goal of the Presentation: describe Façade pattern
- Demonstrate Façade use-case with TVM
- TVM is at this point a “neat idea” (IMHO) that is still evolving

Thank You

.. and thanks to
Pierre Girodias and
Mike Thompson
for their help.

Q&A

We have time for nine questions.