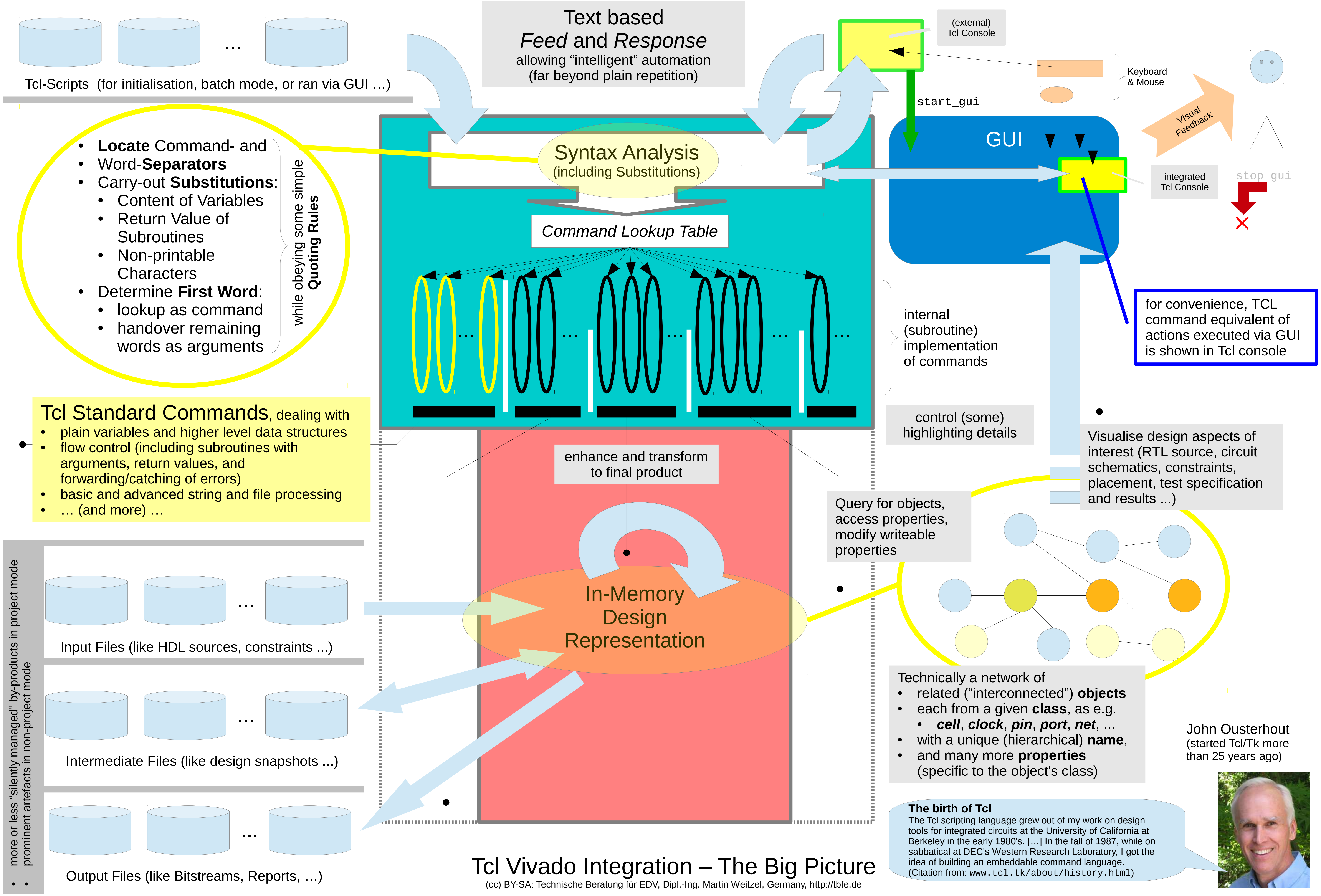


Tcl-Scripts (for initialisation, batch mode, or ran via GUI …)

Text based
Feed and *Response*
allowing “intelligent” automation
(far beyond plain repetition)

Syntax Analysis
(including Substitutions)

- **Locate** Command- and
- Word-**Separators**
- Carry-out **Substitutions**:
  - Content of Variables
  - Return Value of Subroutines
  - Non-printable Characters
- Determine **First Word**:
  - lookup as command
  - handover remaining words as arguments

while obeying some simple **Quoting Rules**

internal (subroutine) implementation of commands

for convenience, TCL command equivalent of actions executed via GUI is shown in Tcl console

Tcl Standard Commands, dealing with

- plain variables and higher level data structures
- flow control (including subroutines with arguments, return values, and forwarding/catching of errors)
- basic and advanced string and file processing
- … (and more) …

control (some) highlighting details

enhance and transform to final product

Visualise design aspects of interest (RTL source, circuit schematics, constraints, placement, test specification and results ...)

Query for objects, access properties, modify writeable properties

Input Files (like HDL sources, constraints ...)

Intermediate Files (like design snapshots ...)

Output Files (like Bitstreams, Reports, …)

more or less “silently managed” by-products in project mode
prominent artefacts in non-project mode

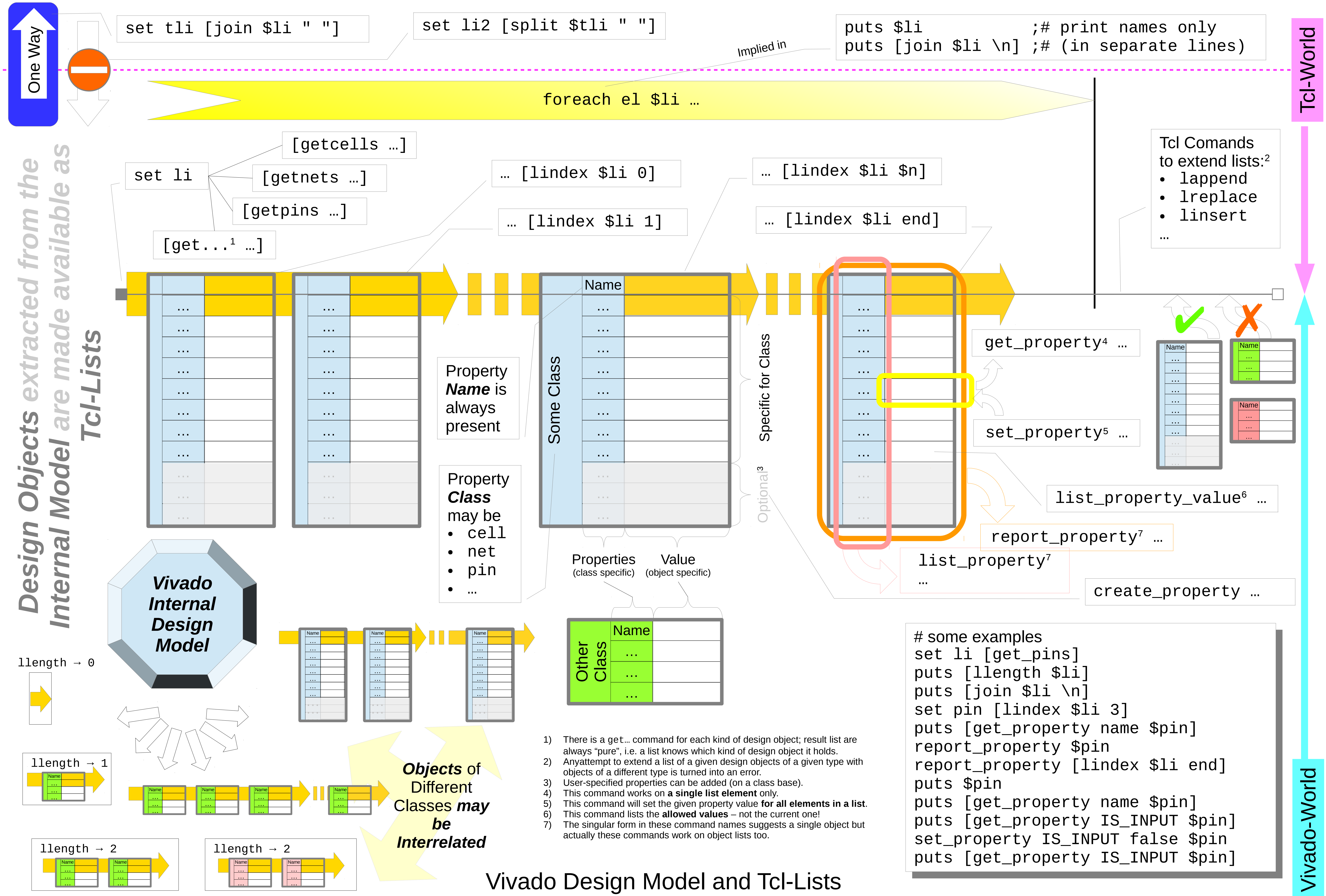
Technically a network of

- related (“interconnected”) **objects**
- each from a given **class**, as e.g.
  - ***cell***, ***clock***, ***pin***, ***port***, ***net***, ...
- with a unique (hierarchical) **name**,
- and many more **properties** (specific to the object's class)

John Ousterhout
(started Tcl/Tk more than 25 years ago)

The birth of Tcl
The Tcl scripting language grew out of my work on design tools for integrated circuits at the University of California at Berkeley in the early 1980's. […] In the fall of 1987, while on sabbatical at DEC's Western Research Laboratory, I got the idea of building an embeddable command language.
(Citation from: `www.tcl.tk/about/history.html`)

Tcl Vivado Integration – The Big Picture

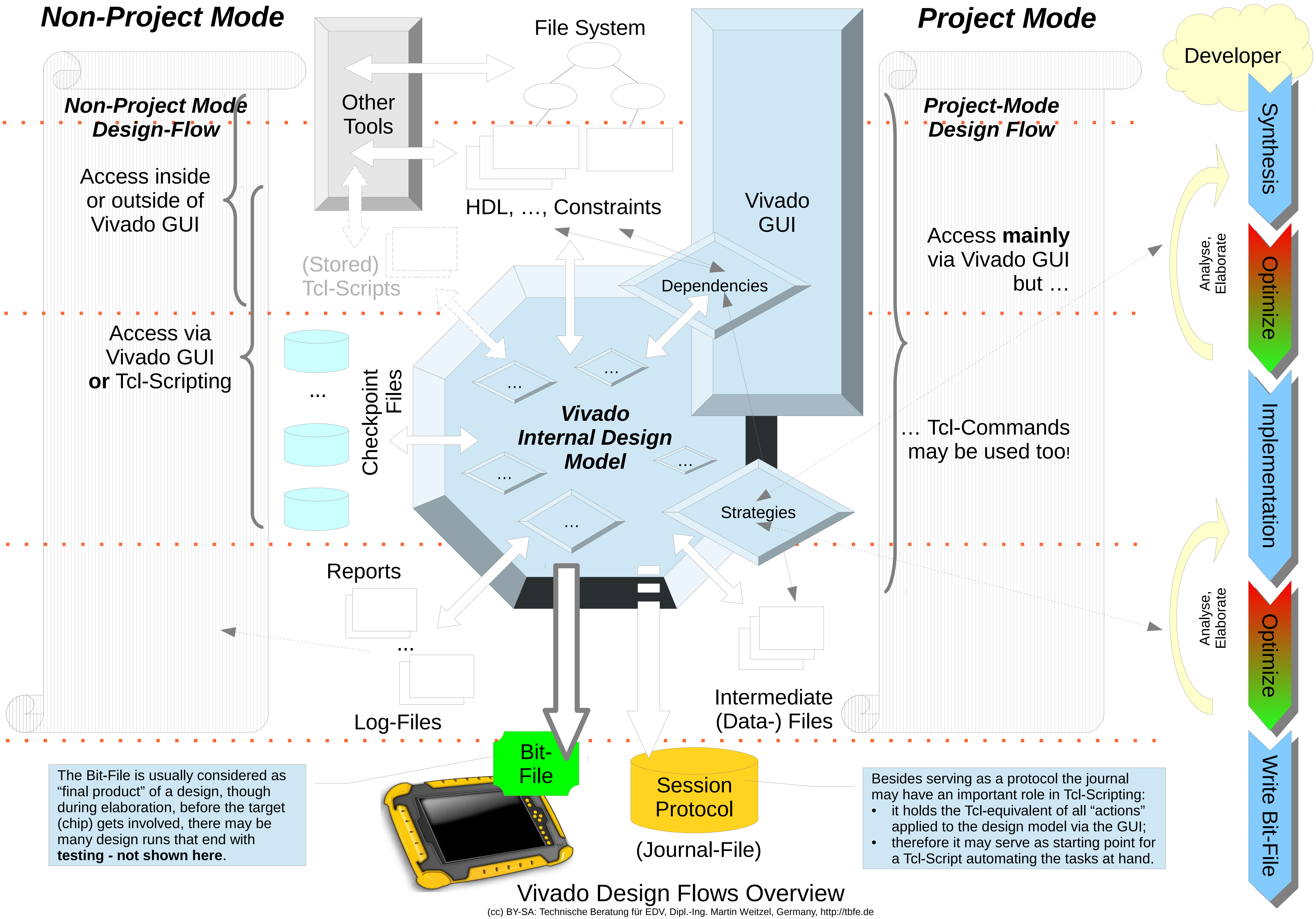

One Way
Tcl-World
Vivado-World
Implied in
Design Objects extracted from the Internal Model are made available as Tcl-Lists
Property Name is always present
Property Class may be
• cell
• net
• pin
• …
Some Class
Name
Specific for Class
Optional3
Properties (class specific)
Value (object specific)
Other Class
Vivado Internal Design Model
Objects of Different Classes may be Interrelated
llength → 0
llength → 1
llength → 2
llength → 2
Tcl Comands to extend lists:2
• lappend
• lreplace
• linsert
…

```
set tli [join $li " "]
set li2 [split $tli " "]
puts $li                ;# print names only
puts [join $li \n] ;# (in separate lines)
foreach el $li …
set li
[getcells …]
[getnets …]
[getpins …]
[get...1 …]
… [lindex $li 0]
… [lindex $li 1]
… [lindex $li $n]
… [lindex $li end]
get_property4 …
set_property5 …
list_property_value6 …
report_property7 …
list_property7
…
create_property …
```

```
# some examples
set li [get_pins]
puts [llength $li]
puts [join $li \n]
set pin [lindex $li 3]
puts [get_property name $pin]
report_property $pin
report_property [lindex $li end]
puts $pin
puts [get_property name $pin]
puts [get_property IS_INPUT $pin]
set_property IS_INPUT false $pin
puts [get_property IS_INPUT $pin]
```

[1] There is a `get…` command for each kind of design object; result list are always “pure”, i.e. a list knows which kind of design object it holds.
[2] Anyattempt to extend a list of a given design objects of a given type with objects of a different type is turned into an error.
[3] User-specified properties can be added (on a class base).
[4] This command works on **a single list element** only.
[5] This command will set the given property value **for all elements in a list**.
[6] This command lists the **allowed values** – not the current one!
[7] The singular form in these command names suggests a single object but actually these commands work on object lists too.

Vivado Design Model and Tcl-Lists



Non-Project Mode

Non-Project Mode Design-Flow

Access inside or outside of Vivado GUI

Access via Vivado GUI **or** Tcl-Scripting

Project Mode

Project-Mode Design Flow

Access **mainly** via Vivado GUI but …

… Tcl-Commands may be used too!

Developer

Other Tools

File System

HDL, …, Constraints

(Stored) Tcl-Scripts

Vivado GUI

Dependencies

Checkpoint Files

…

Vivado Internal Design Model

Strategies

Reports

Log-Files

Intermediate (Data-) Files

Bit-File

Session Protocol

(Journal-File)

Synthesis

Analyse, Elaborate

Optimize

Implementation

Analyse, Elaborate

Optimize

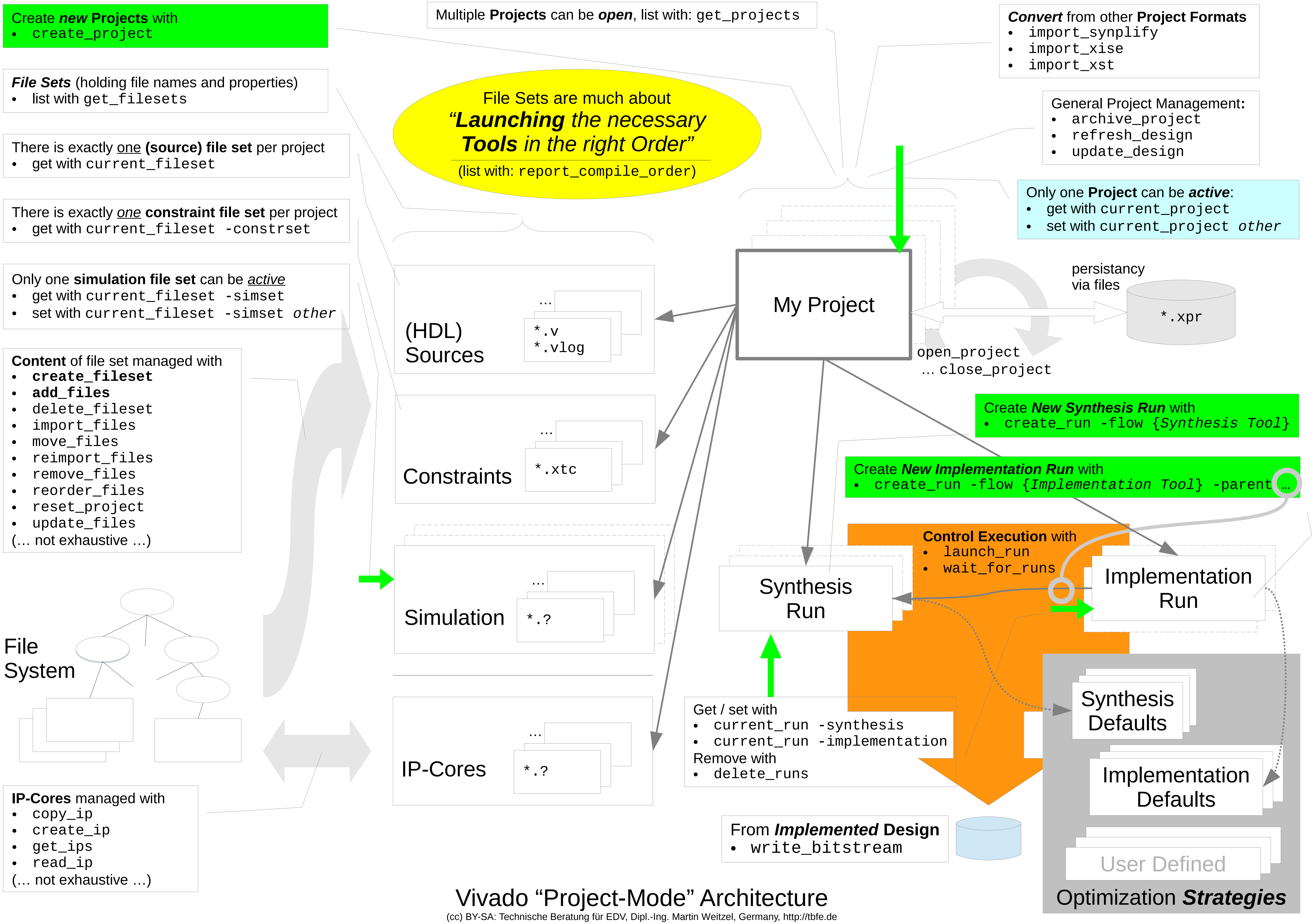
Write Bit-File

The Bit-File is usually considered as “final product” of a design, though during elaboration, before the target (chip) gets involved, there may be many design runs that end with **testing - not shown here**.

Besides serving as a protocol the journal may have an important role in Tcl-Scripting:

- it holds the Tcl-equivalent of all “actions” applied to the design model via the GUI;
- therefore it may serve as starting point for a Tcl-Script automating the tasks at hand.

Vivado Design Flows Overview



Create ***new* Projects** with
- `create_project`

File Sets (holding file names and properties)
- list with `get_filesets`

There is exactly one **(source) file set** per project
- get with `current_fileset`

There is exactly *one* **constraint file set** per project
- get with `current_fileset -constrset`

Only one **simulation file set** can be *active*
- get with `current_fileset -simset`
- set with `current_fileset -simset other`

Content of file set managed with
- **`create_fileset`**
- **`add_files`**
- `delete_fileset`
- `import_files`
- `move_files`
- `reimport_files`
- `remove_files`
- `reorder_files`
- `reset_project`
- `update_files`

(… not exhaustive …)

IP-Cores managed with
- `copy_ip`
- `create_ip`
- `get_ips`
- `read_ip`

(… not exhaustive …)

Multiple **Projects** can be ***open***, list with: `get_projects`

File Sets are much about “***Launching*** the necessary ***Tools*** in the right Order”

(list with: `report_compile_order`)

Convert from other **Project Formats**
- `import_synplify`
- `import_xise`
- `import_xst`

General Project Management:
- `archive_project`
- `refresh_design`
- `update_design`

Only one **Project** can be ***active***:
- get with `current_project`
- set with `current_project other`

persistancy via files

`open_project`

… `close_project`

Create ***New Synthesis Run*** with
- `create_run -flow {Synthesis Tool}`

Create ***New Implementation Run*** with
- `create_run -flow {Implementation Tool} -parent …`

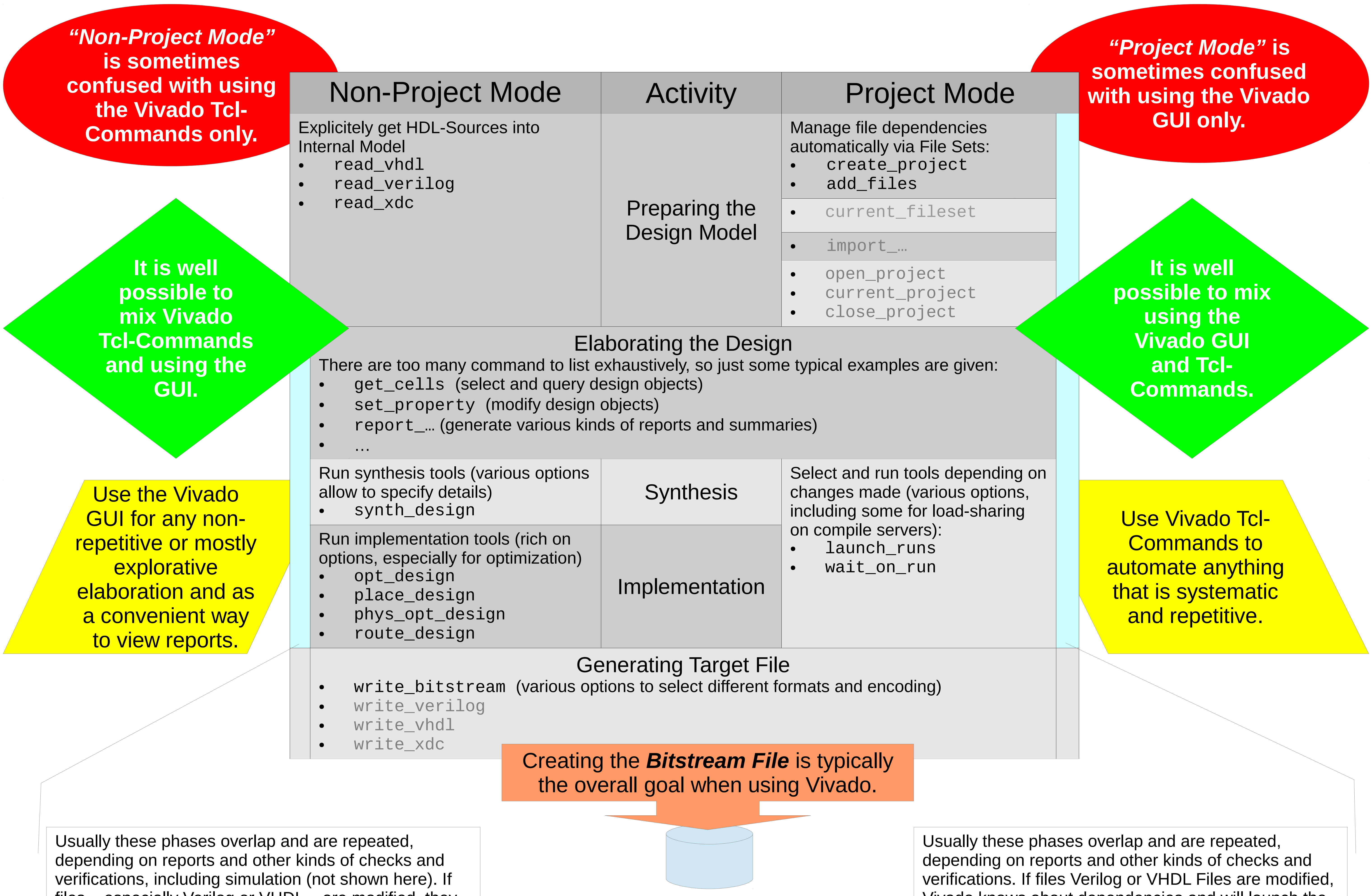
Control Execution with
- `launch_run`
- `wait_for_runs`

Get / set with
- `current_run -synthesis`
- `current_run -implementation`

Remove with
- `delete_runs`

From ***Implemented* Design**
- `write_bitstream`

Vivado “Project-Mode” Architecture



“*Non-Project Mode*” is sometimes confused with using the Vivado Tcl-Commands only.

“*Project Mode*” is sometimes confused with using the Vivado GUI only.

Non-Project Mode	Activity	Project Mode
Explicitely get HDL-Sources into Internal Model • `read_vhdl` • `read_verilog` • `read_xdc`	Preparing the Design Model	Manage file dependencies automatically via File Sets: • `create_project` • `add_files` • `current_fileset` • `import_…` • `open_project` • `current_project` • `close_project`
Elaborating the Design There are too many command to list exhaustively, so just some typical examples are given: • `get_cells` (select and query design objects) • `set_property` (modify design objects) • `report_…` (generate various kinds of reports and summaries) • …		
Run synthesis tools (various options allow to specify details) • `synth_design`	Synthesis	Select and run tools depending on changes made (various options, including some for load-sharing on compile servers): • `launch_runs` • `wait_on_run`
Run implementation tools (rich on options, especially for optimization) • `opt_design` • `place_design` • `phys_opt_design` • `route_design`	Implementation	
Generating Target File • `write_bitstream` (various options to select different formats and encoding) • `write_verilog` • `write_vhdl` • `write_xdc`		

It is well possible to mix Vivado Tcl-Commands and using the GUI.

It is well possible to mix using the Vivado GUI and Tcl-Commands.

Use the Vivado GUI for any non-repetitive or mostly explorative elaboration and as a convenient way to view reports.

Use Vivado Tcl-Commands to automate anything that is systematic and repetitive.

Creating the ***Bitstream File*** is typically the overall goal when using Vivado.

Usually these phases overlap and are repeated, depending on reports and other kinds of checks and verifications, including simulation (not shown here). If files – especially Verilog or VHDL – are modified, they need to be read again explicitely.

Usually these phases overlap and are repeated, depending on reports and other kinds of checks and verifications. If files Verilog or VHDL Files are modified, Vivado knows about dependencies and will launch the tools as required in the right order.

Vivado Modes and Tcl Commands