visualCaseGen

Streamlining CESM Simpler Modeling Efforts and Beyond

Alper Altuntas – NCAR/CGD/OS
Isla Simpson, Scott Bachman, Samuel Levis, Brian Dobbins, Gokhan Danabasoglu

Project funded by an NSF CSSI Award (PIs: Bachman, Simpson)

June 11, 2024

This material is based upon work supported by the NSF National Center for Atmospheric Research, which is a major facility sponsored by the U.S. National Science Foundation under Cooperative Agreement No. 1852977.

Goals:

- Streamline CESM simpler modeling efforts *and beyond*.
- Enable hierarchical modeling: explore/combine different complexity levels.

In Practical Terms:

- Browse existing CESM configurations efficiently.
- Quickly generate new configurations (compsets and grids):
  - Mix and match models and settings in a compatible manner.
  - Create or modify grids as needed.

Part 1: A quick tour of visualCaseGen:

A GUI that guides the user through the process of creating experiments.

```
from visualCaseGen import gui; gui
```

visualCaseGen

? Help

Welcome to visualCaseGen!

visualCaseGen guides users through the process of creating CESM experiments.

Start

Click to add a cell.

Part 2: A Glimpse Inside visualCaseGen Software

visualCaseGen Constraint Solver

How Does visualCaseGen Determine Compatibility?

- **Constraint Specification:** Defines relationships between config variables.
- **Constraint Solver:** Utilizes the Z3 library for logical reasoning.

The Z3 Library

What it Does: Checks if a set of constraints can be satisfied. Finds solutions.

Strengths: Combines logical reasoning with knowledge about specific domains (bools, ints, reals, strings, etc.). Manages complex relationships.

Development and Usage:

- Widely used in academia and industry. (~10k Citations, ~10k GitHub Stars)
- Developed by Microsoft Research. Open source and free (MIT license).
- Robust Python API. Available via pip.

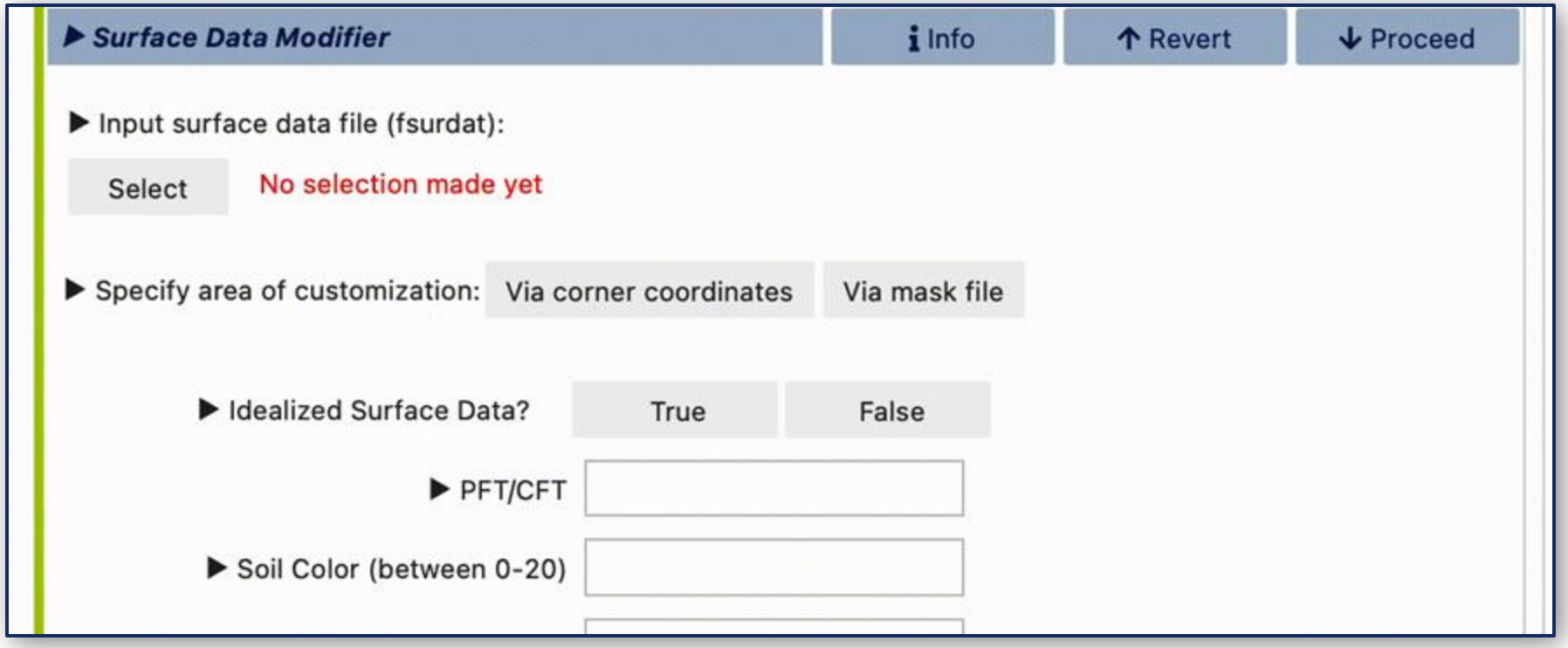
Relational Constraint Specification in visualCaseGen

visualCaseGen Constraint Solver

```
LND_DOM_PFT >= 0.0:
    "PFT/CFT must be set to a nonnegative number"
```

visualCaseGen Constraint Solver

```
LND_DOM_PFT >= 0.0:
    "PFT/CFT must be set to a nonnegative number"
```

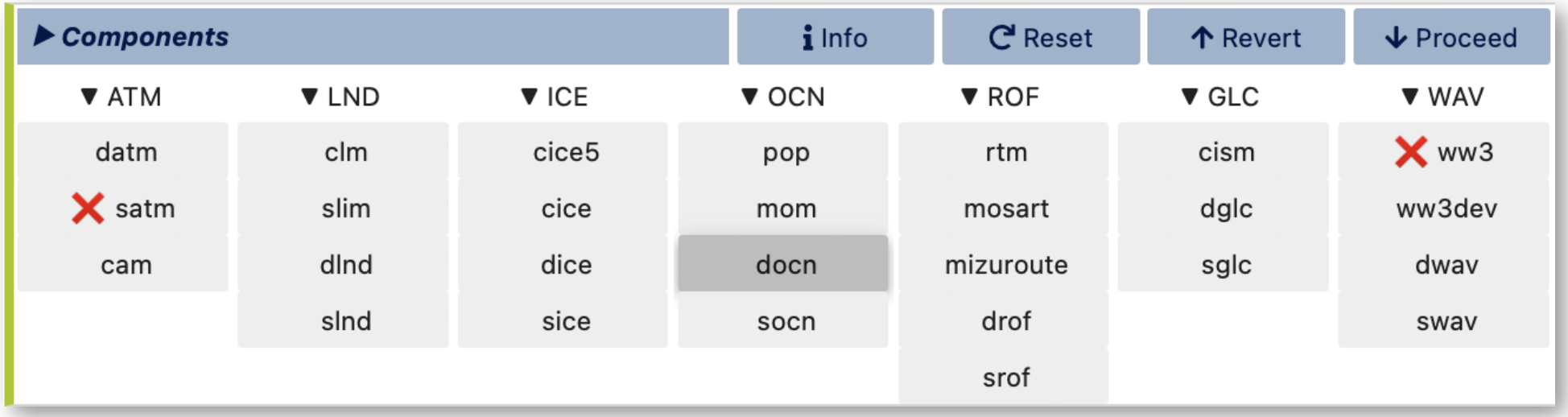


visualCaseGen Constraint Solver

```
Or(COMP_OCN=="mom", OCN_GRID_MODE=="Standard"):
    "Custom OCN grids can only be generated for MOM6."
```

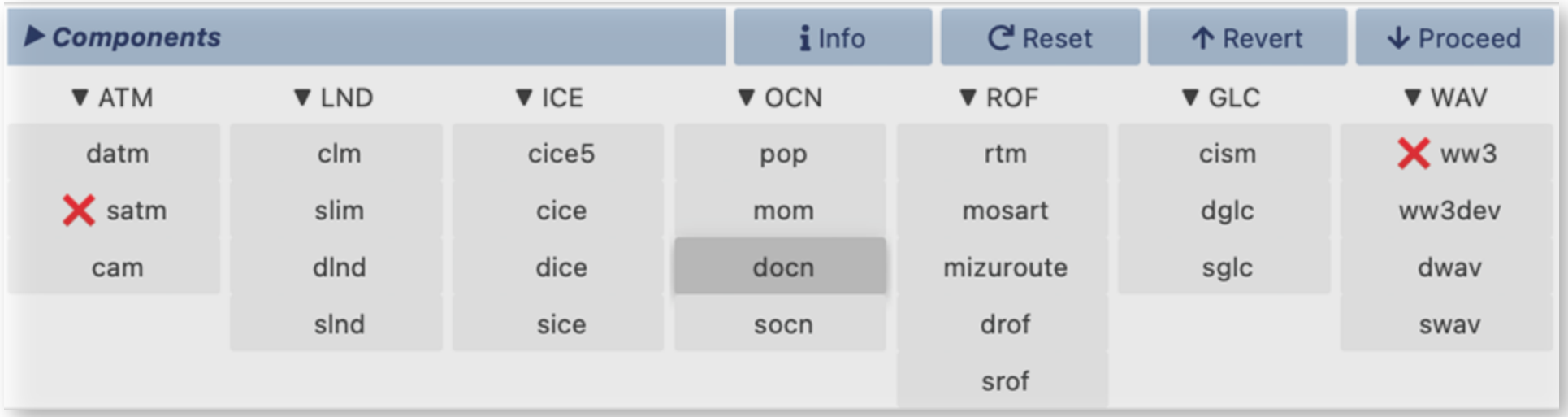
visualCaseGen Constraint Solver

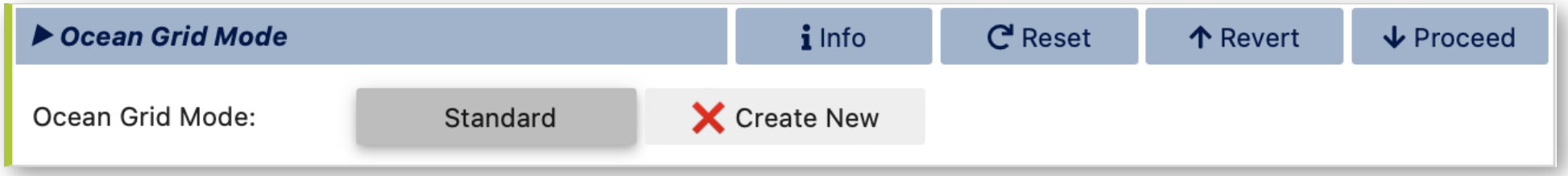
```
Or(COMP_OCN=="mom", OCN_GRID_MODE=="Standard"):
    "Custom OCN grids can only be generated for MOM6."
```



visualCaseGen Constraint Solver

```
Or(COMP_OCN=="mom", OCN_GRID_MODE=="Standard"):
    "Custom OCN grids can only be generated for MOM6."
```





visualCaseGen Constraint Solver

```
Implies(And(COMP_OCN=="mom", COMP_LND=="slnd", COMP_ICE=="sice"), OCN_LENY<180.0):
    "If LND and ICE are stub, custom MOM6 grid must exclude poles (singularity).",
```

visualCaseGen Constraint Solver

The constraint specification syntax might seem unfamiliar at first.

visualCaseGen Constraint Solver

	Python	Z3 Constraints
Comparison	`==, !=, >, <, >=, <=`	*same*
Arithmetic	`+, -, *, /`	*same*

Logical	`p and q`
	`p or q`
	`not p`
	`not p or q`
		`And(p, q)`
		`Or(p, q)`
		`Not(p)`
		`Implies(p, q)`

String	`a in b`
	`a.startswith(b)`
	...
		`Contains(b, a)`
		`PrefixOf(a,b)`
		...

visualCaseGen Constraint Solver

	Python	Z3 Constraints
Paradigm:	`Imperative`	`Declarative`

visualCaseGen Constraint Solver

	Python	Z3 Constraints
Paradigm:	`Imperative`	`Declarative`

1996). Building configurators using standard imperative programming is often a nightmare (Barker and O'Connor 1989; Piller et al. 2014) because (1) many functionalities need to be provided, (2) they are complex to implement, and (3) the resulting programs turn out to be extremely difficult to maintain when existing constraints of the application do-

A more natural and efficient fit for constraint specification.

Carbonnelle, Pierre, et al. "Interactive configurator with FO (.) and IDP-Z3." (2022).

visualCaseGen Constraint Solver

The constraint specification syntax might seem unfamiliar at first, ***but*** it enables the use of z3 as the core of the visualCaseGen constraint solver.

Why use a solver?

Constraint satisfaction problem (CSP) is inherently complex (NP-complete).

- **Hidden Conflicts**: Individual constraints might be satisfied, but their combinations can lead to conflicts.
- **Dead-Ends:** Solvers prevent scenarios where no feasible options remain for configuration variables.
- **Constraint Analysis**: Are the constraints satisfiable? Any unreachable options? Any constraint redundant?
- **Scalability and Efficiency:** As variables and constraints increase, complexity grows exponentially. CSP solvers tackle this efficiently.

visualCaseGen Constraint Solver

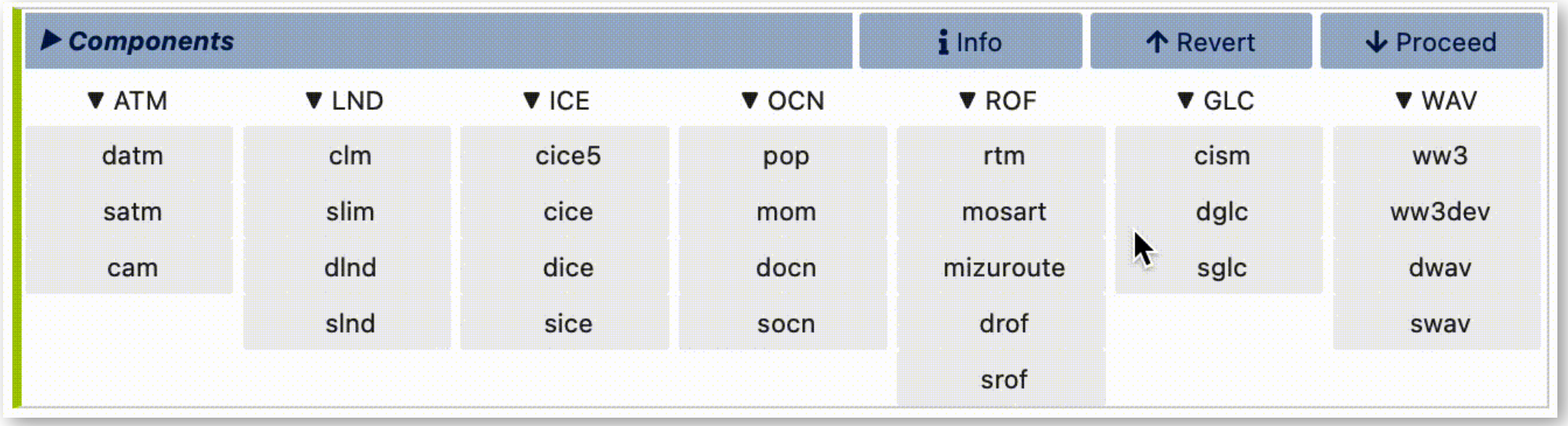
```
Implies(COMP_WAV=="ww3", In(COMP_OCN, ["mom", "pop"])):
    "WW3 can only be selected if either POP2 or MOM6 is the ocean component.",

Implies(COMP_ATM=="satm", COMP_OCN=="socn"):
    "An active/data atmosphere is needed to force the ocean model."
```

visualCaseGen Constraint Solver

```
Implies(COMP_WAV=="ww3", In(COMP_OCN, ["mom", "pop"])):
    "WW3 can only be selected if either POP2 or MOM6 is the ocean component.",

Implies(COMP_ATM=="satm", COMP_OCN=="socn"):
    "An active/data atmosphere is needed to force the ocean model."
```



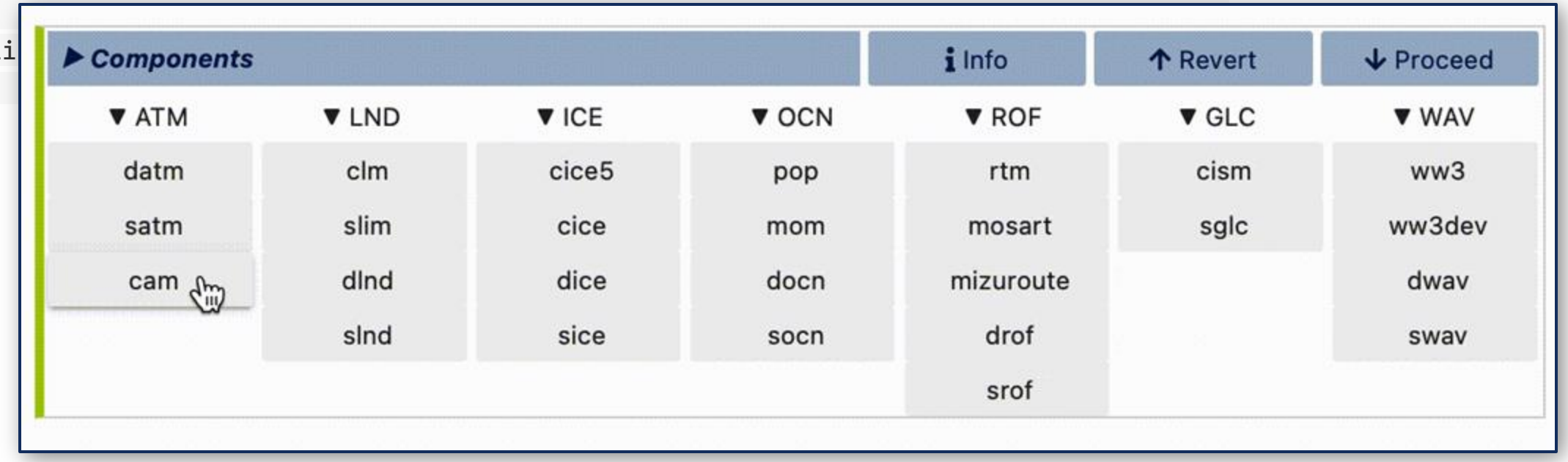
visualCaseGen Constraint Solver

```
Implies(COMP_LND=="clm", COMP_ROF!="drof") :
    "CLM cannot be coupled with a data runoff model.",
Implies(COMP_LND=="slim", And(COMP_GLC=="sglc", COMP_ROF=="srof", COMP_WAV=="swav")) :
    "GLC, ROF, and WAV cannot be coupled with SLIM.",
Implies(COMP_OCN=="mom", COMP_WAV!="dwav") :
    "MOM6 cannot be coupled with data wave component.",
Implies(COMP_LND=="slnd", Or(COMP_OCN=="mom", COMP_GLC=="sglc")) :
     "GLC cannot be coupled with a stub land model, unless it is coupled with MOM6.",
Implies(COMP_LND=="dlnd", COMP_ATM!="cam"):
     "CAM-DLND coupling is not supported.",
```

visualCaseGen Constraint Solver

```
Implies(COMP_LND=="clm", COMP_ROF!="drof") :
    "CLM cannot be coupled with a data runoff model.",
Implies(COMP_LND=="slim", And(COMP_GLC=="sglc", COMP_ROF=="srof", COMP_WAV=="swav")) :
    "GLC, ROF, and WAV cannot be coupled with SLIM.",
Implies(COMP_OCN=="mom", COMP_WAV!="dwav") :
    "MOM6 cannot be coupled with data wave component.",
Implies(COMP_LND=="slnd", Or(COMP_OCN=="mom", COMP_GLC=="sglc")) :

Impli
```



***The bottom line*,**

The interaction of constraints, even simple ones, can lead to hidden conflicts, dead ends, and chain reactions. Robust constraint handling is vital.

The `Stage` Concept in visualCaseGen

and some key lessons in software design

The Stage Concept in visualCaseGen

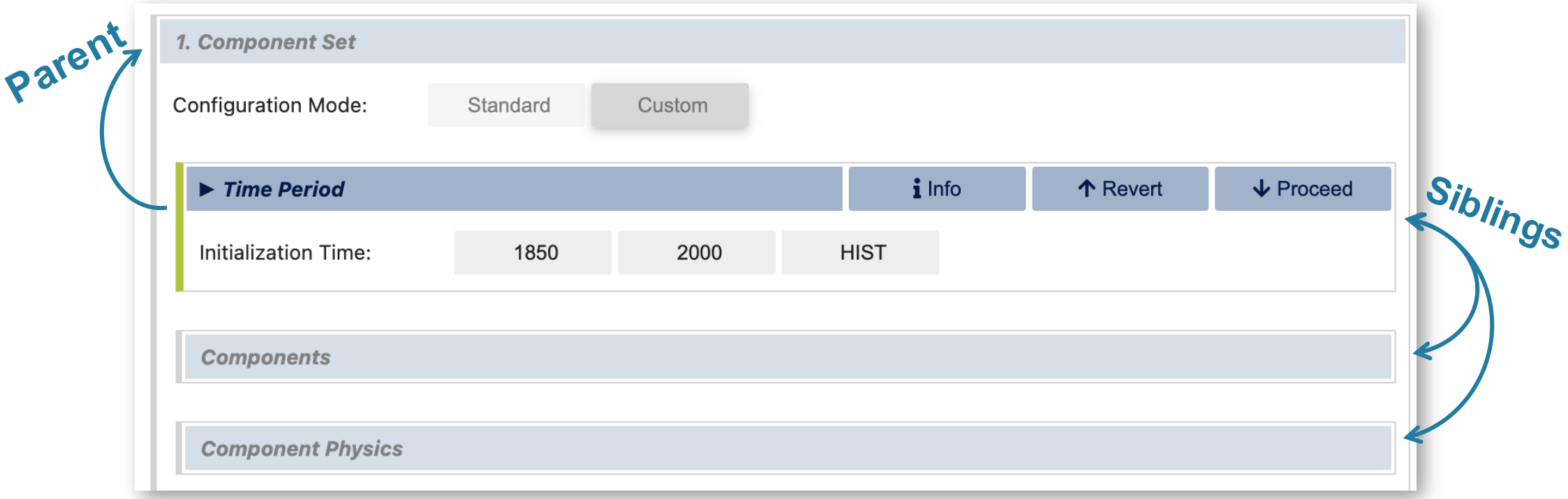
Stage: A collection of config variables that can be configured simultaneously.

Frontend Representation:

▶ Components					Info	Revert	Proceed
▼ ATM	▼ LND	▼ ICE	▼ OCN	▼ ROF	▼ GLC	▼ WAV	
datm	clm	cice5	pop	rtm	cism	ww3	
satm	slim	cice	mom	mosart	dglc	ww3dev	
cam	dlnd	dice	docn	drof	sglc	dwav	
	slnd	sice	socn	srof		swav	

The Stage Concept in visualCaseGen

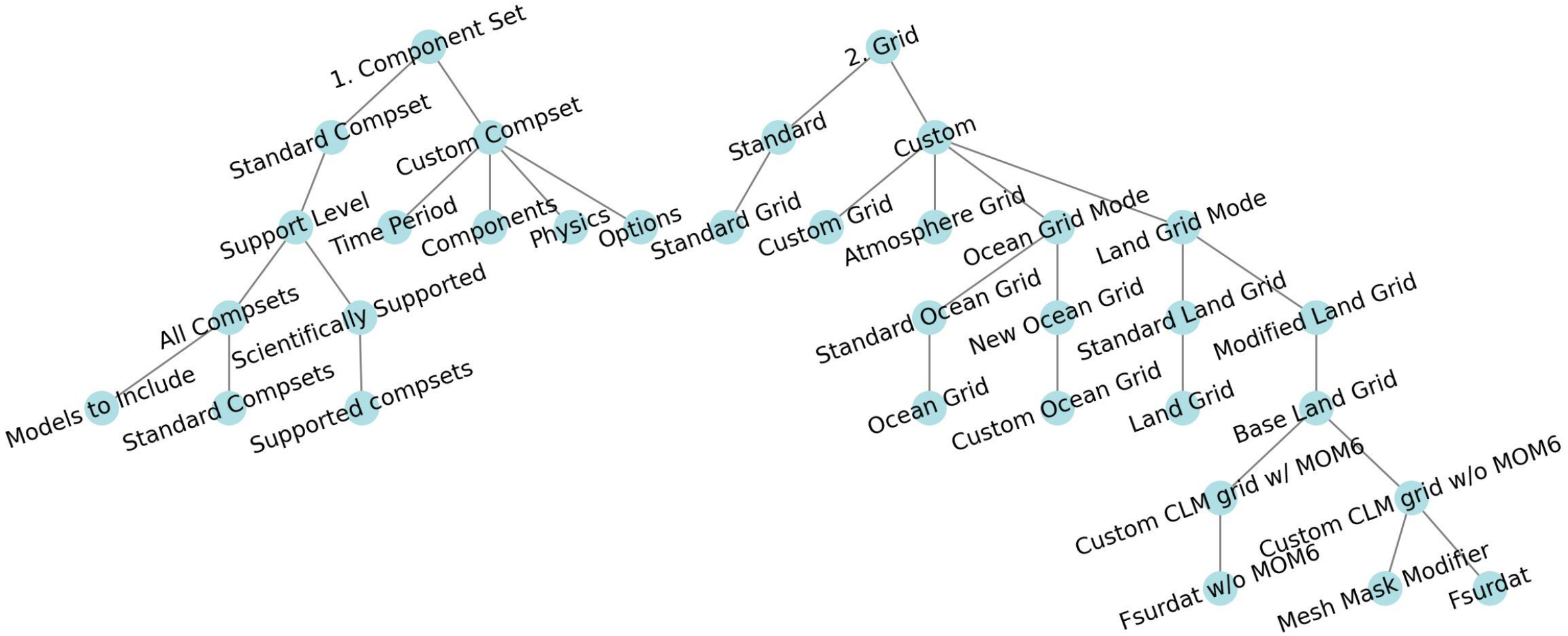
Stage is a *hierarchical* concept.



- Based on the hierarchy, visualCaseGen generates a **Stage** tree and pipeline.

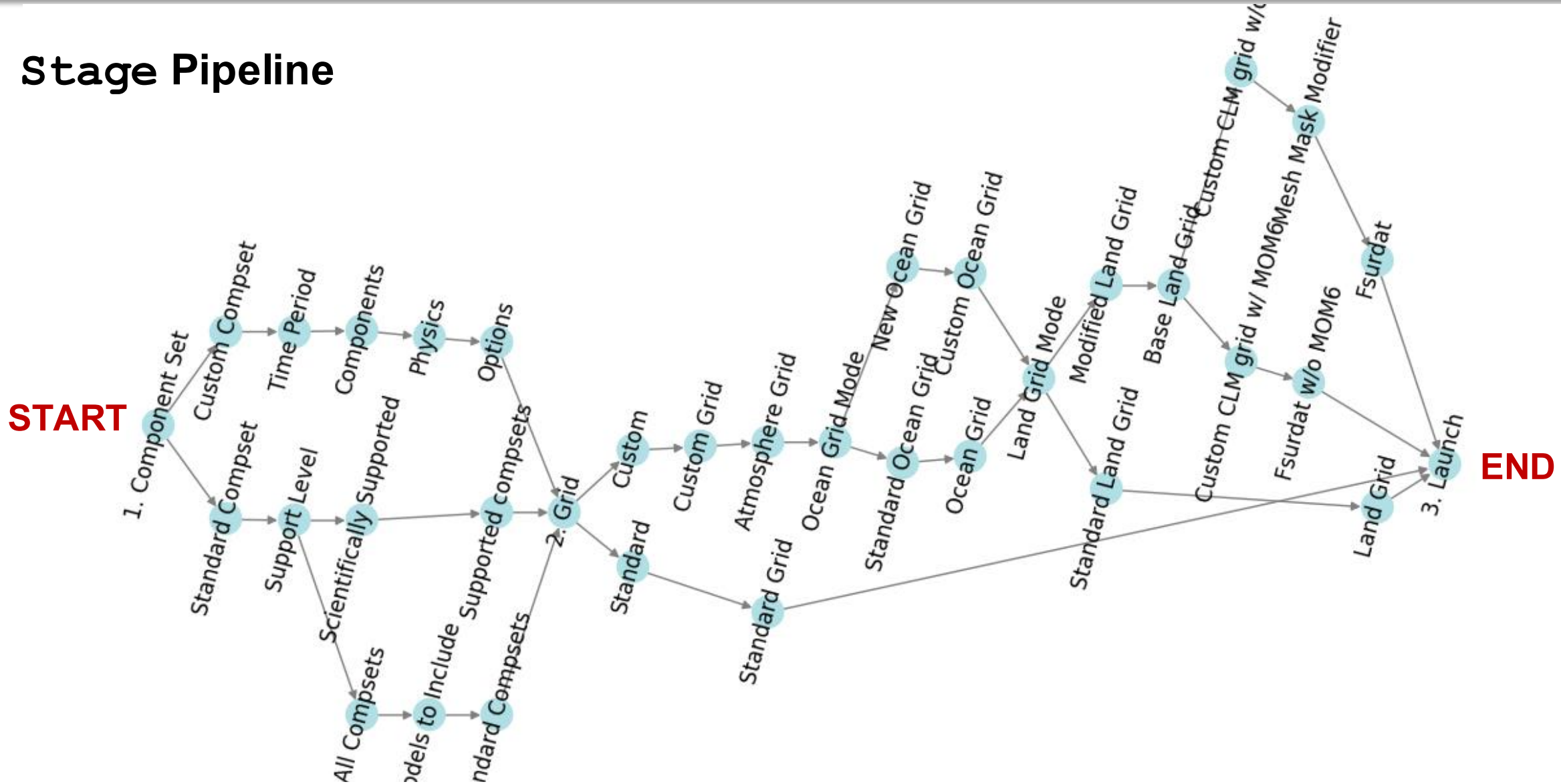
The Stage Concept in visualCaseGen

Stage Tree



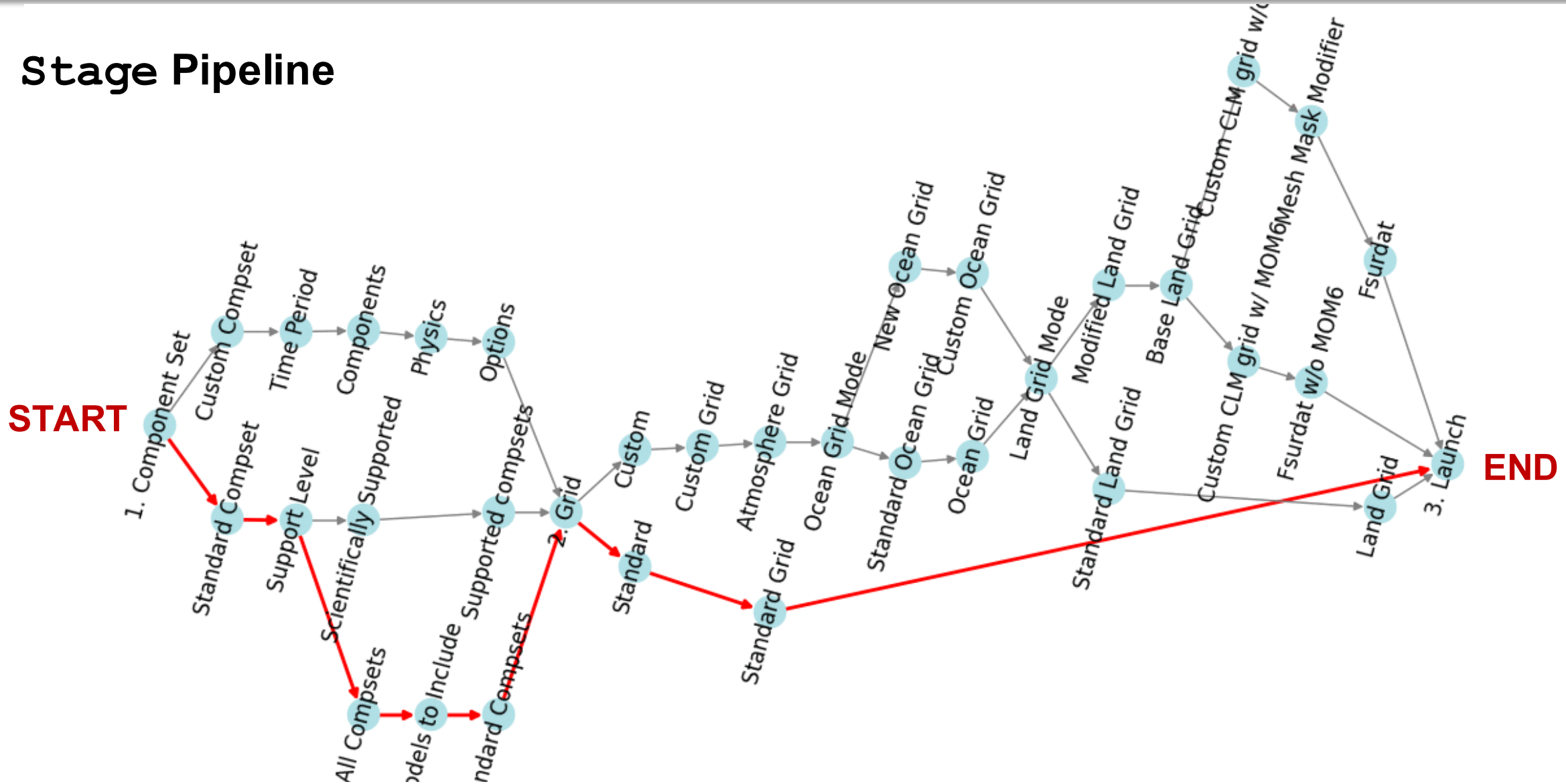
The Stage Concept in visualCaseGen

Stage Pipeline



The Stage Concept in visualCaseGen

Stage Pipeline



The Stage Concept in visualCaseGen

Stage pipeline dictates ***variable precedence***, such that:

- Variables in earlier stages have higher precedence.
- Variables within the same stage have equal precedence.

A complicating factor: The same variable can appear in multiple stages, as long as they are not reachable along the same path.

The `Stage` Concept in visualCaseGen

The **`Stage` Pipeline** must form a directed acyclic graph (DAG). This ensures that:

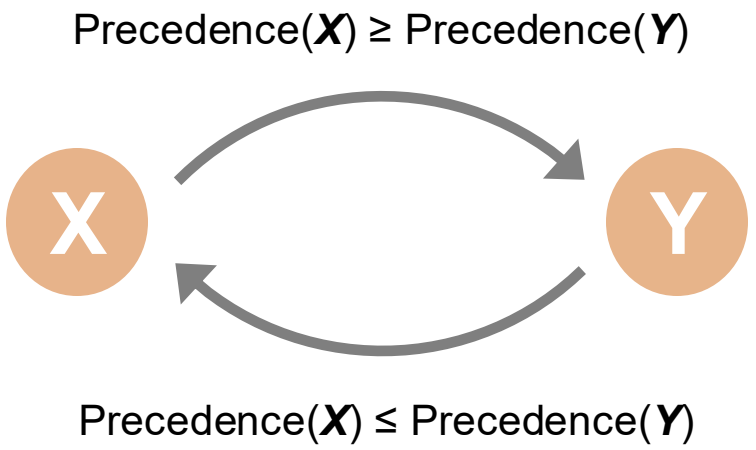
- A consistent **variable precedence** can be established.
- No cycles are encountered by the user.

Stage Mechanism + Constraint Solver

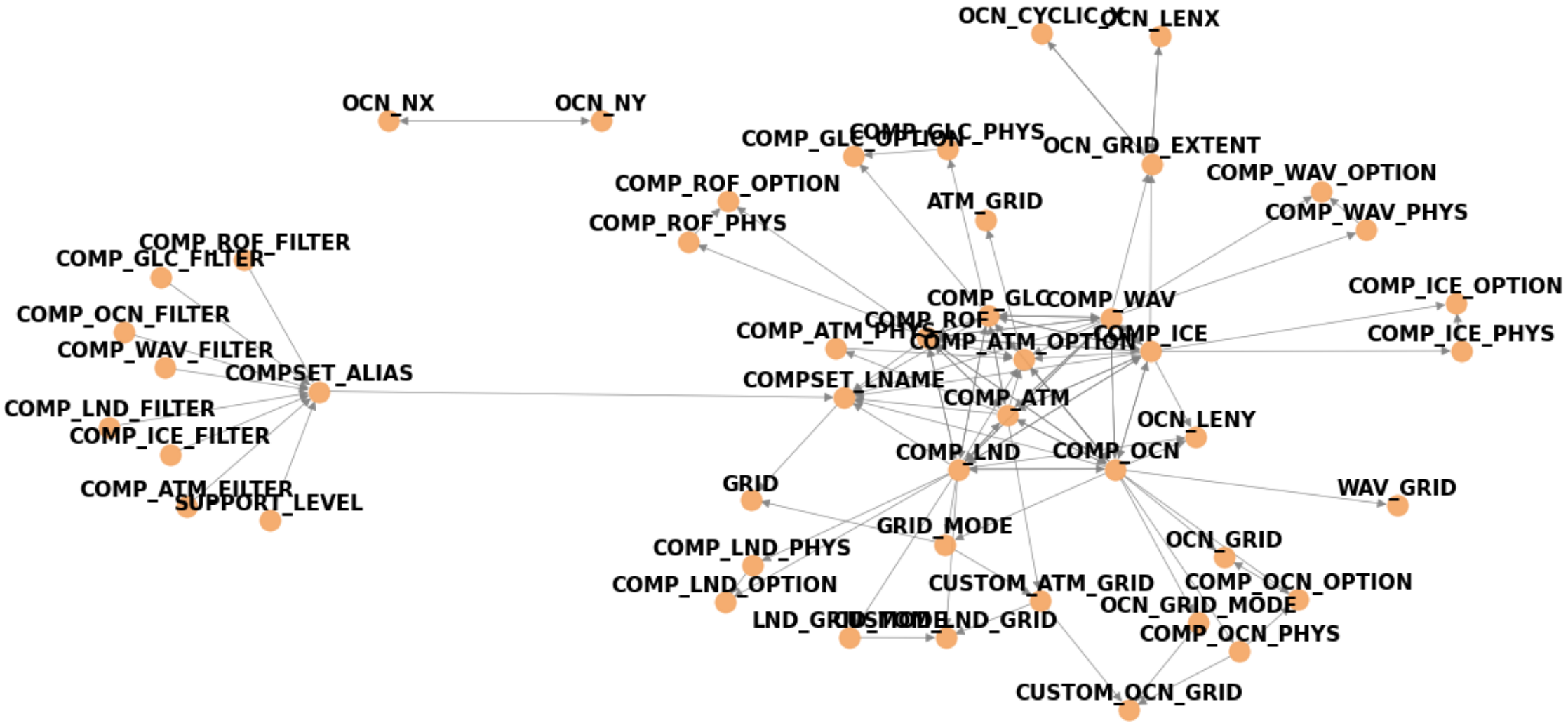
Stage Mechanism + Constraint Solver

Constraint Graph: Formed by the specified constraints and variable precedence.

- For each variable ***X*** and ***Y*** occurring in the same constraint:



Constraint Graph



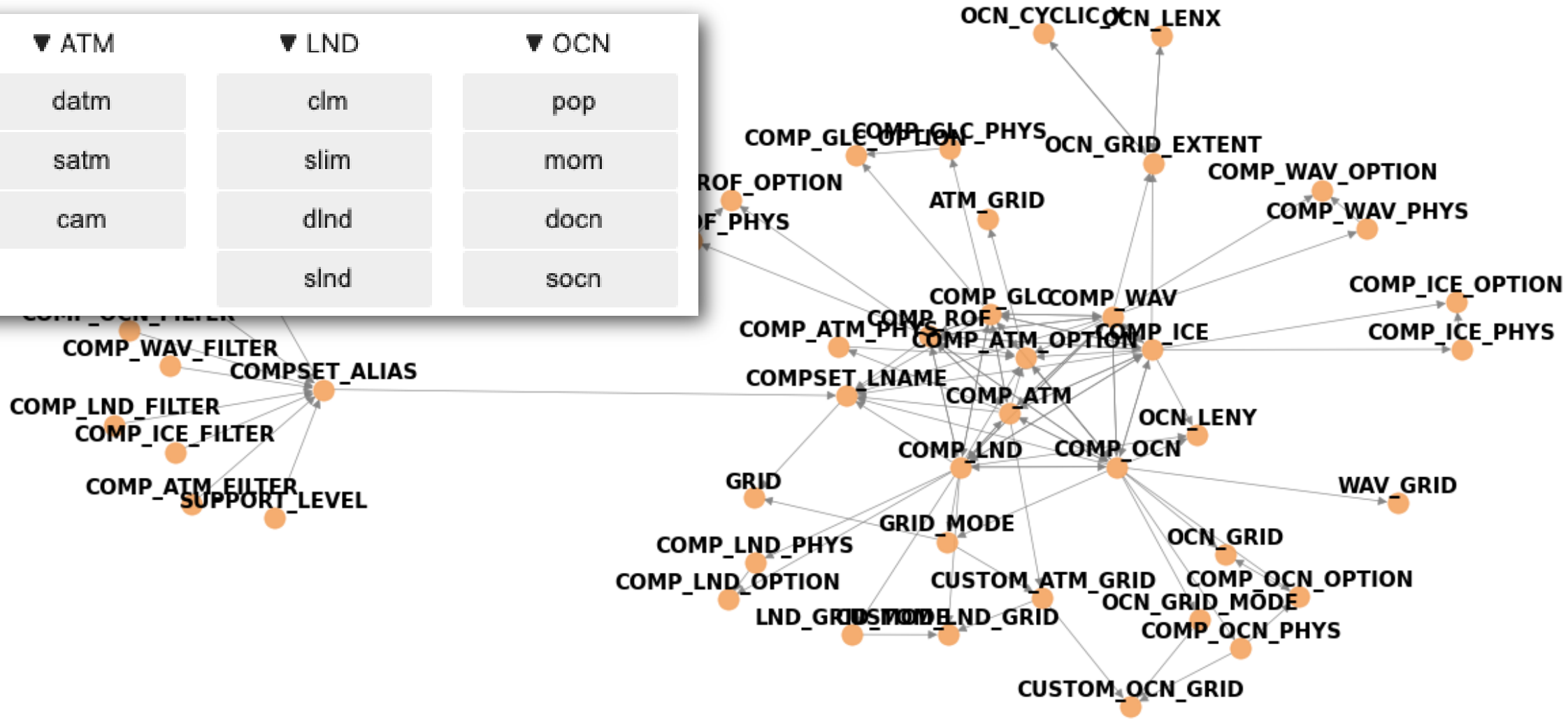
Constraint Graph

A user change initiates a traversal of the constraint graph:

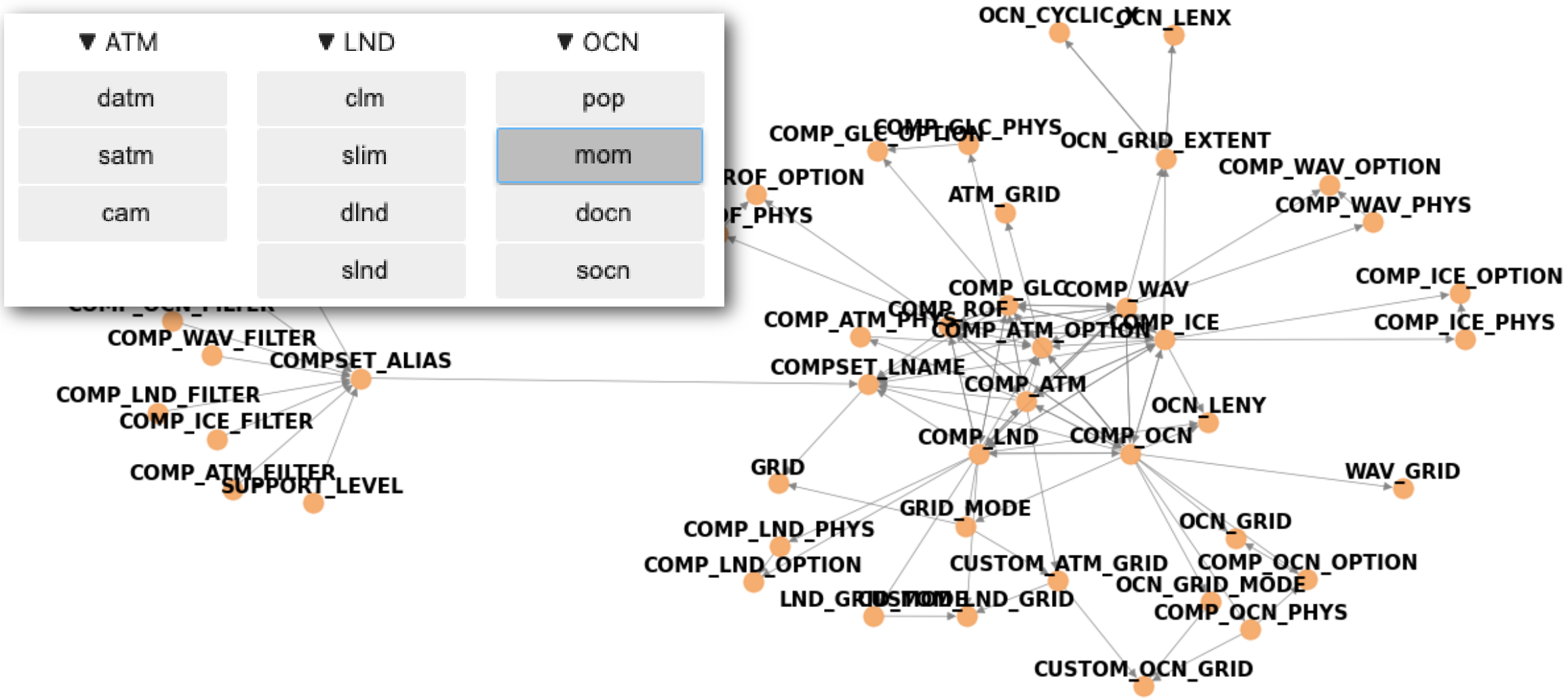
- All potentially affected variables are visited: This involves calling Z3 to check if the validity of options changed.
- The extent of the traversal depends on the user input, the stage hierarchy, and constraints.

Constraint Graph

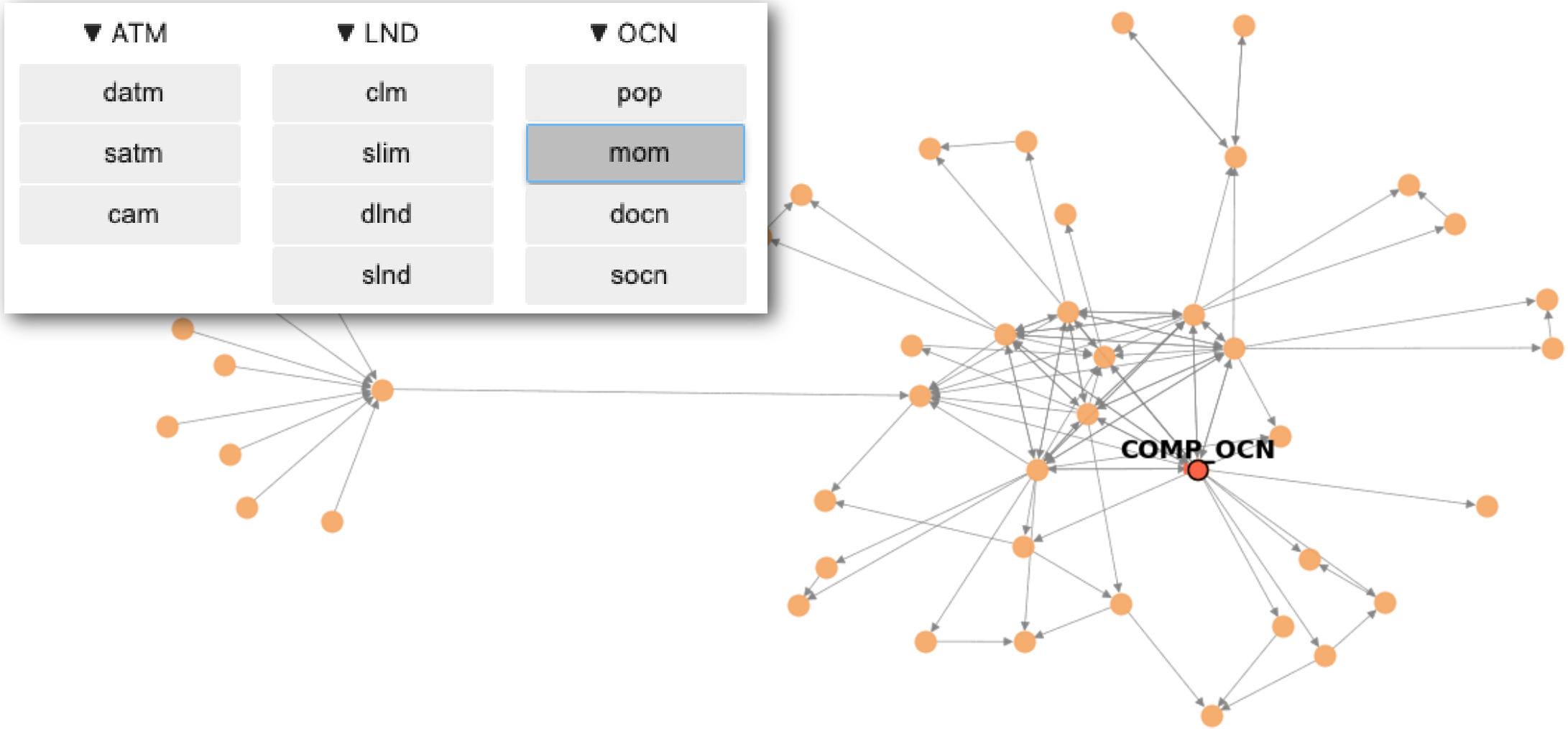
ATM	LND	OCN
datm	clm	pop
satm	slim	mom
cam	dlnd	docn
	slnd	socn



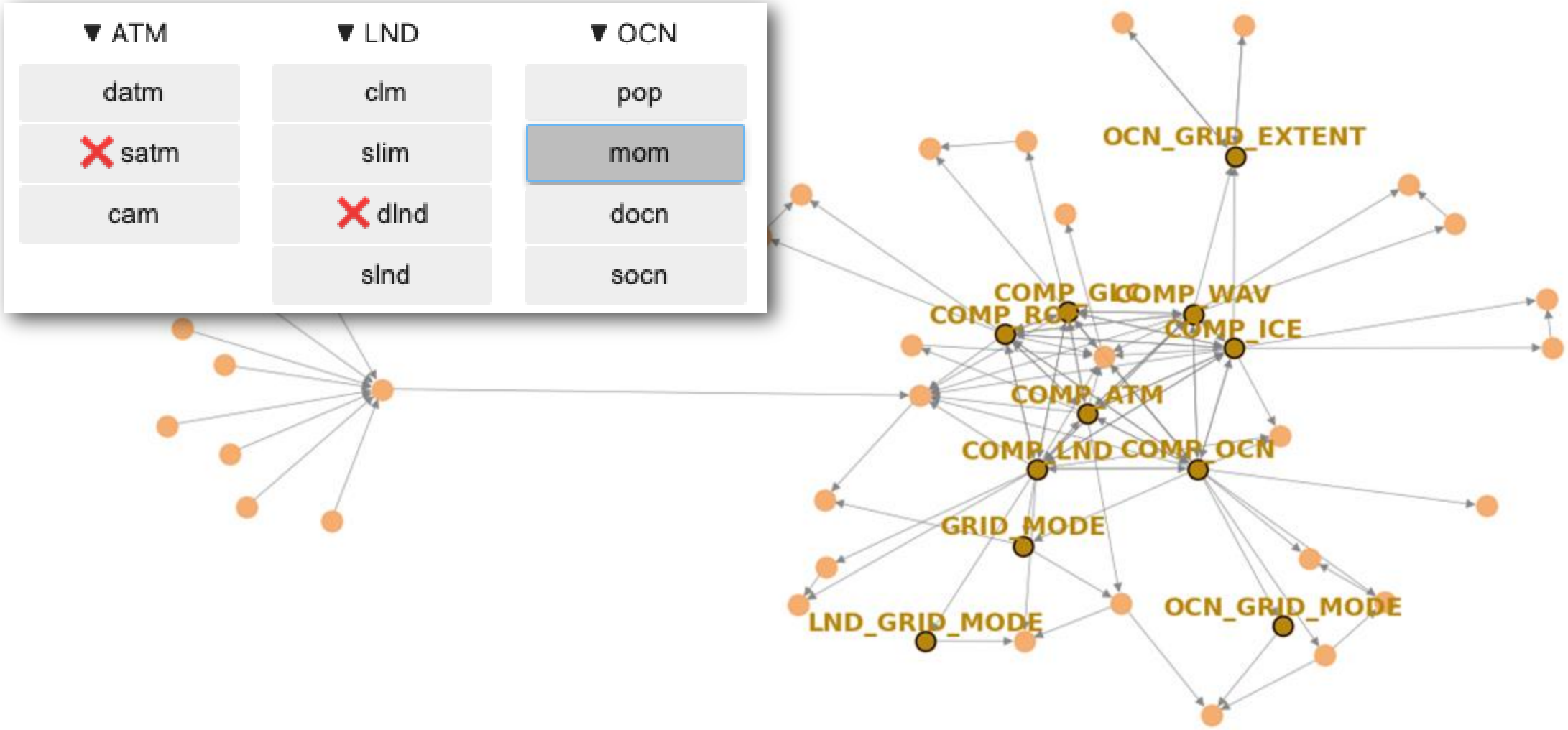
Constraint Graph



Constraint Graph

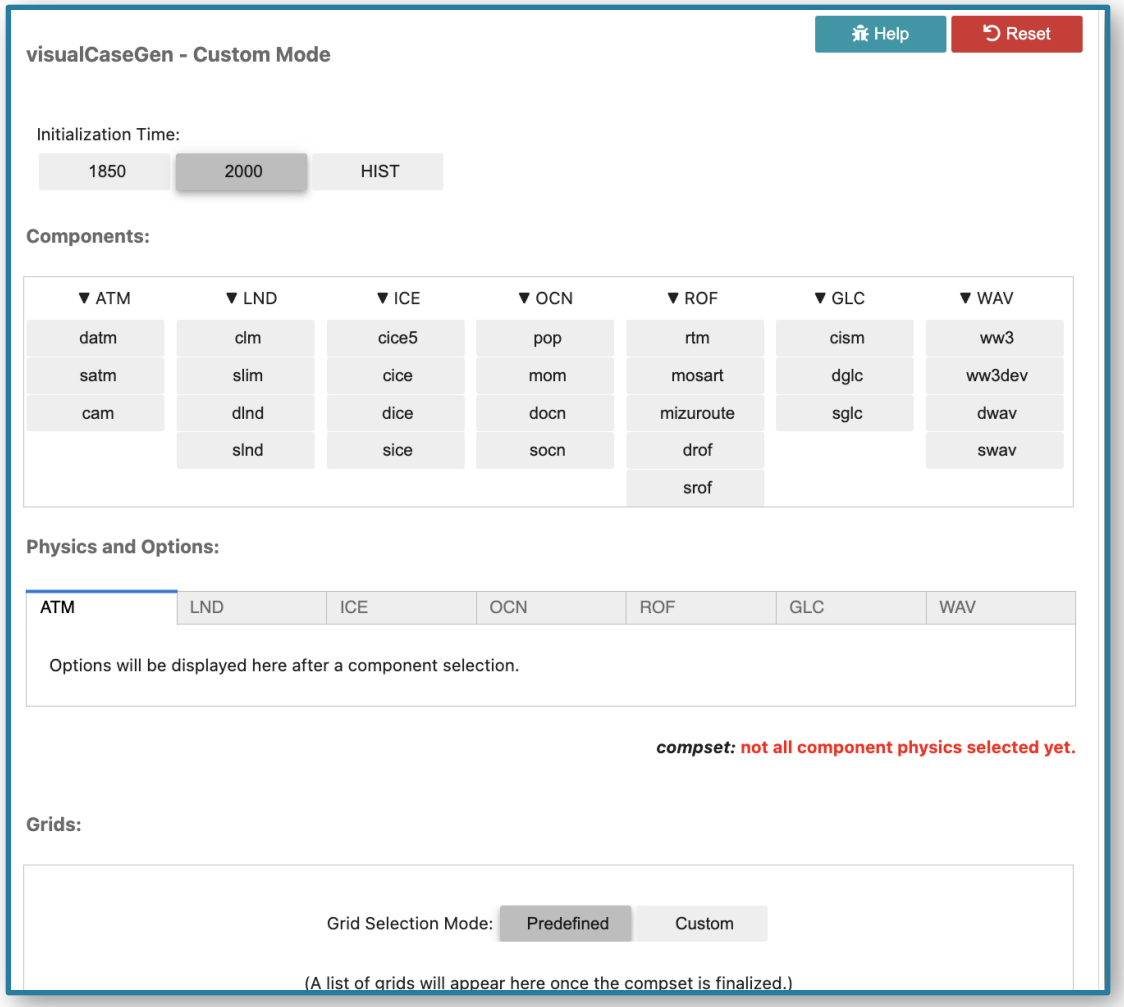


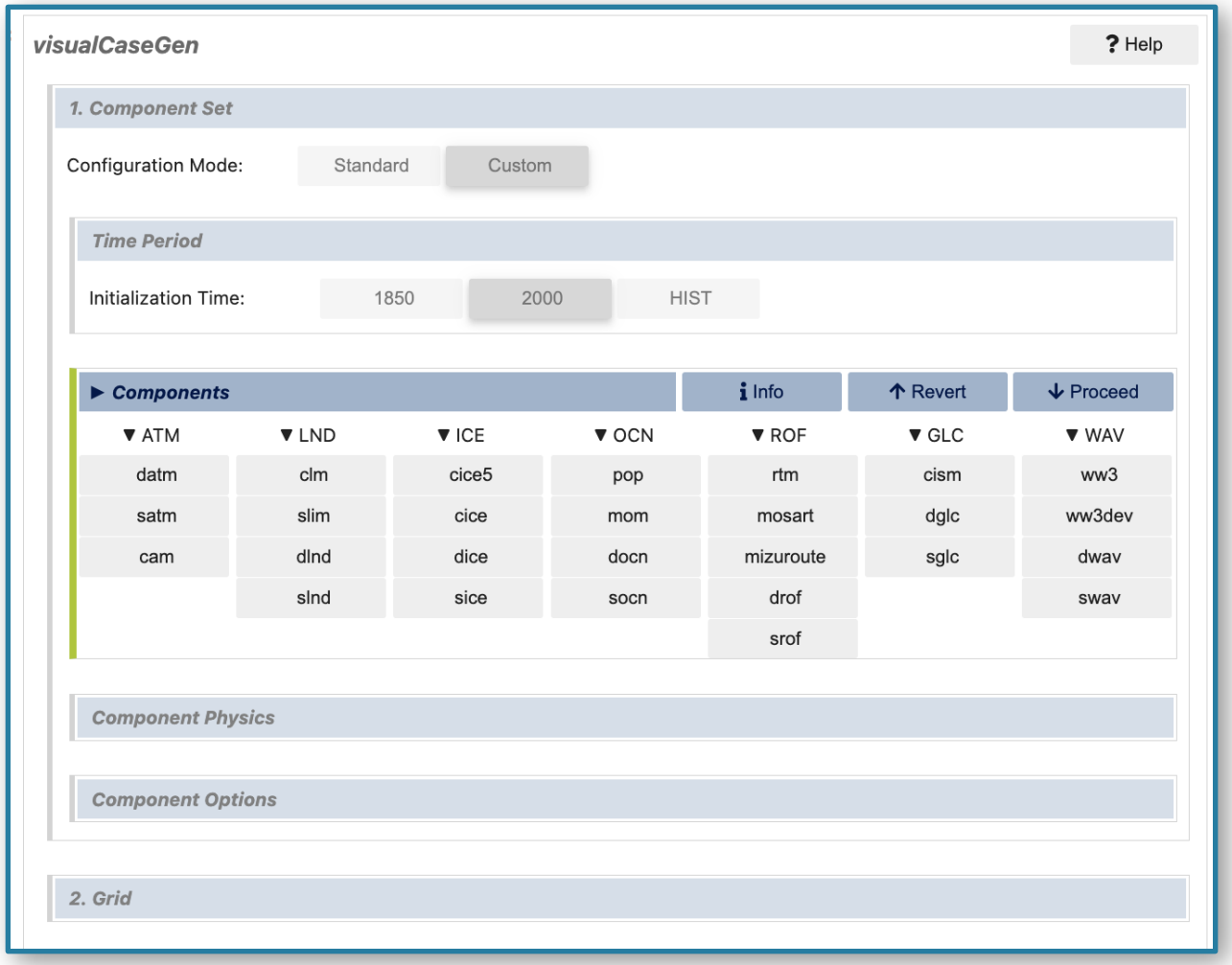
Constraint Graph



Conceptual Construct Matters.

Prototype vs Product





The `Stage` Concept in visualCaseGen

With the introduction of the **`Stage`** Concept:

- **UX enhanced**: Clearer guidance on user actions.
- **Robustness increased**: Clearer requirements and invariants such as variable precedence, state change rules, and relational dependencies.
- **Maintainability improved**: LOC decreased significantly.
- **Better performance**: A more efficient constraint solver implementation tripled the computational performance.

Takeaway

Software architecture is infrequently discussed or considered: Focus tends to be on low-level details.

But high-level design constructs (i.e., functionalities, patterns, structures) have a significant influence on overall software quality. We should make sure:

- All the conceptual constructs are identified, incorporated, and documented.
- The relationships between them are well-established and understood.
- Requirements are carefully analyzed and adhered to.

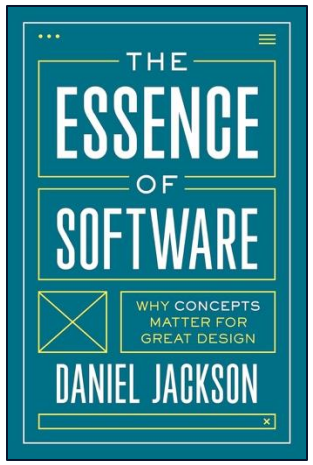
> The hard part of building software is the conceptual construct,
> not the labor of representing it.

– FP Brooks. “No Silver Bullet” (1987)

> The hard part of building software is the conceptual construct,
> not the labor of representing it.
>
> – FP Brooks. “No Silver Bullet” (1987)

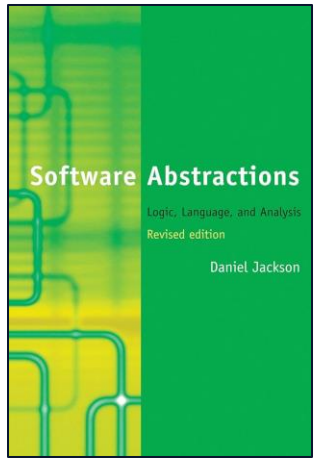
> Agilistas prioritize **code** over ***design + requirements + specifications***.
> But in 10 years, those will be the only things we’ll write.
>
> – Daniel Jackson, “What Makes Software Work?” (2024)

References



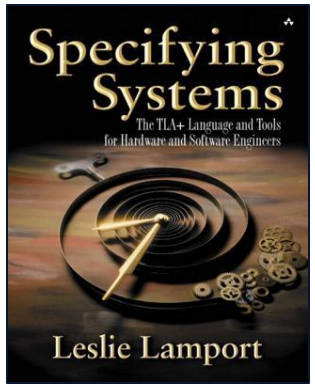
"What matters is the fundamental structure of the design. If you get it wrong, there is no amount of bug fixing and refactoring that will produce a reliable, maintainable, and usable system."

– D. Jackson. ***The essence of software***. (2021)



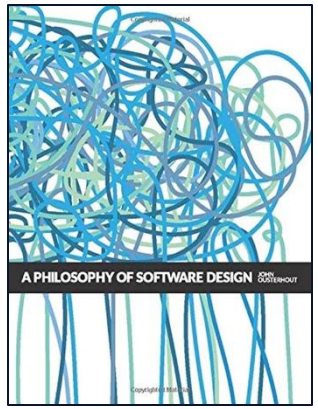
"Code is a poor medium for exploring abstractions."

– D. Jackson. ***Software Abstractions***. (2012)



"Writing is nature's way of letting you know how sloppy your thinking is."

– L. Lamport. ***Specifying Systems***. (2002)



"Be on the lookout for opportunities to improve the design and plan on spending some fraction of your time on design improvements."

– J. Ousterhout. ***A Philosophy of Software Design***. (2018)

A stable beta version released and available at:
https://github.com/ESMCI/visualCaseGen

Official release this winter.

Thanks!

altuntas@ucar.edu