

# Populations, Variety and Selection: Verifying Complex Designs

Austin DVClub 4/19/2011

Ken Albin

[ken.albin@systemsemantics.com](mailto:ken.albin@systemsemantics.com)

# What is going on?

Why is it that no matter how thorough we are:

- customers still find bugs?
- randomization finds combinations we didn't think about for our directed tests?
- writing formal properties finds errors and ambiguities in the specification?

Is doing the same thing more efficiently really going to get us to a higher level of quality sooner?

# Why this talk?

- Verification planning is often focused on low-level details without first considering the higher-level alternatives.
- Focus on ROI seems to be taking us away from methods employing variety.
- Complexity is the root problem we struggle with
  - what can we learn from complexity research?

# What are complex systems?

It turns out that there are several definitions and models of complex systems, generally tuned to the field in which they are being used.

In the book *Harnessing Complexity*, Axelrod and Cohen describe systems “...where the full consequences of actions may be hard – even impossible – to predict.”

Their Complex Adaptive Systems model captured ideas I saw applied in successful verification planning and execution efforts.

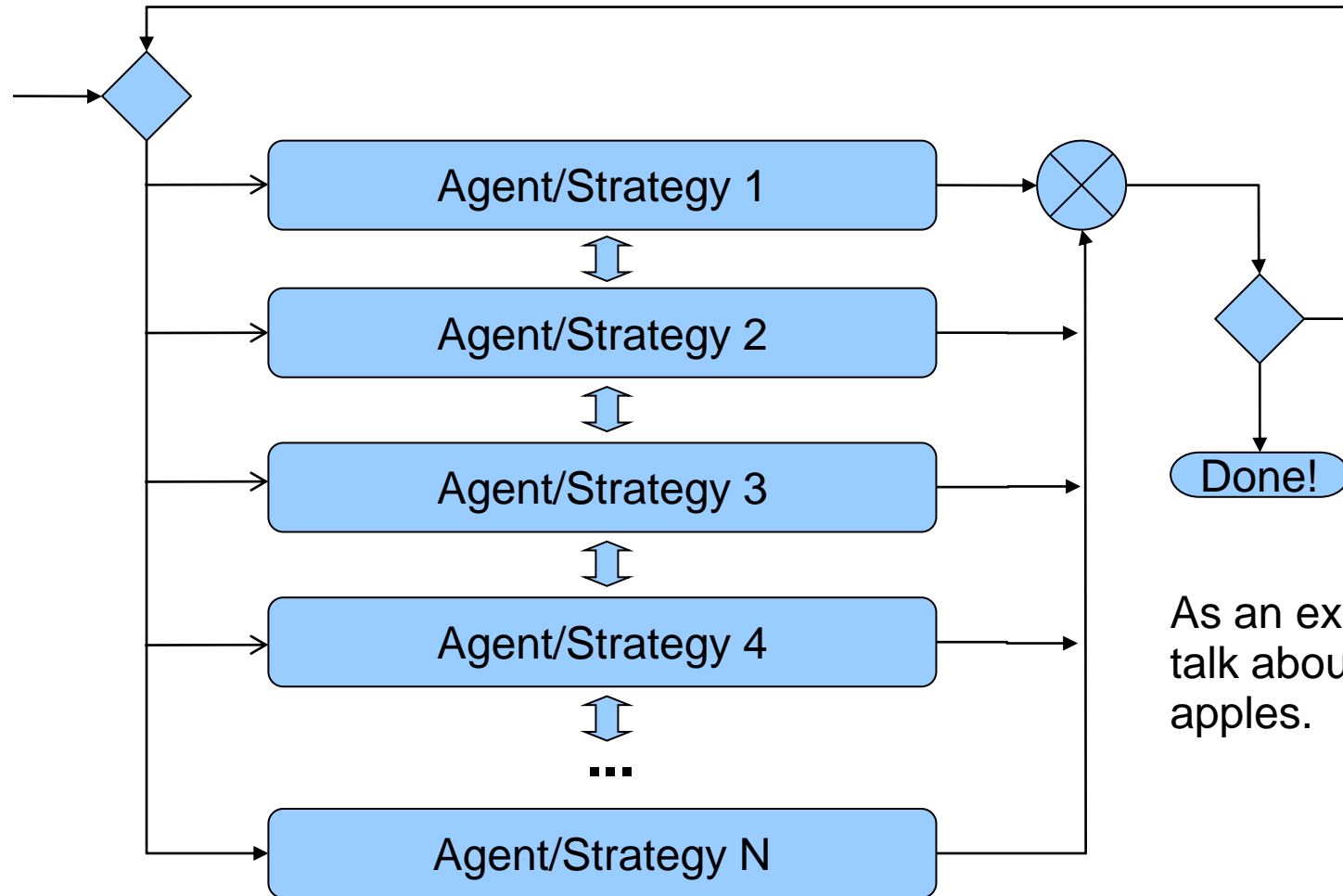
# A Brief Introduction to Complex Adaptive Systems

# Complex Adaptive Systems (CAS)

- Axelrod and Cohen's CAS framework combines aspects of two familiar complex systems:
  - Biological systems
  - Agent-based software systems

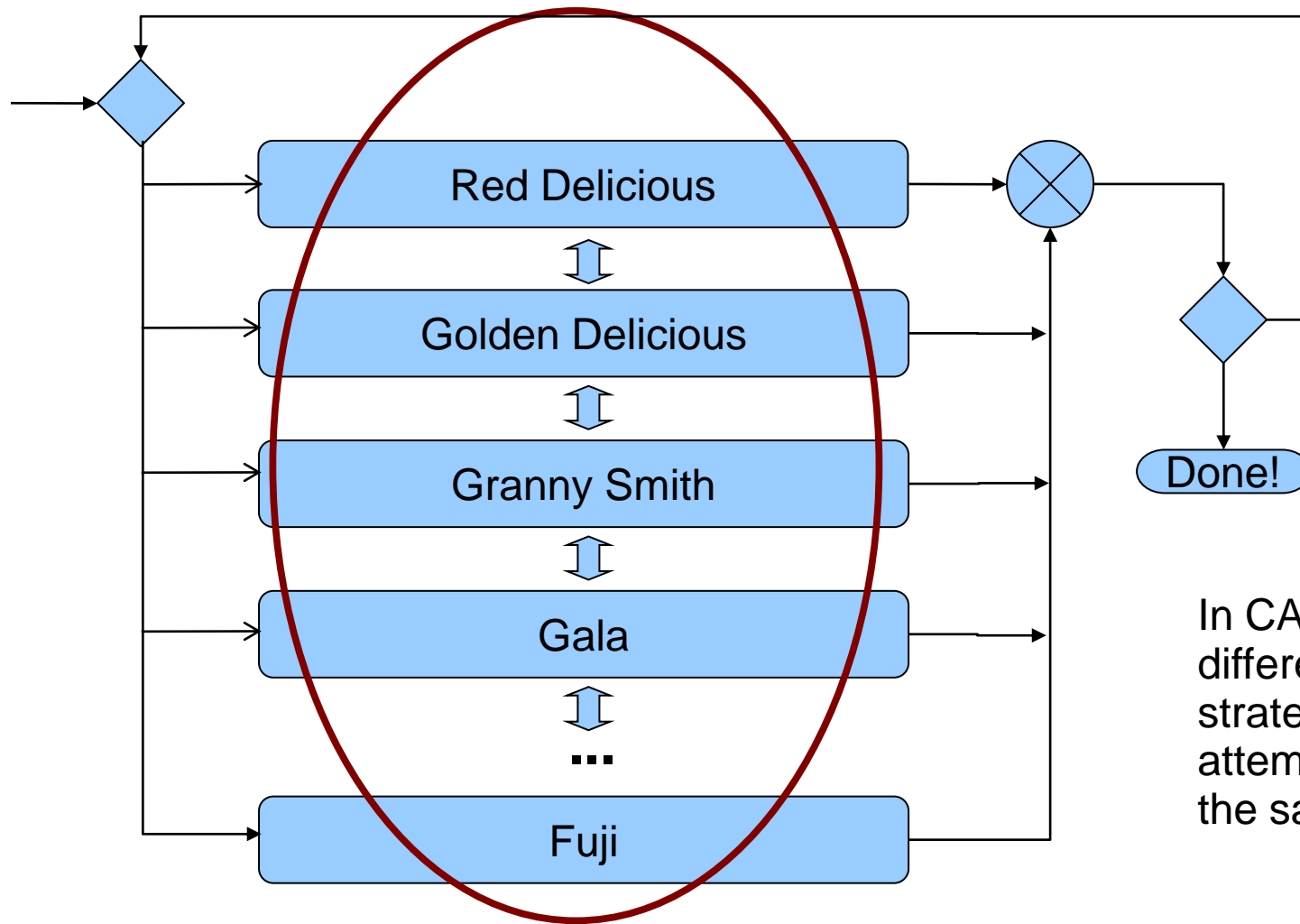
The result is a framework for reasoning about complex systems – not a magic oracle, but an aid to decision making.

# A simplified CAS model



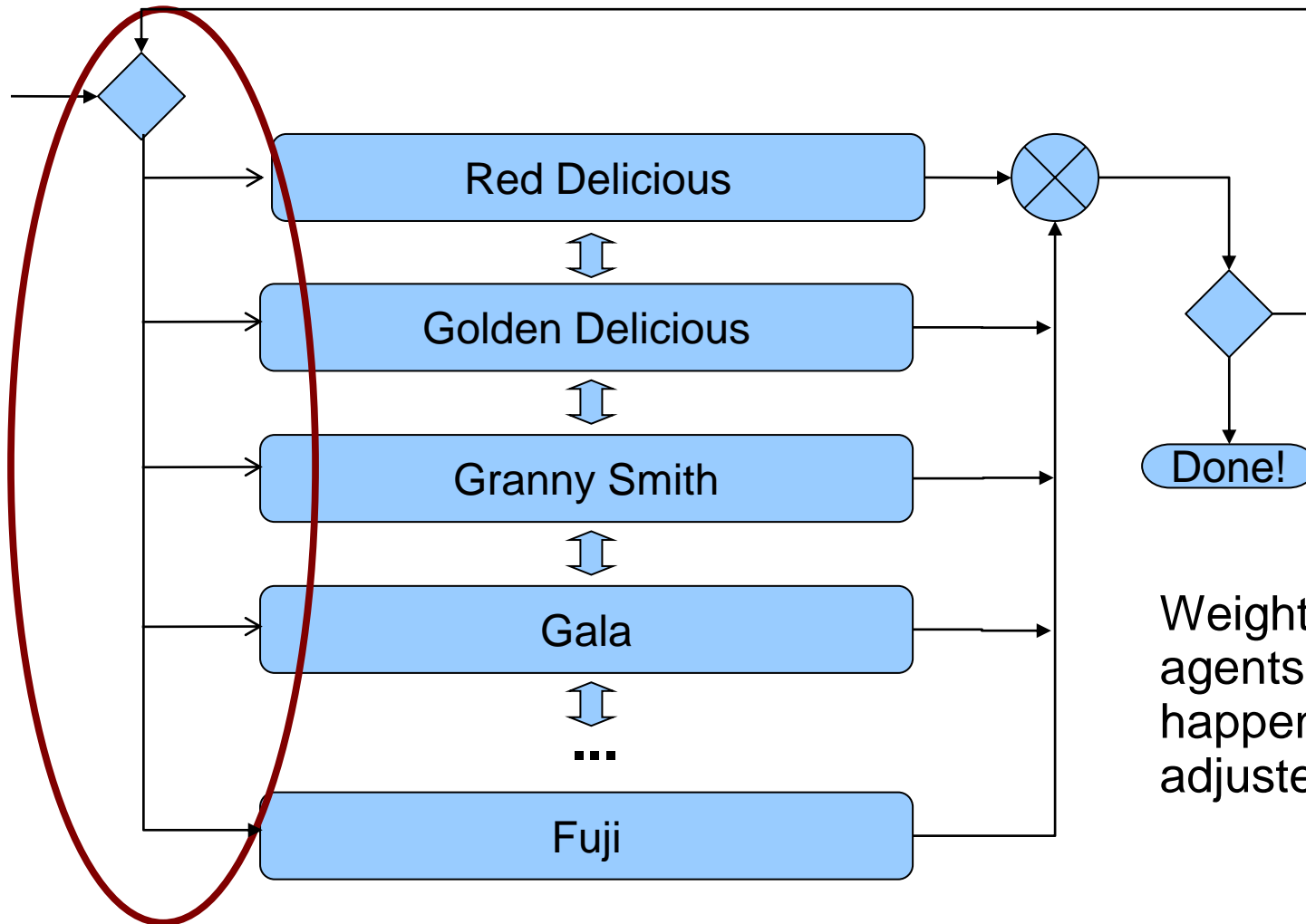
As an example, let's talk about growing apples.

# Populations of agents or strategies



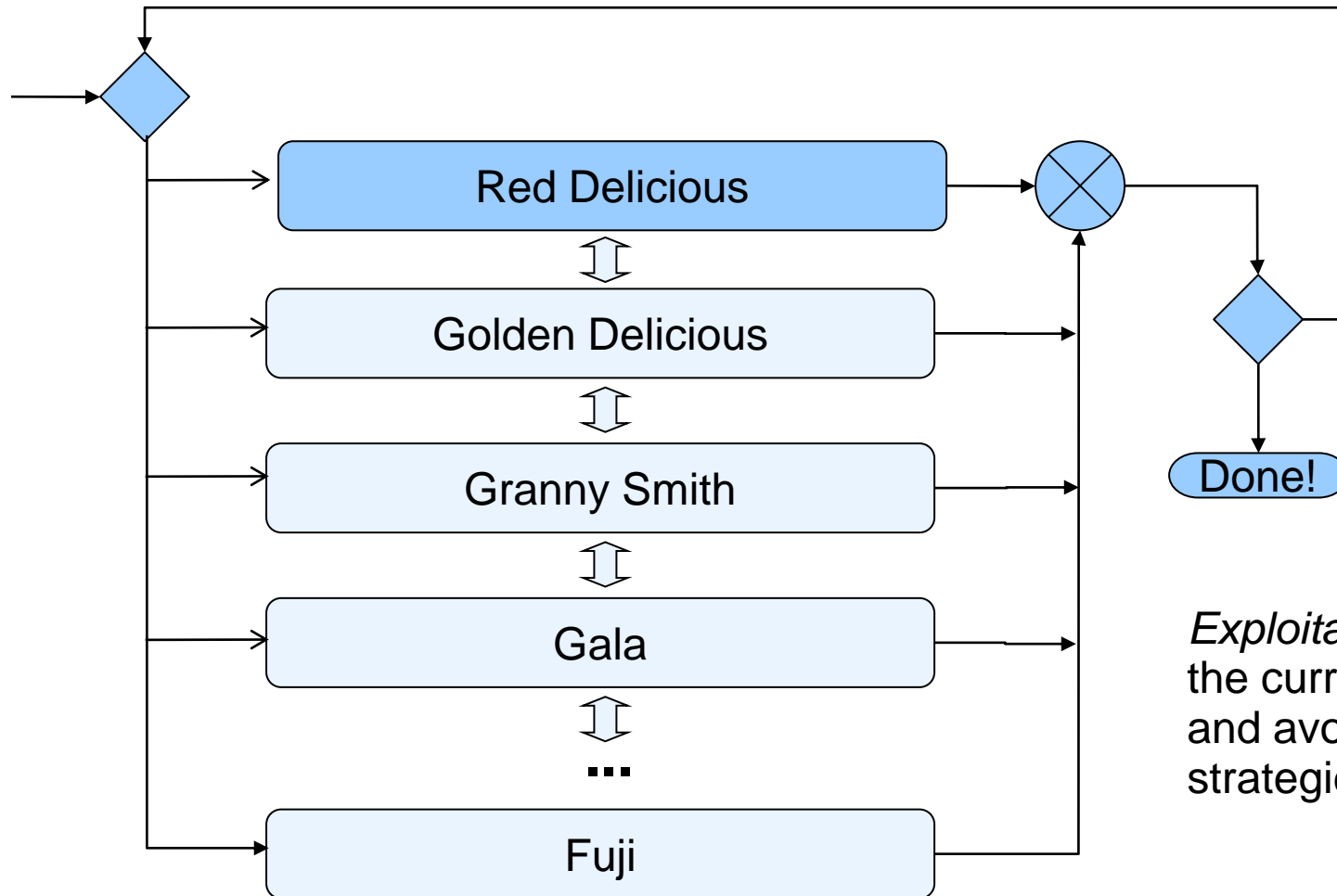
In CAS, there is a pool of different types of agents or strategies in action, each attempting to accomplish the same goal.

# Selecting the amount of variety



Weighting of the different agents or strategies happens initially and is adjusted during execution.

# Exploitation (limiting variety)



*Exploitation* selects only the current best solution and avoids less efficient strategies.

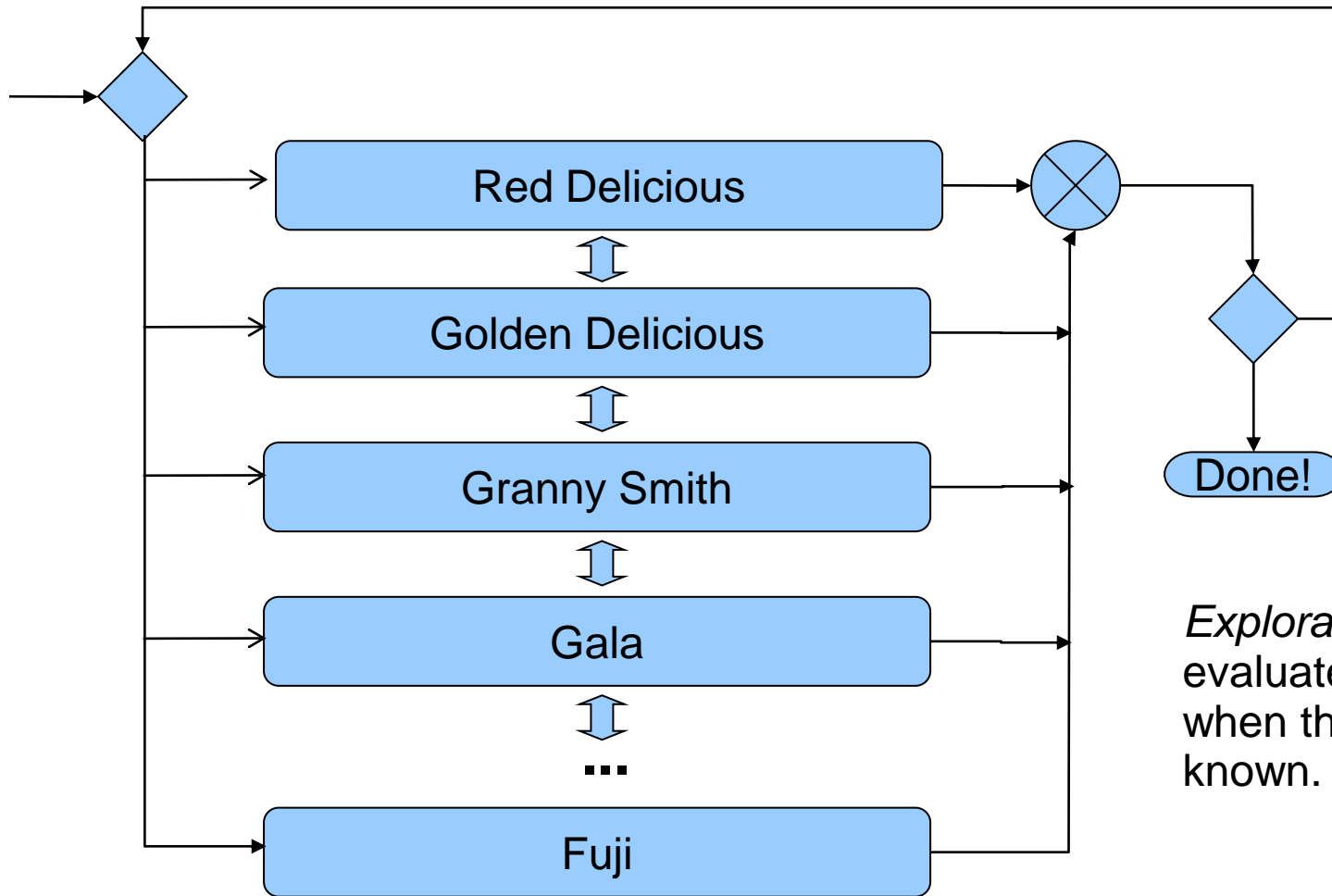
# When to limit variety

## ***Exploitation is appropriate if:***

- You are sure you have the best solution
- The problem is not going to change

*In our example, if you have determined everyone's favorite apple and are sure there will be no changes in market preference and ideal growing conditions, then producing other types of apples would be a waste of resources.*

# Exploration (encouraging variety)



*Exploration* tries and evaluates different strategies when the best solution is not known.

# When to encourage variety

***Exploration is valuable for problems that:***

- are long term or widespread
- provide fast reliable feedback
- have low risk of catastrophe from exploration \*
- have looming disaster

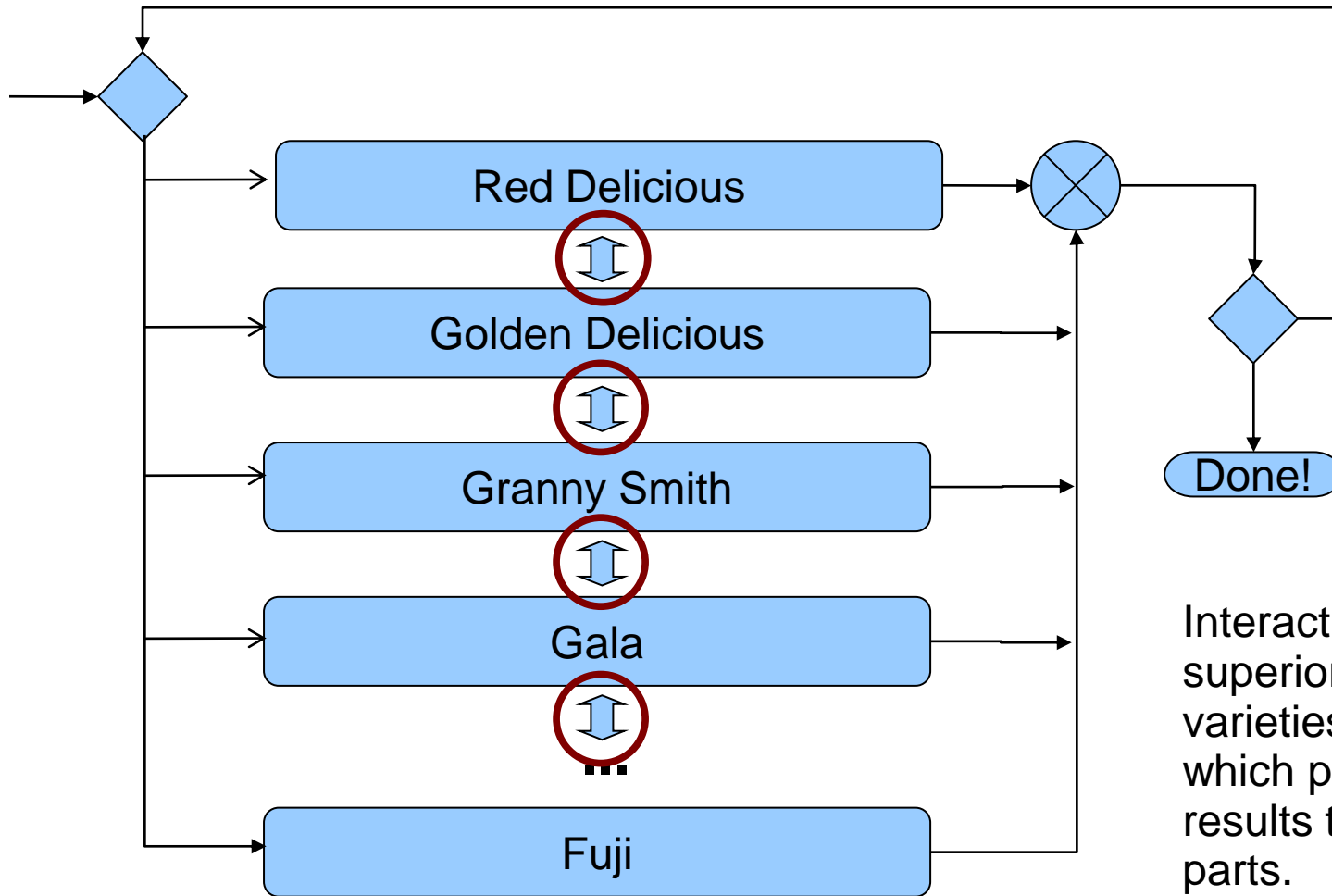
*With apple trees, some amount of variety seems prudent.*

# Exploitation vs. Exploration

**The big knob to turn in a CAS is the amount of variety in the agents or strategies.**

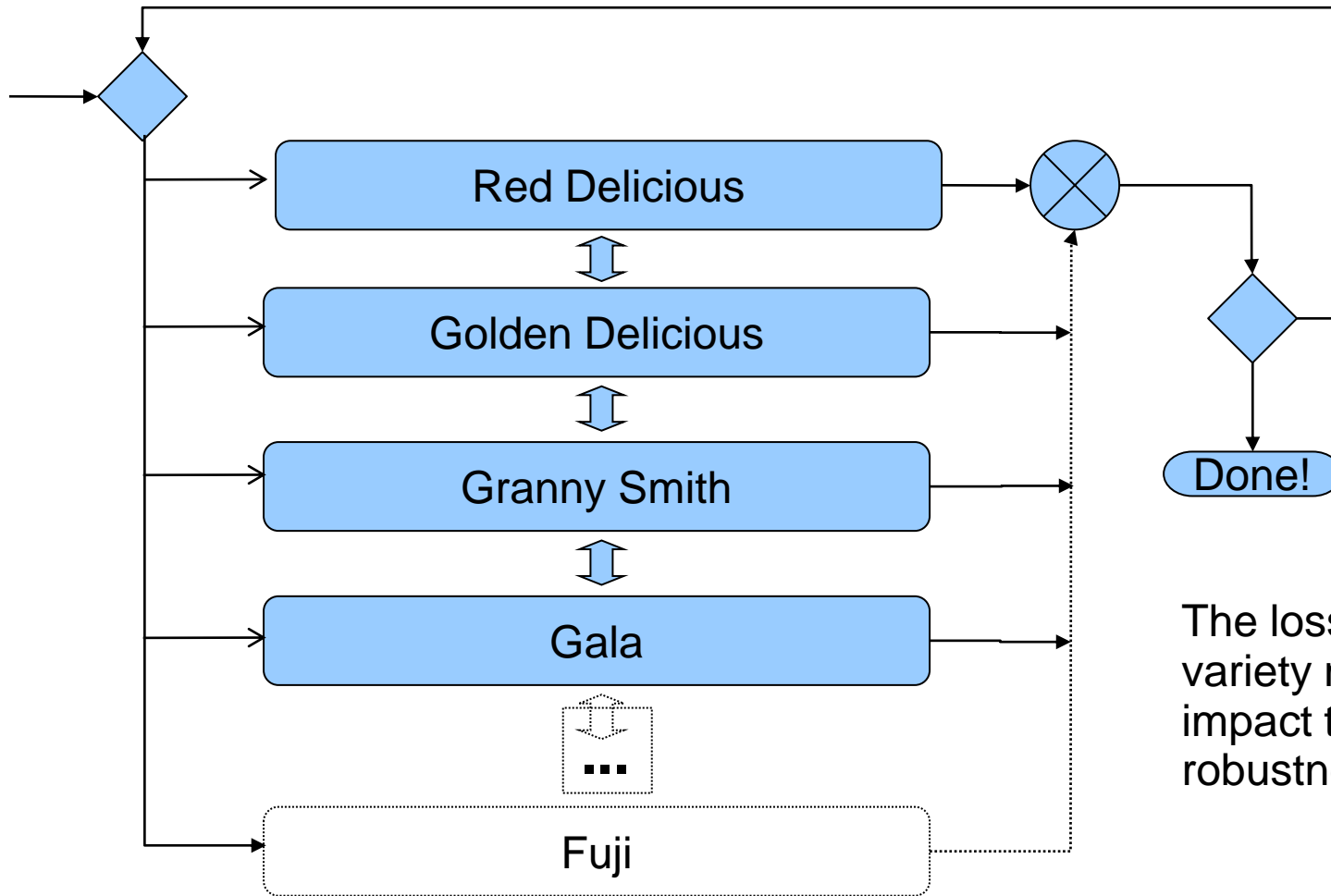
- It is often the case that a CAS begins in exploration mode and moves to exploitation over time.
- However, reducing variety too quickly results in *premature convergence*, and too slowly results in *eternal boiling*

# Interaction



Interaction can create superior new hybrid varieties or strategies which produce better results than the sum of the parts.

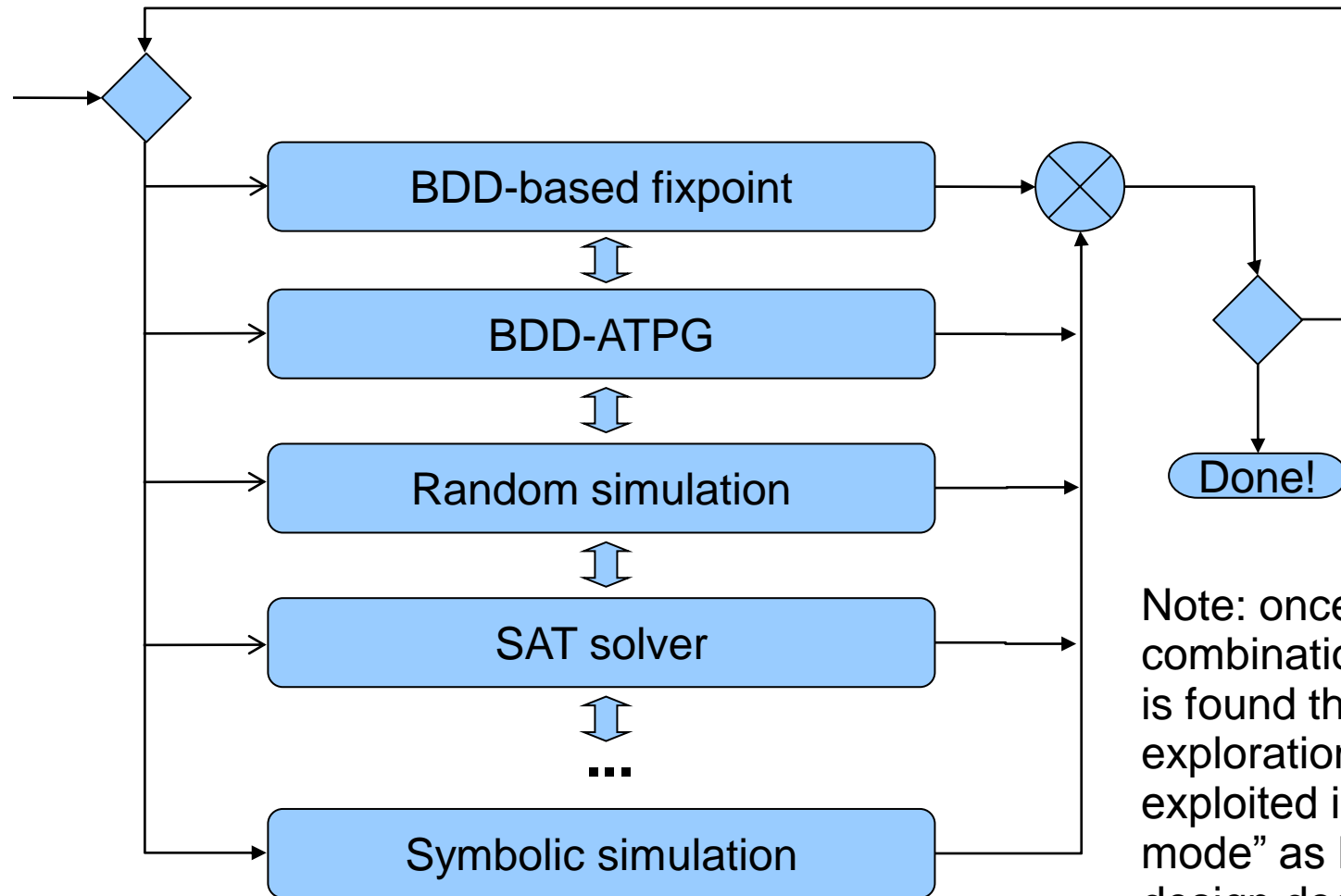
# Extinction



The loss of available variety may seriously impact the adaptability and robustness of a system.

# Example: intelligent agents in action

# Under the hood in formal tools



Note: once a successful combination of engines is found through exploration, it can be exploited in “regression mode” as long as the design doesn't change much.

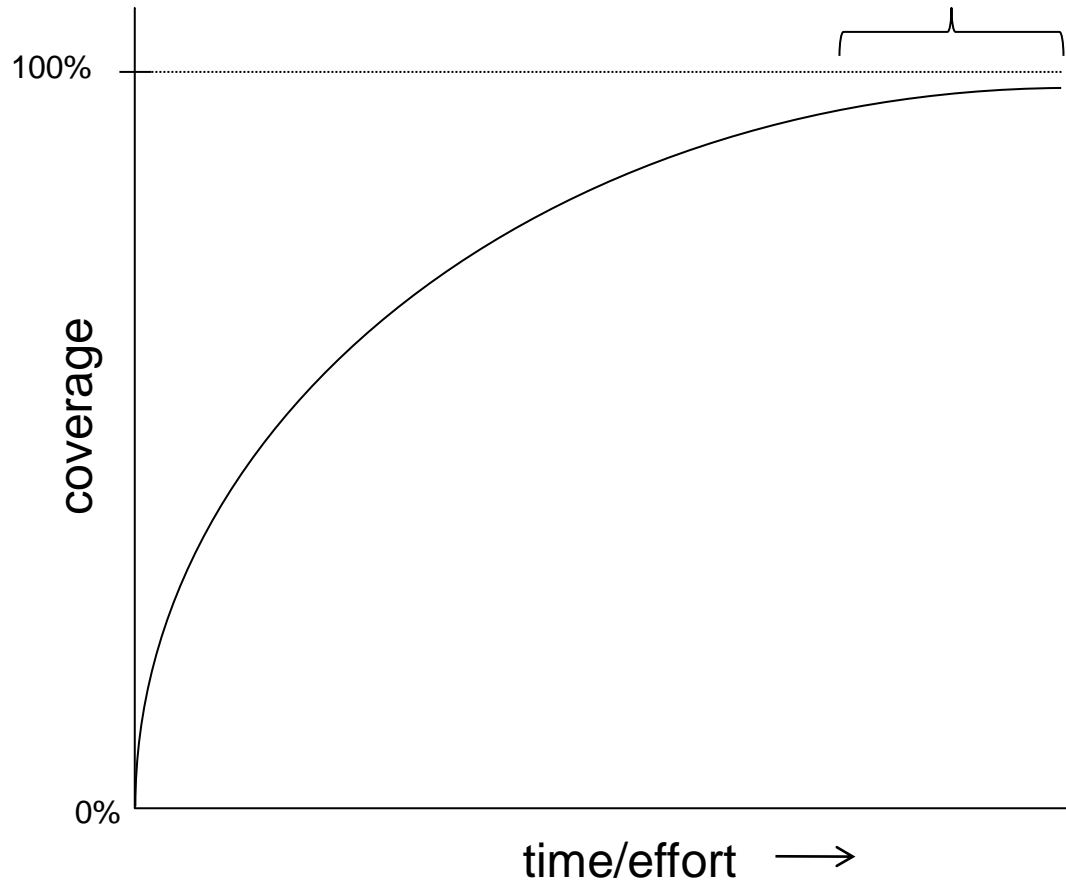
# Applying the framework to verification planning and execution

# Checklist for applying the framework

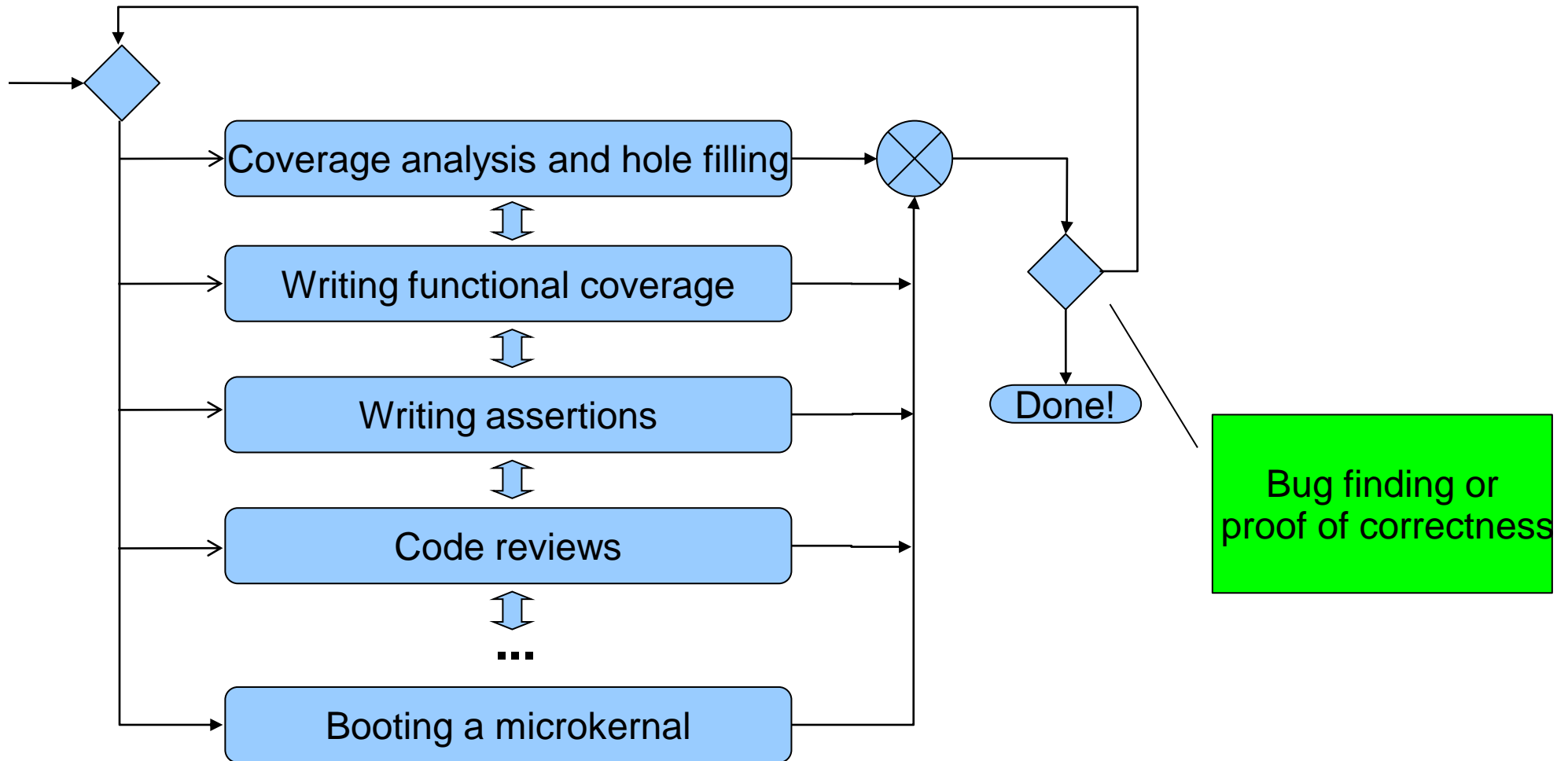
- Identify population**
- Identify selection criteria**
- Analyze impact of variety**
  - Exploitation
    - Best solution?
    - Stable problem?
  - Exploration
    - Long term problem?
    - Fast, reliable feedback?
    - Looming disaster?
- Analyze interaction benefits**
- Analyze extinction risk**

Example: is it worth achieving 100% coverage?

# The last few percent



# Verification planning and execution



# 100% coverage analysis

- ✓ **Identify population**
- ✓ **Identify selection criteria**
- ✓ **Analyze impact of variety**
  - Exploitation
    - Best solution?
    - Stable problem?
  - ✓ Exploration
    - ✓ Long term problem?
    - ✓ Fast, reliable feedback?
    - ✓ Looming disaster?
- ✓ **Analyze interaction benefits**
- ✓ **Analyze extinction risk**

Not known to be best solution, correlating coverage to design changes over time is a problem.

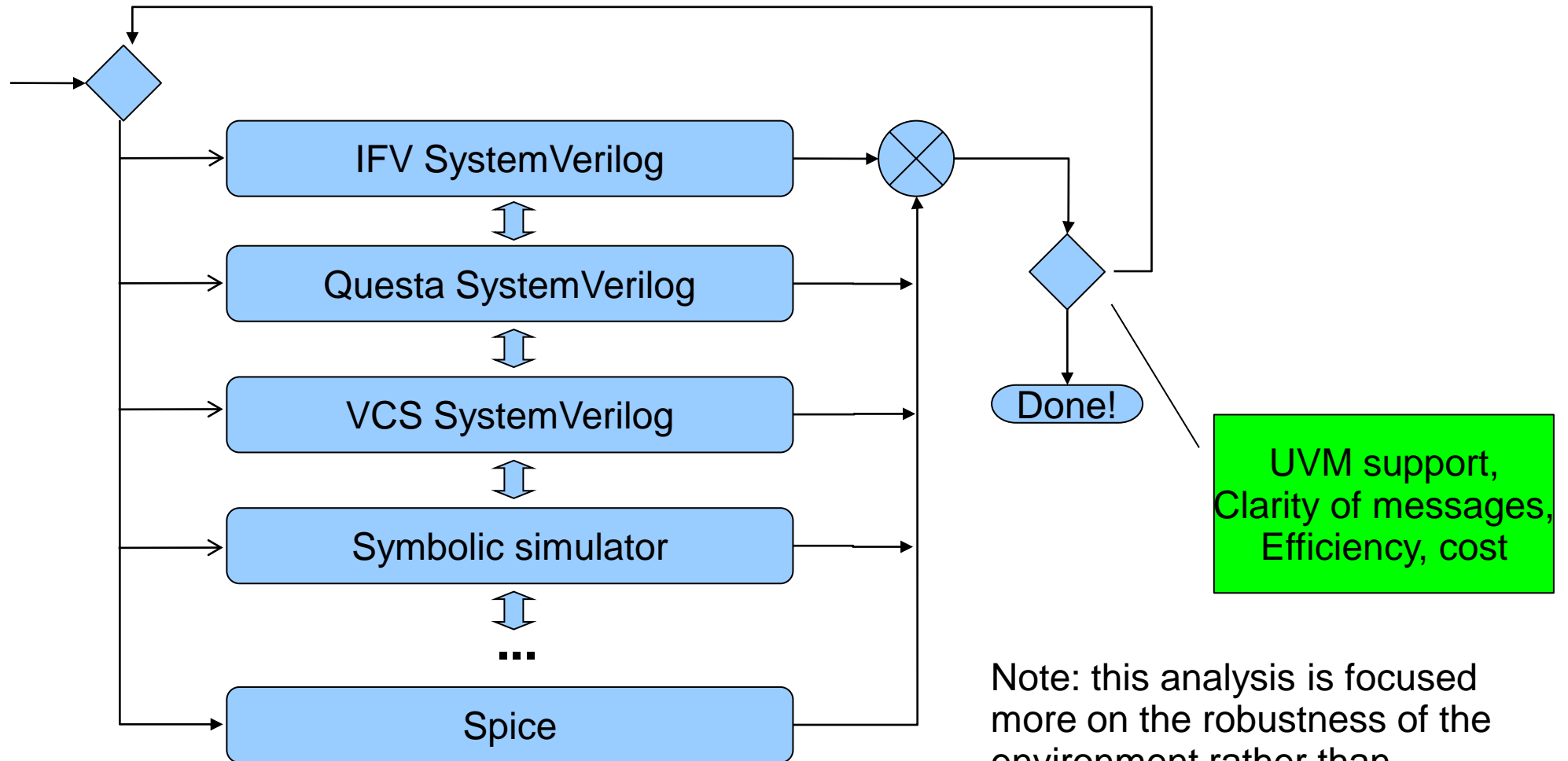
Finding bugs is a problem throughout the product lifetime, methods finding bugs are quickly noticed, disaster is always looming.

Comparison of source and functional coverage, joint code/coverage reviews, formal reachability analysis don't require 100% coverage..

Infrastructure and skills either not created or going stale limit adaptability.

Example: should multiple simulators be used?

# Use of multiple simulators



Note: this analysis is focused more on the robustness of the environment rather than efficiency in finding bugs.

# Choice of multiple simulators

- ✓ **Identify population**
- ✓ **Identify selection criteria**
- ✓ **Analyze impact of variety**
  - Exploitation
    - ✓ Best solution?
    - Stable problem?
  - ✓ Exploration
    - ✓ Long term problem?
    - ✓ Fast, reliable feedback?
    - ✓ Looming disaster?
- ✓ **Analyze interaction benefits**
- ✓ **Analyze extinction risk**

One simulator may be more efficient for some classes of problem, but if new language features are being added (e.g., SVTB) the problem may not be stable.

The feature implementations may not be resolved by the end of the project, feedback is nearly immediate, and disaster is always looming.

Cross check LRM conformance, clarity of error messages, identifying race conditions.

Alternatives not available in case of implementation show stopper or vendor switch.

# Other verification examples

- emulation vs. sim farm
- rerunning unit tests vs. system tests
- static analysis vs. simulation
- code reviews vs. writing assertions

# Key ideas

- We need to consider exploitation vs. exploration trade-offs in verification planning and execution.
- Variety from exploration provides more robustness and is more easily adapted to changing requirements.
- Plan for interaction between strategies to achieve better overall results.

# Further info

There are many different definitions of complex systems, but the ideas for this talk were based on the definition of Complex Adaptive Systems found in this book:

**Harnessing Complexity: Organizational Implications of a Scientific Frontier** (2001) by Robert Axelrod, Michael D Cohen ([amazon link](#))