2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA
MARCH 4-7, 2024

Next-Generation Formal Property Verification: Lightweight Theorem Proving Integrated into Model Checking

Erik Seligman, *Cadence Design Systems*, seligman@cadence.com
Karthik Baddam, *Qualcomm*, kbaddam@qti.qualcomm.com
Barbara Leite Almeida, Thamara Andrade, Poliana Bueno, Carla Ferreira, Matheus Fonesca, Lars Lundgren, Raquel Lara dos Santos Pereira, Fabiano Peixoto, Vincent Reynolds, *Cadence Design Systems*

accellera SYSTEMS INITIATIVE

Agenda

- Motivation & background
- Key Proof Structure Concepts
- Success At Qualcomm
- Summary

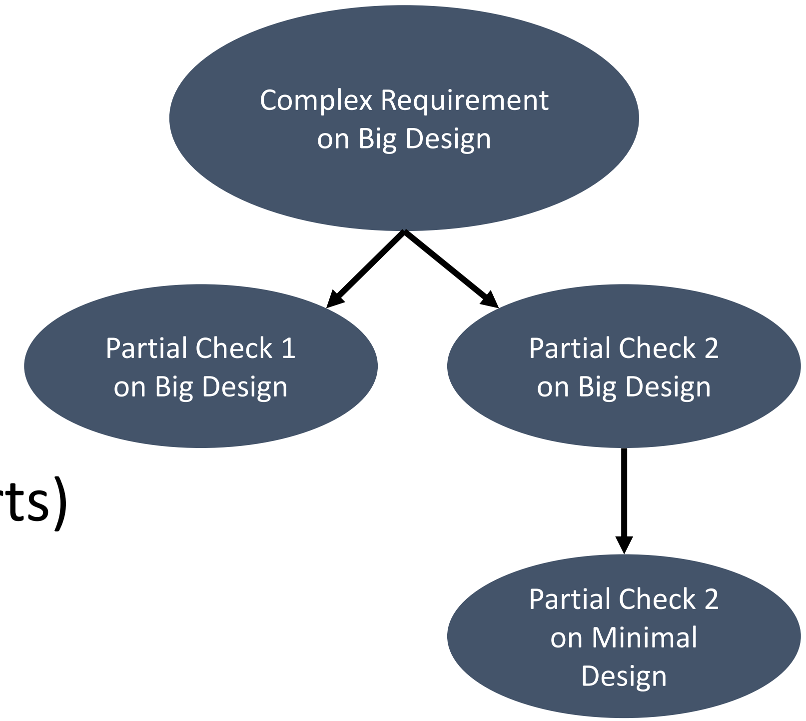
Agenda

- Motivation & background
- Key Proof Structure Concepts
- Success At Qualcomm
- Summary

Background – Proof Convergence Using Decomposition

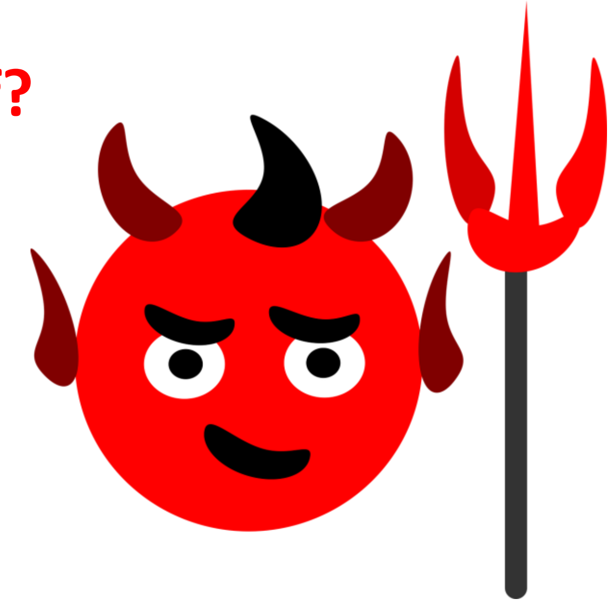
- FV experts decompose problem with advanced flows
  - Multi-step, variety of operations
  - Sub-step results together make overall proof
  - Use large tcl script running many tasks
- Examples of common techniques
  - Partitioning (different proof options for subgroups of asserts)
  - Case splitting
  - Stopats
  - Assume-Guarantee (Helper Assertions)
- As tool capacity grows, so do user ambitions

  ➔ Engine improvements will never replace these methods!



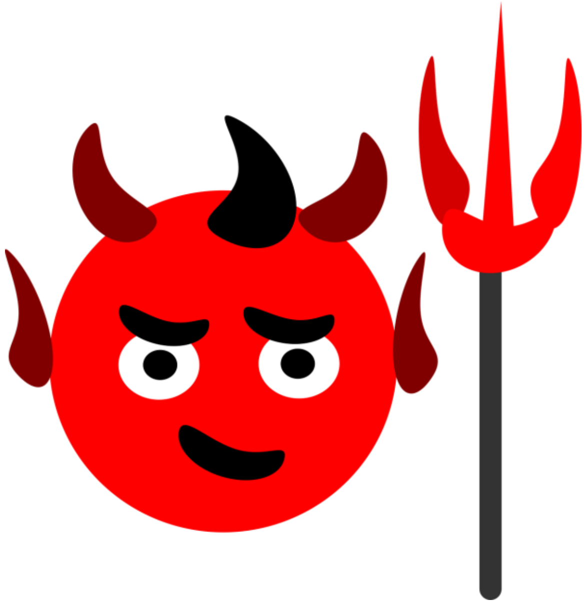
Decomposition in Jasper: The Problem

- Traditionally done with user-developed TCL
  - Large, complex, linear scripts
  - Multiple “tasks” (subprocesses) for decomposed problem
    - Each contains many individual assertion proofs
  - Users need to manage tasks and configurations
- **Does the combination of tasks compose a valid overall proof?**
- **Is this decomposition correct?**
  - **… now?**
  - **… after future design changes?**
  - **… when inherited by next project?**
- **Have I proven the right set of properties?**



Proof Decomposition By Hand: Some Common Mistakes

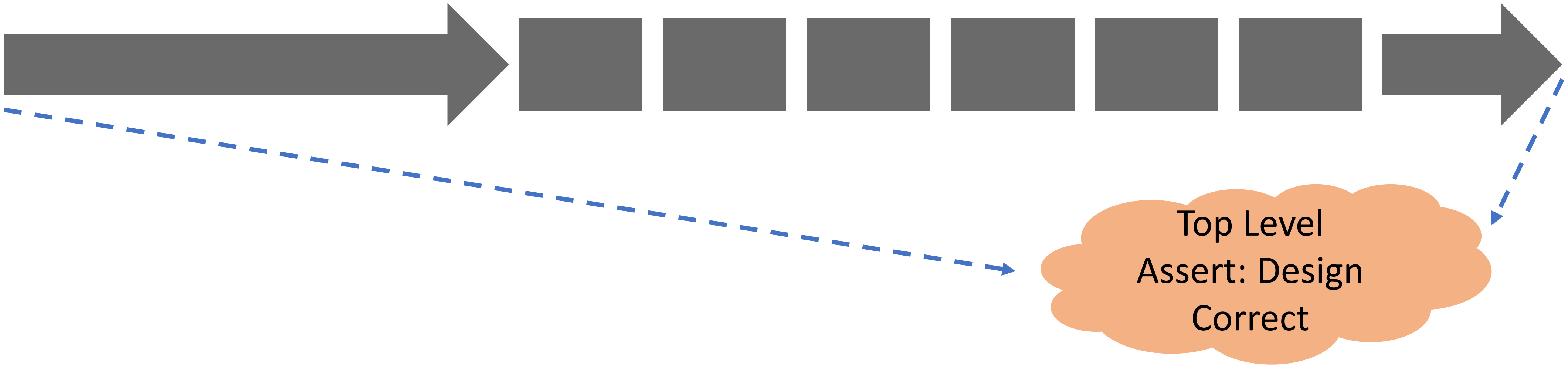
- **Ignored Properties**
  - Tasks divide properties into groups, miss new ones in RTL
- **Assume Guarantee Without Guarantee**
  - Naïve user completes “full proof” on task that assumes a helper…
  - Without checking that the helper was proven in another task
- **Incomplete Case Split**
  - Tasks created for each possible opcode, but one is overlooked
- **Wrong Composition of Bounds**
  - Main task in Assume Guarantee gets “full proof” of key targets…
  - But key helper only had bounded proof, so should consider all AG proofs bounded



Proof Structure Vision

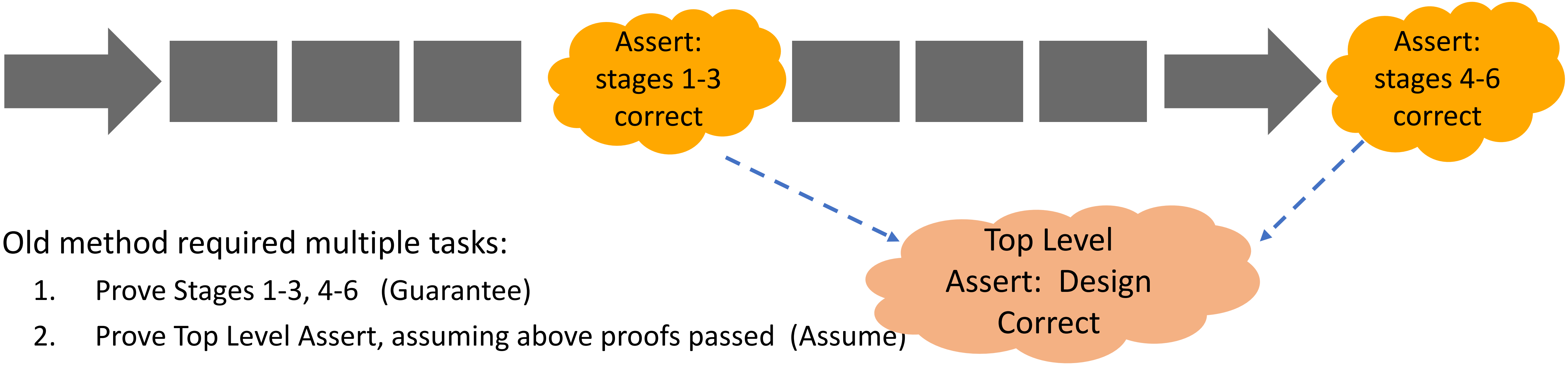
Usage Element	Traditional Model Checking	Model Checking with Proof Structure
What to prove	Individual Properties	Properties + Composition Rules
Multiple proof tasks	User controlled "bag of tasks" No defined relations between tasks	"Nodes" (Proof Structure tasks) with provable logical relationships
Documentation of decomposition rules	Completely implicit in user scripts	Defined when nodes created Correct By Construction
Detecting logical gaps in decomposition	Need thorough manual review	Tool enforces overall correctness "Proven" report checks valid decomposition

Example: Proving A Pipeline



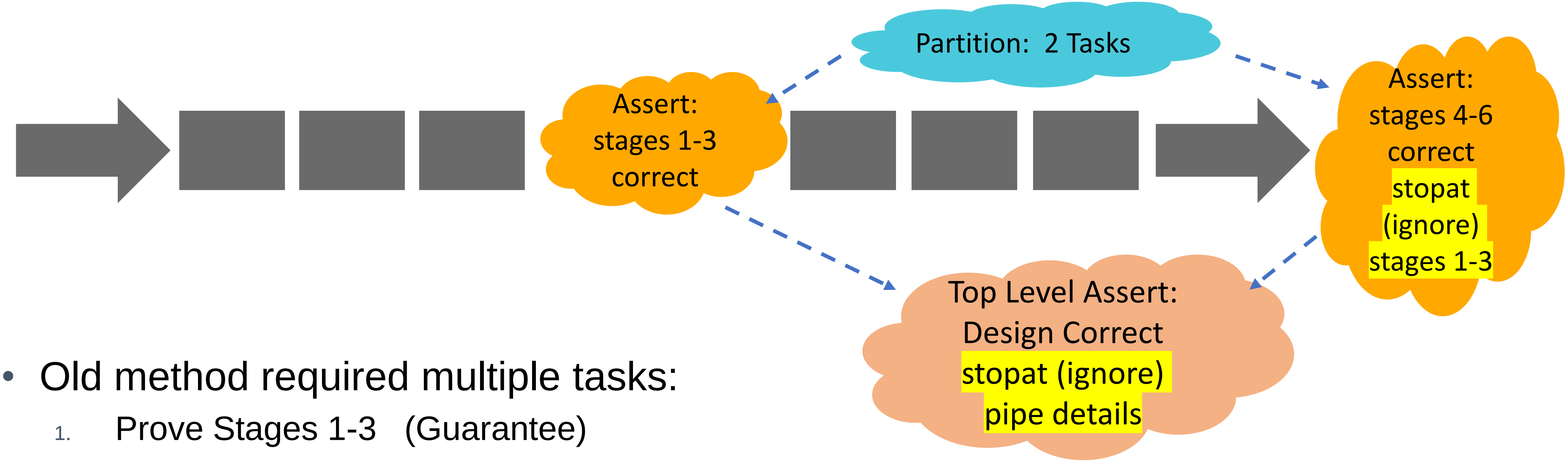
- Full pipeline too complex for proof

Assume-Guarantee: Proving Pipeline with "Helper" Assertions



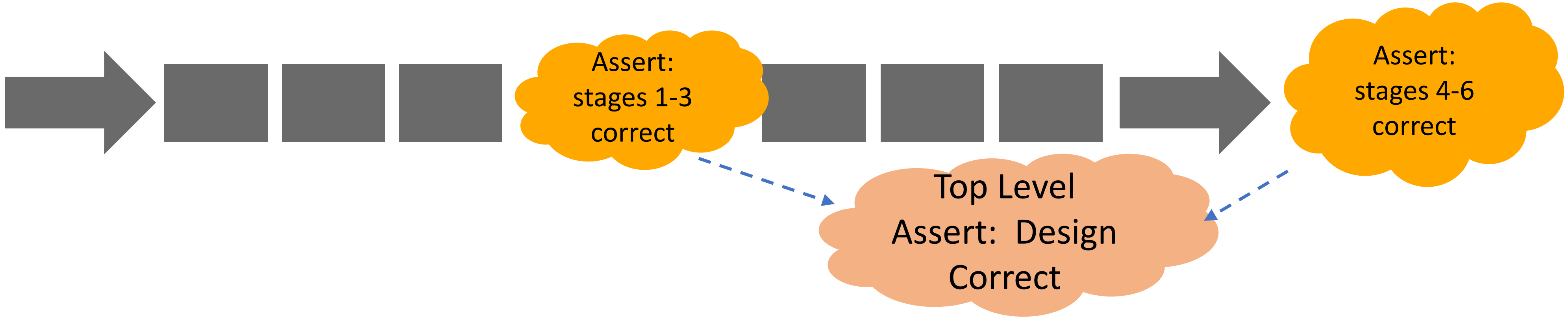
- Old method required multiple tasks:
  1. Prove Stages 1-3, 4-6 (Guarantee)
  2. Prove Top Level Assert, assuming above proofs passed (Assume)
- Lots of risks if users manually track (especially on less trivial cases!)
  - Were all helpers proven in some task?
  - Does sum total of tasks == fully sound proof?
  - Were any top level properties lost during task setup?

Assume-Guarantee: Proving Pipeline with "Helper" Assertions



- Old method required multiple tasks:
  1. Prove Stages 1-3 (Guarantee)
  2. Prove Stages 4-6 (Guarantee) + simplify with stopat
  3. Prove Top Level Assert, assuming above proofs passed (Assume) + simplify with stopat
- Lots of risks if you manually track (especially on less trivial cases!)

Solution: Proof Structure



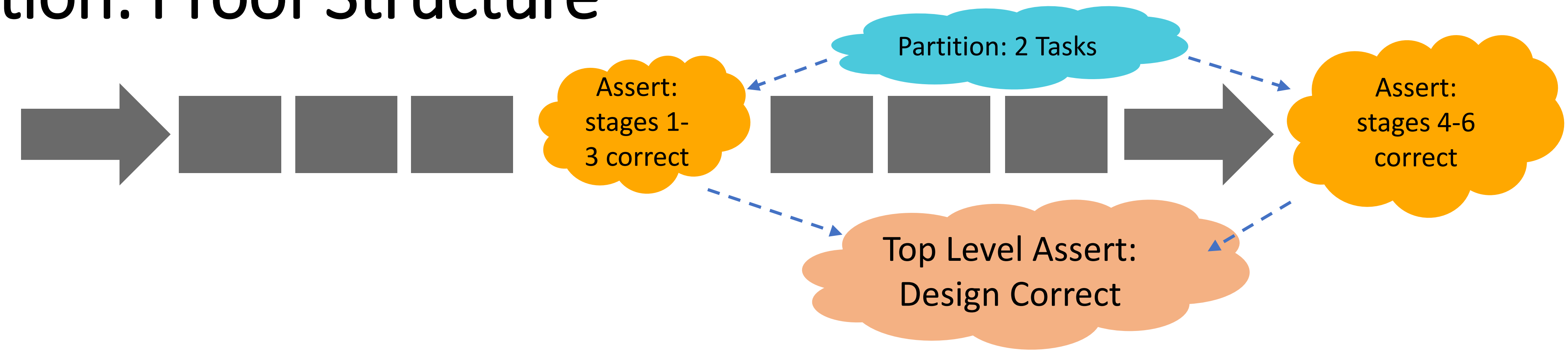
Proof Structure

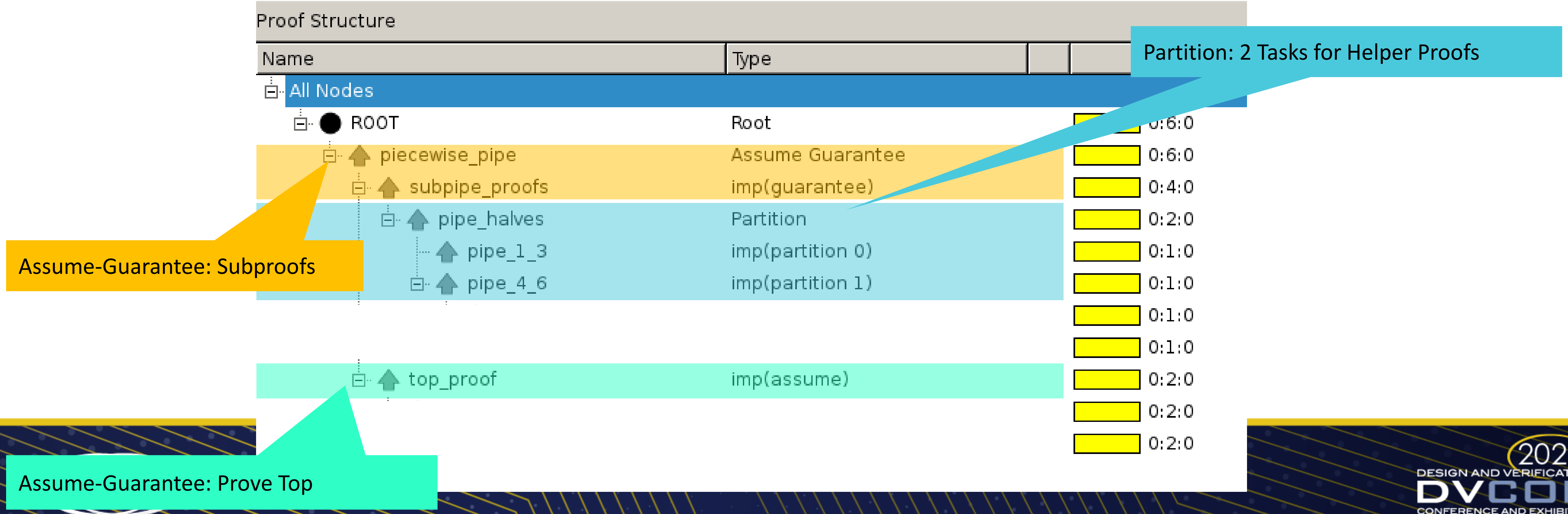
Name	Type			PS Result
All Nodes				
ROOT	Root			0:6:0
piecewise_pipe	Assume Guarantee			0:6:0
subpipe_proofs	imp(guarantee)			0:4:0
				0:2:0
				0:1:0
				0:1:0
				0:1:0
				0:1:0
top_proof	imp(assume)			0:2:0
				0:2:0
				0:2:0

Assume-Guarantee: Subproofs

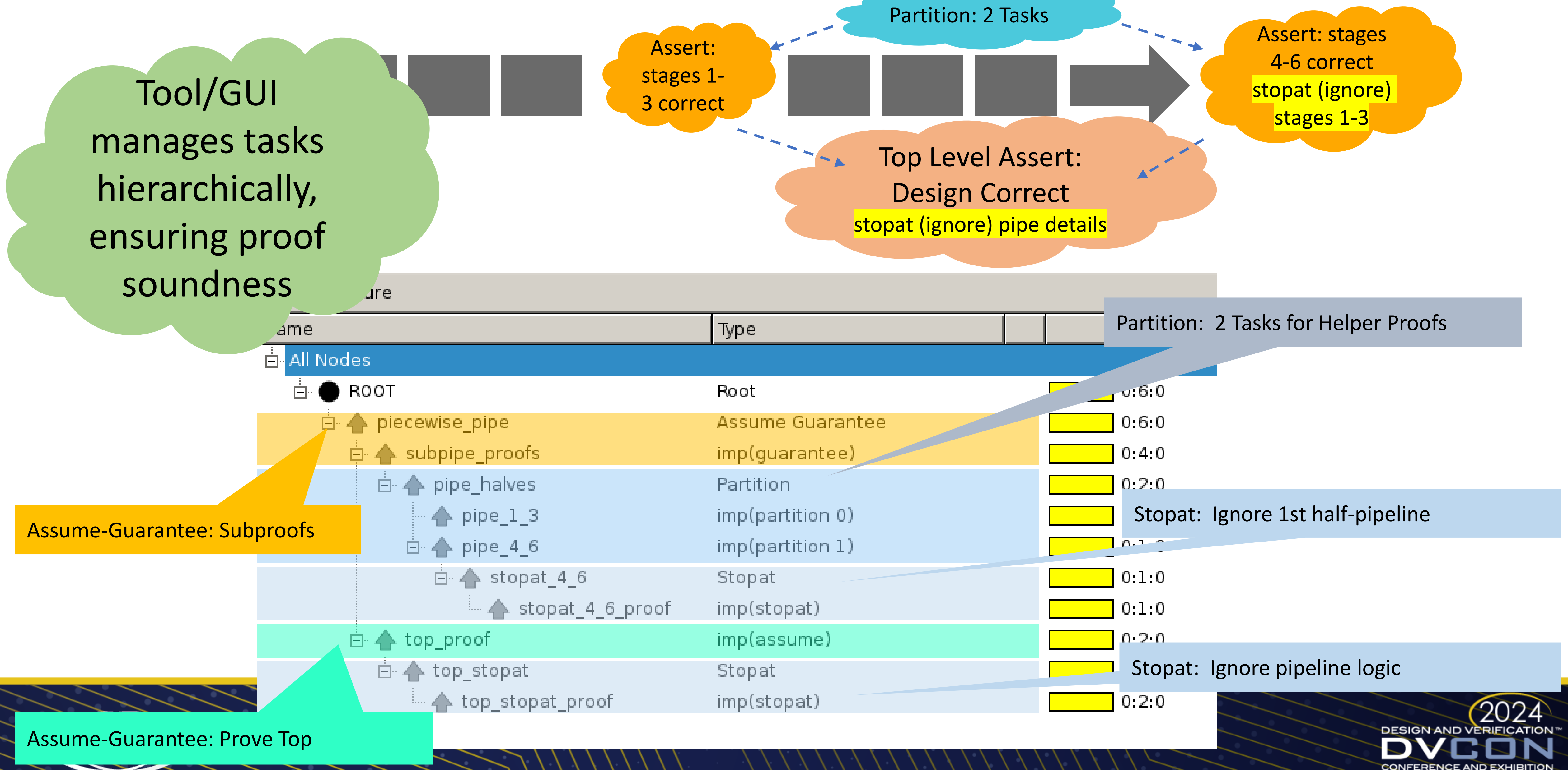
Assume-Guarantee: Prove Top

Solution: Proof Structure





Solution: Proof Structure



Key Benefits of Using Proof Structure

Benefit	Details
Well-Defined Environments	New node env == derived from decomposition rule used
Correct By Construction	Proofs propagate up tree based on sound rules
Correct Bounded Propagation	Bounds for proofs include all dependent proofs
Visible Proof Strategy	Proof Structure tree fully defines the decomposition
Enable Parallelism	Nodes can execute independently- results merged later
Enable Complex Strategies	Some strategies hard to script but easier in Proof Structure

Agenda

- Motivation & background
- **Key Proof Structure Concepts**
- Success At Qualcomm
- Summary

Old Methods vs Proof Structure

Custom scripts to manage tasks

Task Tree

Name		Result
All Tasks		
<embedded>		0:2:0
SETUP		0:6:0
A1		1:0:0
G1		0:2:0
GET_NEEDED_ASSUMPTIONS		0:2:0
A2		1:5:0
XPROP		0:6:0
G2		6:0:0

Design Hierarchy | Task Tree | Proof Structure

Traditional use model:
Flat task-based solution
Ad hoc script judges soundness

Proof Structure:
Tool understands decomposition
Tool knows how nodes relate

Proof Structure

Name	Type		PS Result	Propagation
All Nodes				
ROOT	Root		1:5:0	None
AG1	Assume Gu...		1:5:0	All
AG1.G	imp(guaran...		1:4:0	All
AG2	Assume Gu...		1:4:0	All
AG2.G	imp(guaran...		1:3:0	All
AG2.A	imp(assume)		0:1:0	Only Cex+Covered
AG1.A	imp(assume)		1:0:0	Only Cex+Covered

Design Hierarchy | Task Tree | Proof Structure

Core Proof Structure Backbone: Well-Controlled Nodes

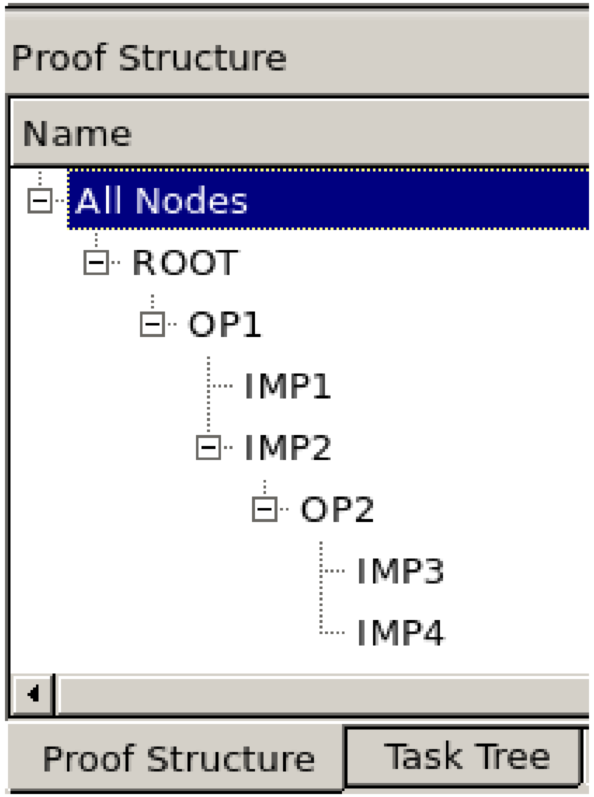
- Proof Structure node like traditional task…
  - BUT with limitations to enforce soundness
  - No manual cut points, abstractions, etc
- Any environment change is well-defined
  - Inherited from source task when node created
  - Proof Structure modifies based on node type

➔Proof Structure ***understands*** parent-child relationship

    ➔& can enforce correct propagation

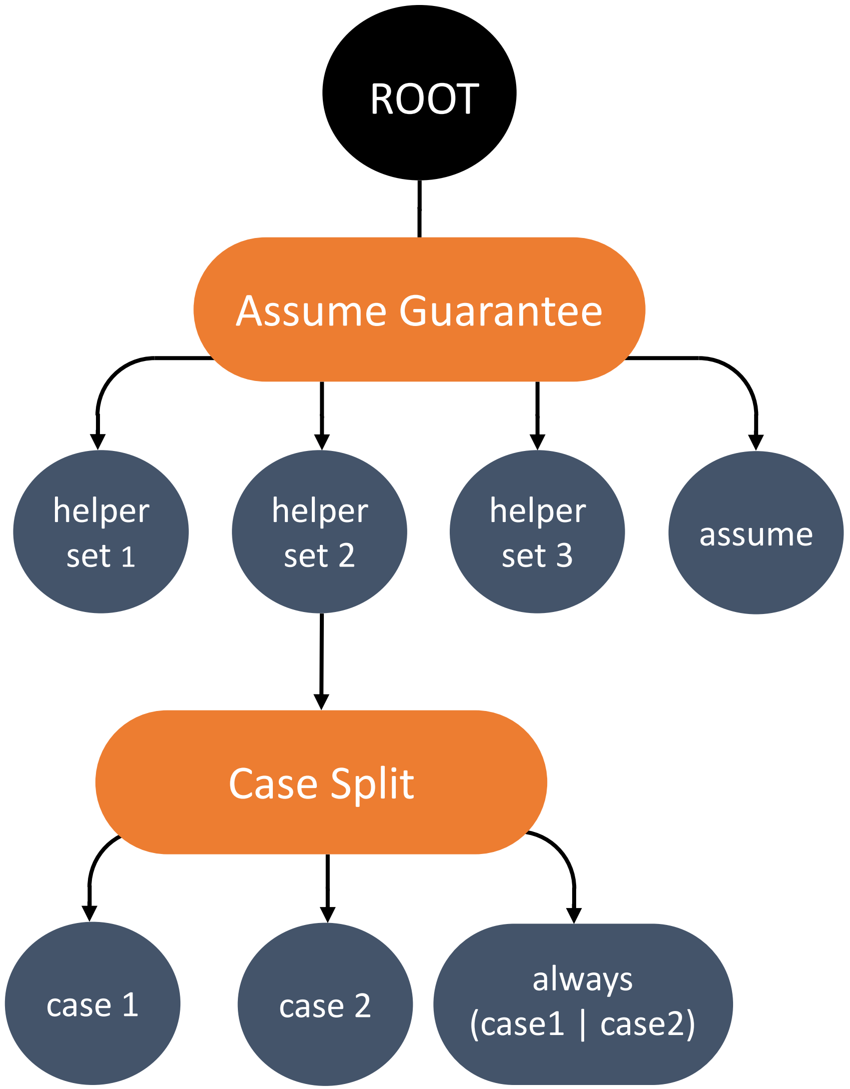
What is Proof Structure?

- Proof flows are implemented using '**operations**'
  - Operation == pre-defined decomposition step
  - Each operation has one or more 'implementation nodes'
  - Examples: Assume Guarantee, Case Split, Partition
- Multiple operations iteratively refine a proof
  - Cascaded operations form a hierarchy
  - Results ***propagate*** from lower levels to higher levels
    - Propagate == decide how subtree proofs travel up tree
    - Propagation based on soundness rules
    - Propagate ***full proofs***, ***bounded proofs***, and ***counterexamples (cex)***



What is an operation?

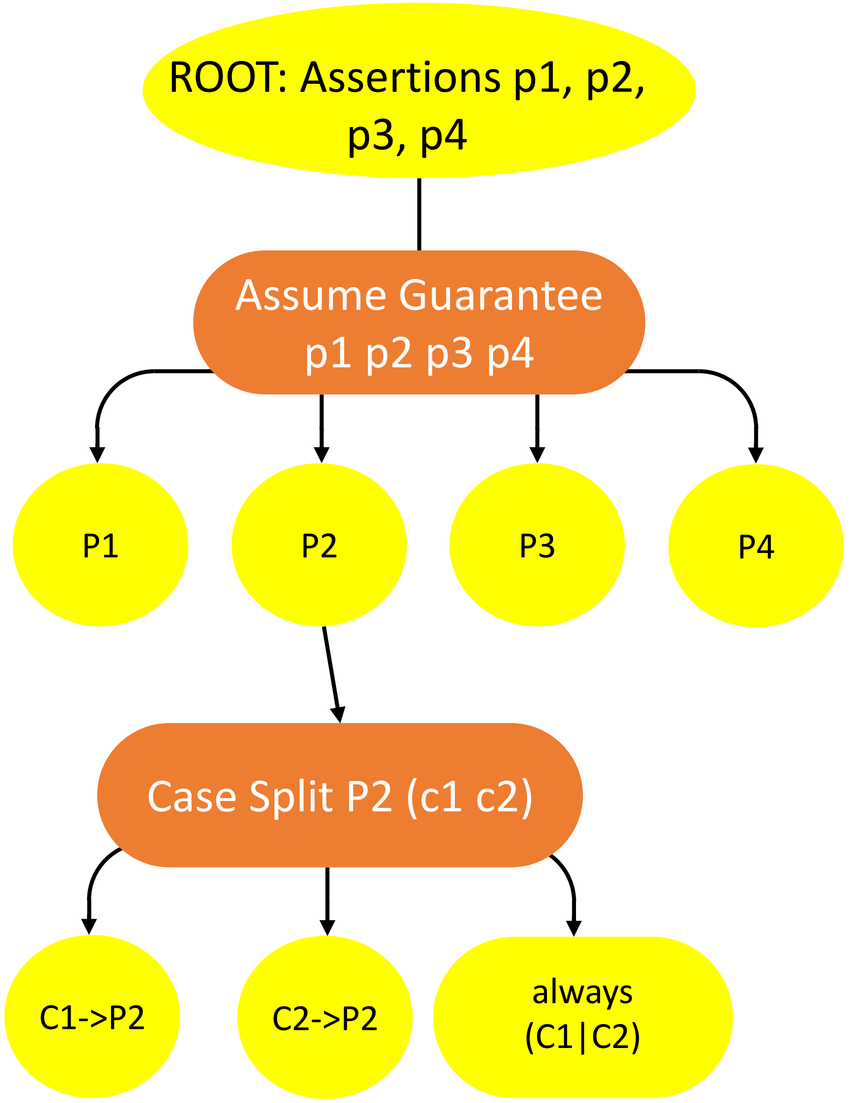
- A specific step in the proof flow
  - Cascading operations == successive refinement
  - Results propagate up the chain
- **Operation nodes** handle proof propagation
  - Operation type defines propagation rules
- ***Implementation nodes*** provide proof targets
  - ➔One or more ***implementation nodes*** per operation
  - ➔Self-documenting, no hidden steps
  - ➔***Assertion proofs run on implementation nodes***



Proof With Propagation Example
Step 1: Setup

- Assume Guarantee helper chain
  - P1 -> P2 -> P3 -> P4
  - Each node to help proofs to right
    - P2 proof assumes P1
    - P3 proof assumes P1, P2
    - P4 proof assumes P1, P2, P3

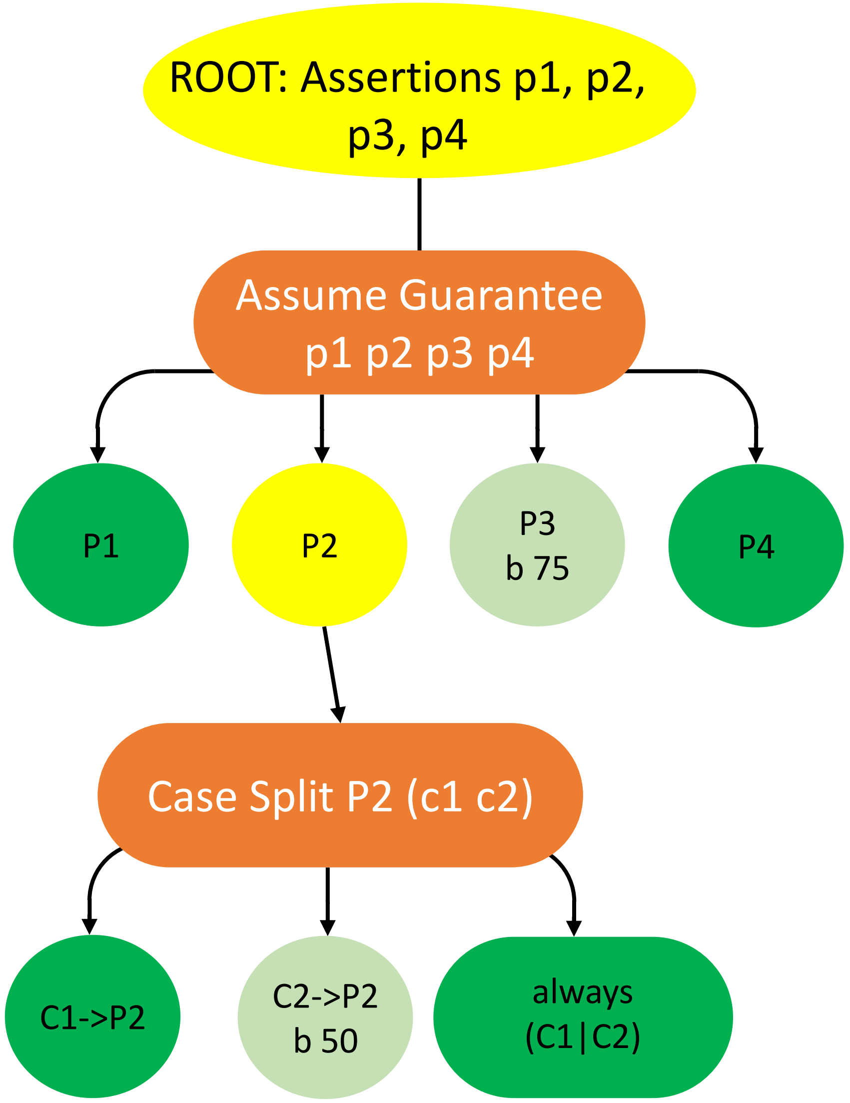
- Case Split to solve challenging assertion P2
  - Separate proof for cases C1 and C2



Proof With Propagation Example

Step 2: Leaf Level Proofs

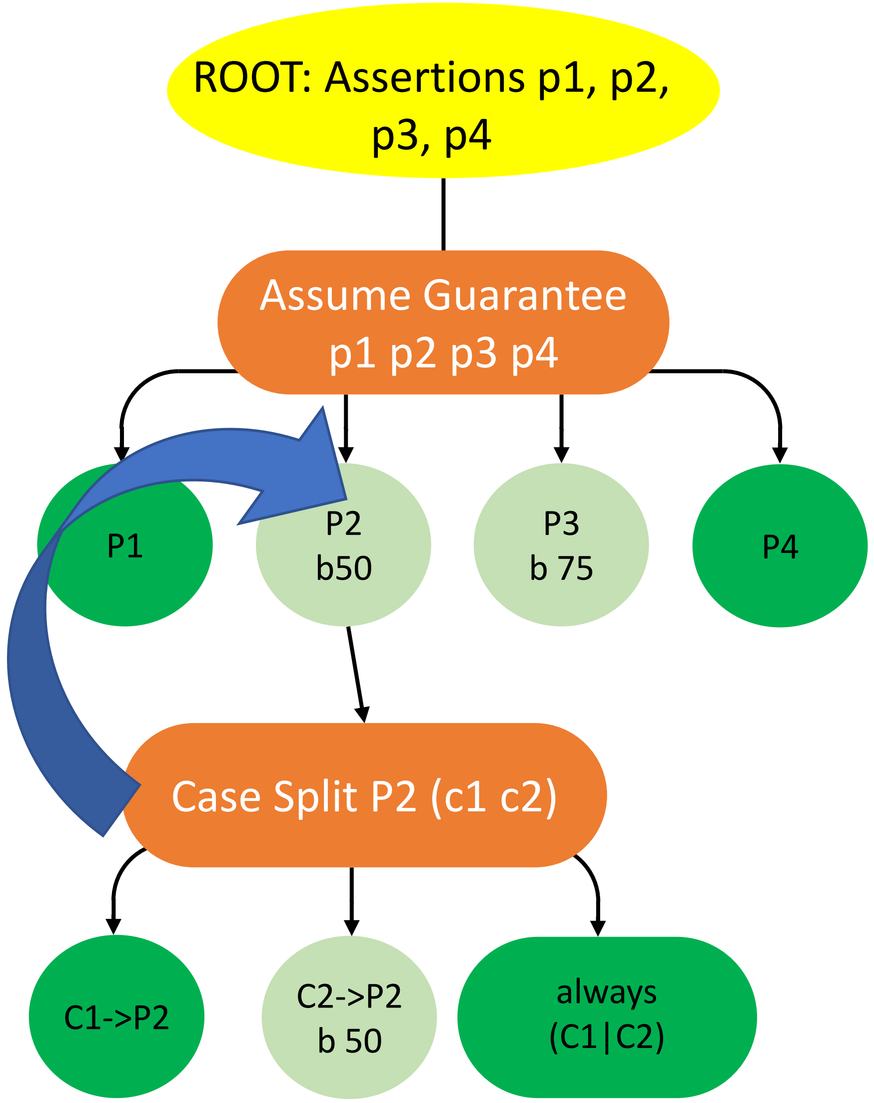
- All these proofs can run in parallel
- P1, P4 fully proven
- P3 proves to bound 75
- Case C1, Completeness proven
- Case C2 proves to bound 50



Proof With Propagation Example

Step 3: Layer 1 propagation

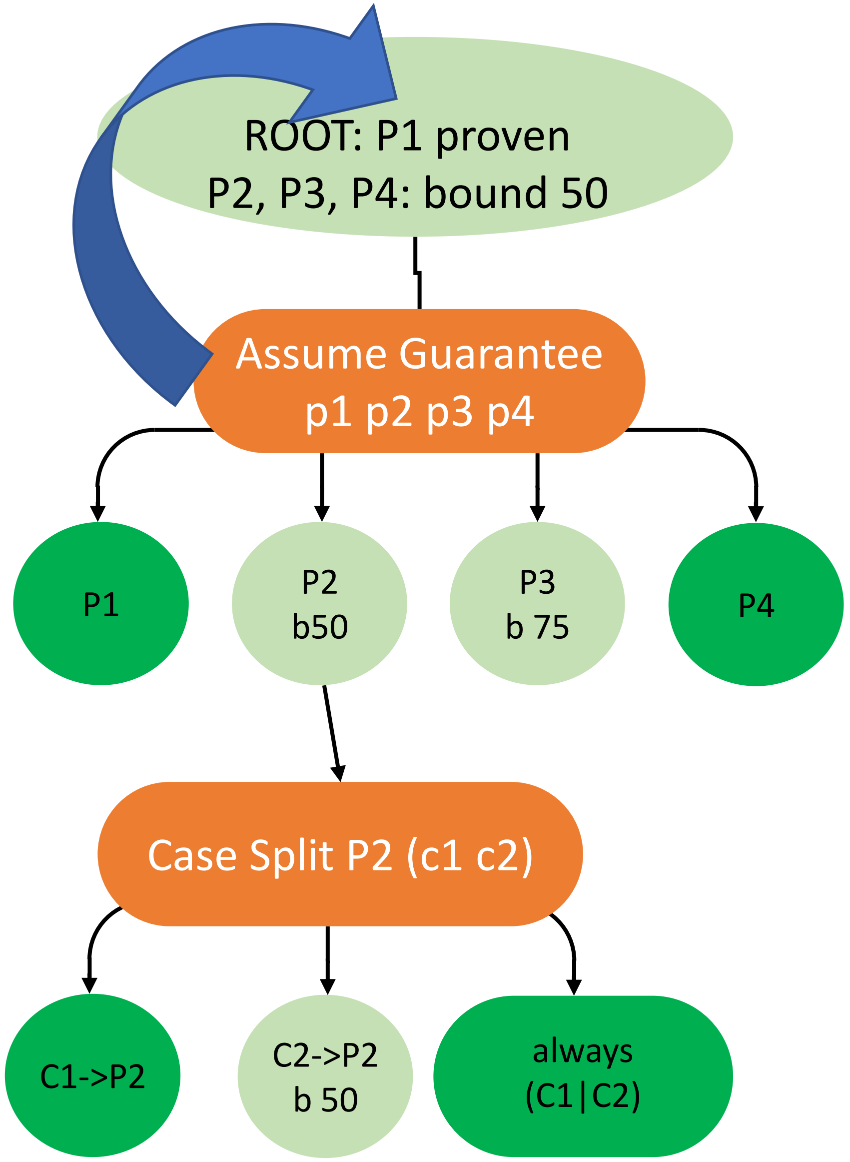
- Overall Case Split proof of P2 is bounded at 50
  - Due to worst case C2
  - Thus Bound 50 proof propagated up for P2



Proof With Propagation Example

Step 4: Complete Propagation

- Assume Guarantee propagation
  - P1 first in chain: added no assumes or helpers
    - So proof fully propagates
  - P2 inherits propagated bound 50
    - With fully proven P1 as helper
  - P3, P4 only propagate bound 50
    - Since P2 is helper for both, with bound only 50
    - Local proofs of P3, P4 only valid to that bound



Major Operations In Current Implementation

- **Underconstrain (Stopat)**
- **Overconstrain**
- **Partition**
- **Case Split (Soft and Hard)**
- **Assume Guarantee**
- **Compositional Assume Guarantee**

Main Propagation Rules & Hazard Prevention

Operation	Propagation	Automatic Supplemental Proofs
Underconstrain (Stopat)	Proofs, but not cex	-
Overconstrain	Cex, but not proofs	-
Partition	Both proofs and cex	No property missed
Case Split	Cex, and proofs if all cases + auto checks pass	Completeness, Validity
Assume Guarantee	Cex, and proofs if relevant helpers pass	-
Compositional Assume Guarantee	Cex, and proofs if all mutual helpers pass	-

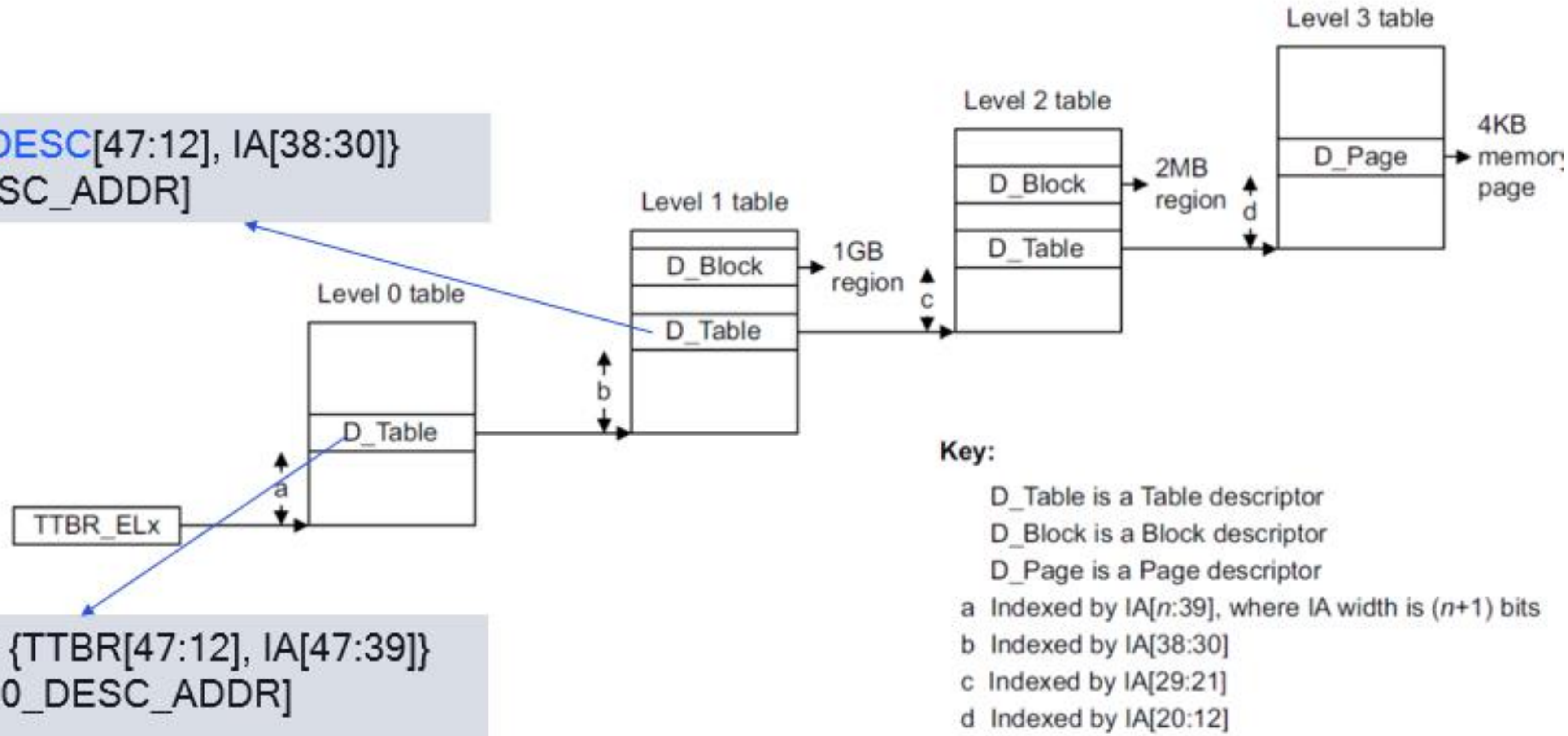
Agenda

- Motivation & background
- Key Proof Structure Concepts
- **Success At Qualcomm**
- Summary

Qualcomm Table Walk Design

Natural Fit for Assume Guarantee

L1_DESC_ADDR = {L0_DESC[47:12], IA[38:30]}
L1_DESC = MEM[L1_DESC_ADDR]



L0_DESC_ADDR = {TTBR[47:12], IA[47:39]}
L0_DESC = MEM[L0_DESC_ADDR]

Qualcomm Table Walk
Before and After Proof Structure

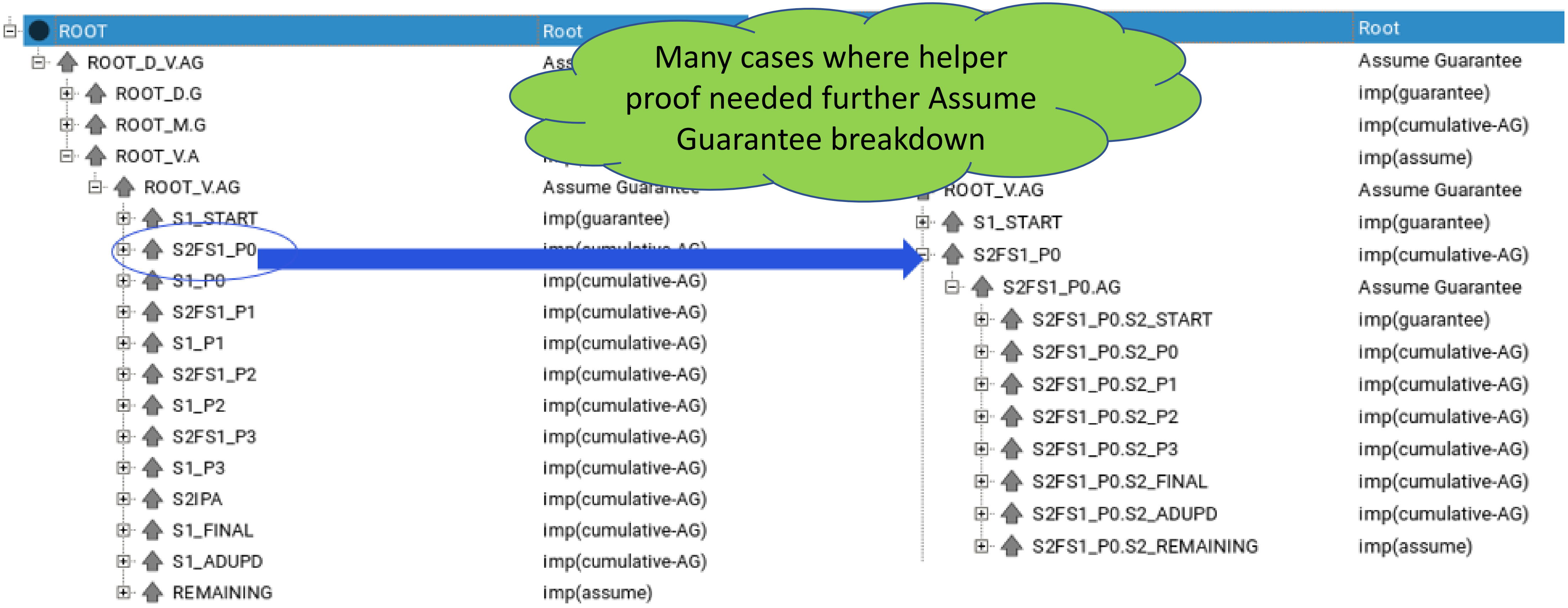
- Before: Traditional FPV
  - 50+ tasks, very complex to manage
  - User afraid of proof gaps, so serialized dependent proofs
  - Multiple days for each proof run
  - Hard to experiment with alternate strategies for subproofs

Qualcomm Table Walk
Before and After Proof Structure

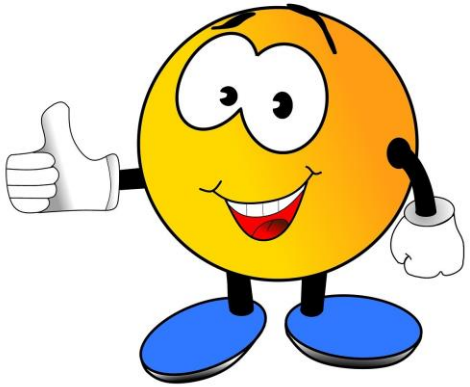
- Before: Traditional FPV
  - 50+ tasks, very complex to manage
  - User afraid of proof gaps, so serialized dependent proofs
  - Multiple days for each proof run
  - Hard to experiment with alternate strategies for subproofs

- After Proof Structure: Success!
  - Organized tracking of Assume Guarantee chains
  - Leveraged Compositional Assume Guarantee in some areas
  - Full proof run completes in less than a day
  - **Found 7 high-quality logic bugs**
    - Including one missed for 2 years by all validation flows

Qualcomm Table Walk
Proof Structure Snapshot



Other Proof Structure Customer Success



- **Qualcomm:** Nitish Sharma presenting at this conference
  - Just before this talk!
  - Significantly **improved productivity** for complex decomposition
- **Marvell:** DAC 2023 presentation on Proof Structure Case Split
  - Complex environment with 10,000+ assertions
  - Estimate: **saved several weeks of proof setup / scripting effort**
  - + Caught proof setup scripting bugs – likely escapes otherwise
- **HPE:** Jasper User Group 2022 presentation
  - Multi-layered example with Assume Guarantee, Partitions, & Underconstraints
  - Logistics enabled much more easily with Proof Structure
  - **Found 15 design bugs**, including 2 missed by simulation/emulation for long time

Agenda

- Motivation & background
- Key Proof Structure Concepts
- Success At Qualcomm
- Summary

Summary: Why use Proof Structure?

- **Organization**
  - Decomposition tree is clear, not implied by 1000 line script
  - Correct by construction
- **Sound Proof Reasoning**
  - Environments are consistent
  - Automatic correctness proofs
  - Bounds correctly integrated
- **Maintainability**
  - New owner can easily understand strategy
  - Propagation automatically blocks no-longer-valid proofs
- **Parallelism**
  - All leaf nodes have sufficient context to run independently
- **Enable Complex Strategies**
  - Complex multi-layer strategies now more feasible to implement efficiently

Key Lessons of Proof Structure

- Decomposition is here to stay
  - Tools get more powerful…
  - … but users throw bigger problems at them!
- Huge opportunity: support safe decomposition
  - Historical view: domain of user-level scripting
    - Many chances for errors in custom scripts
    - Very dangerous for long-term maintainability
  - Proof Structure has shown that we can do better
    - Leverage ideas from Theorem Proving
    - Tool reduces burden/risk of user scripts

Don't Just Verify Properties, Verify the Methodology!

Try Proof Structure Yourself in Cadence Jasper™ Tools

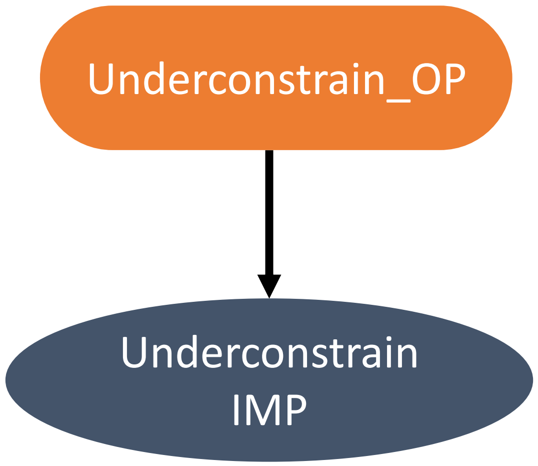
- "Rapid Adoption Kit" class/lab available at https://support.cadence.com
  - Search for "**Proof Structure**" in search box
- Take the exam & earn a badge!

Questions

Backup Slides

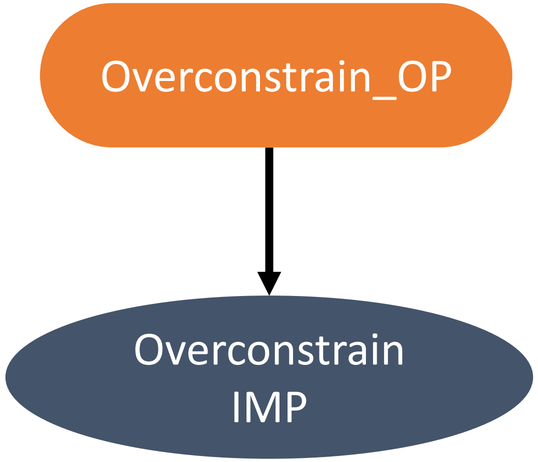
Underconstrain Operation

- Underconstrain = ignore some aspect of the logic
  - Cut Point / Stopat: Treat specific signal(s) as free input
  - Assumption Removal: Ignore some constraints
  - Abstraction: Substitute simpler logic
    - Ignore reset value / Counter can skip forward / etc.
- Very common technique- Proofs are fully valid
  - But counterexamples can’t be trusted
- Propagation rules
  - Counterexamples are untrustworthy– don’t propagate
  - Any proof is valid, can propagate
    - Including bounded proofs: propagate with same bound

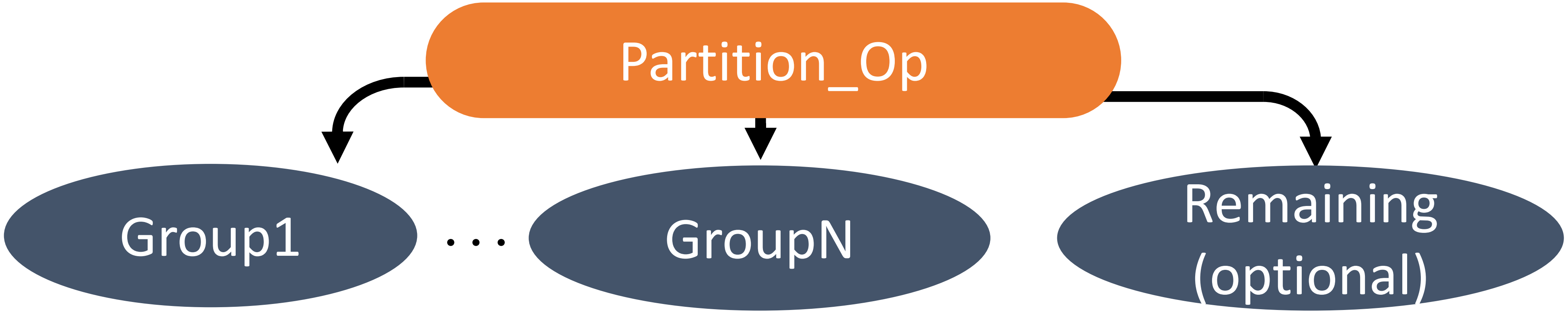


Overconstrain Operation

- Overconstrain = disallow some real behaviours
  - Add assumptions, or tie signal to constant
- Great for bug hunting- counterexamples are valid
  - But proofs can't be trusted
- Propagation rules
  - Counterexamples are valid, always propagate
  - Proofs are not valid, never propagate
    - Same is true of bounded proofs



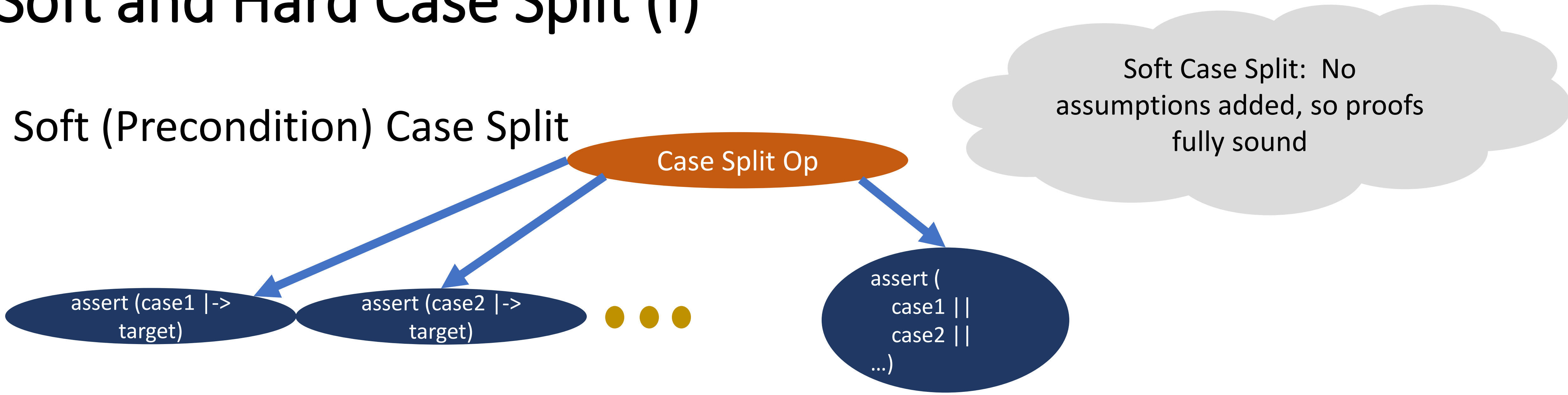
Partition: Divide Properties Into Groups



- Simple operation, but important benefits
  - Built-in checks: duplication, missing properties
  - Option to auto-group missing properties in default node
  - Enable using different strategies in later subtrees
- Propagation is easy: environment is unchanged from parent
  - ➔ proofs, counterexamples, and bounds always valid in any node

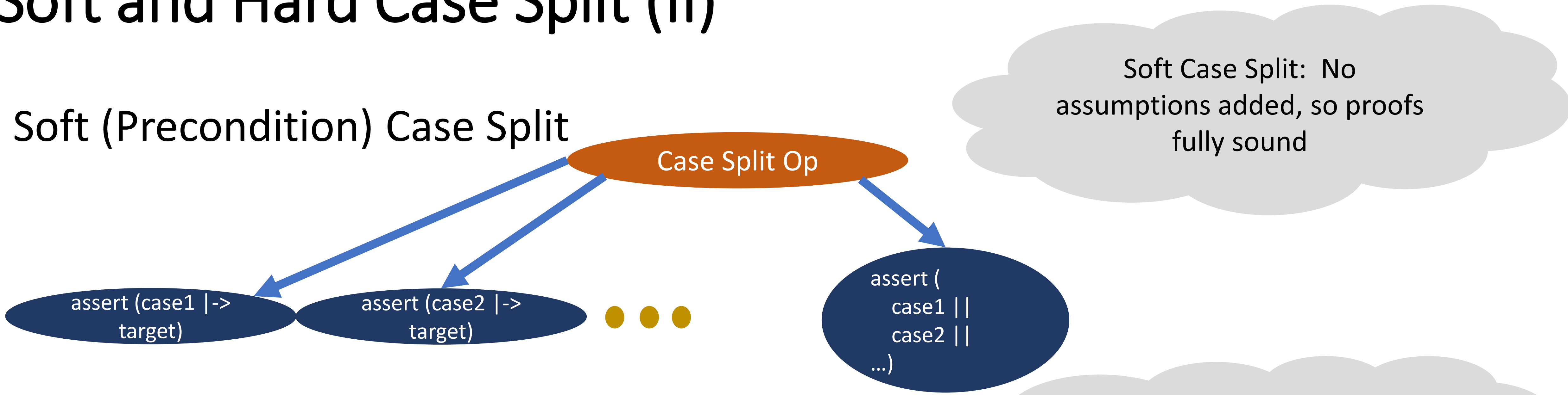
Soft and Hard Case Split (I)

- Soft (Precondition) Case Split

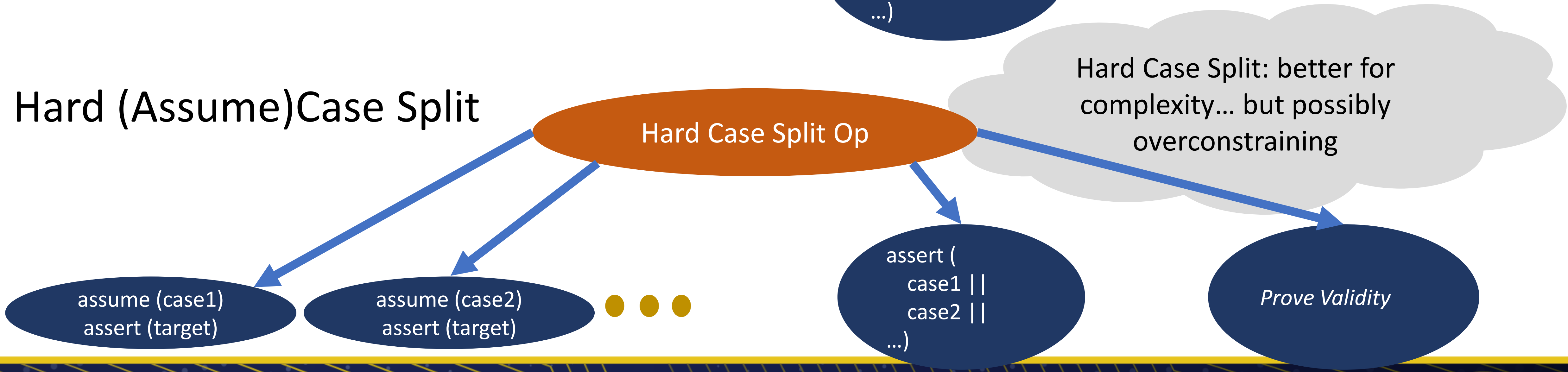


Soft and Hard Case Split (II)

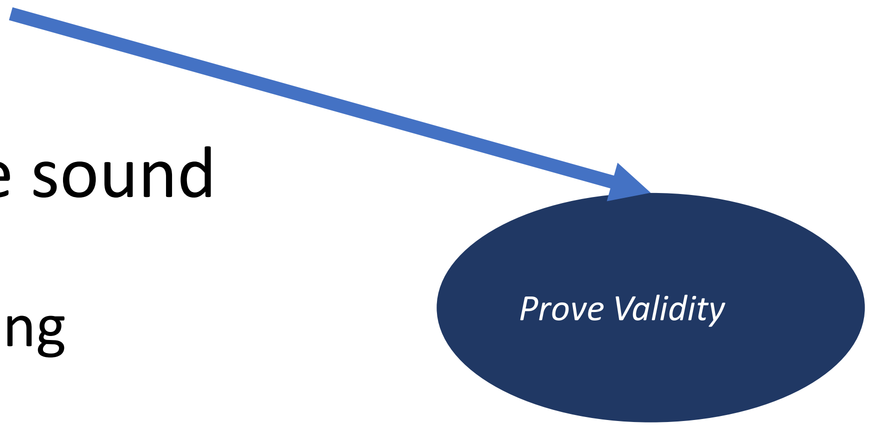
- Soft (Precondition) Case Split



- Hard (Assume)Case Split

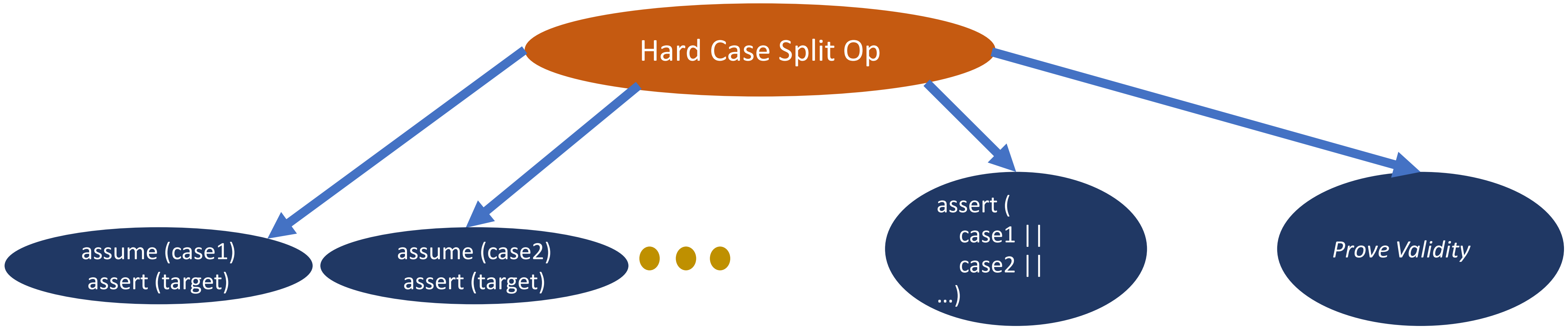


Hard Case Split Validity



- Since assumptions are added, proof may not be sound
  - ➔ Proof Structure adds a node to prove validity
  - Counterexamples are fully valid- great for bug hunting
- Validity types (selected when node created)
  - **Combinational**: Prove model contains no state elements
  - **Invariant**: Prove case is constant in model
    - Example: constant assumptions from FPV register setup
  - **Exhaustive**: Node assumes *(case1|case2|…),* proves *target*
    - Logically redundant with individual case nodes
    - Case nodes == faster bug hunting, get to good depths much more easily
    - Validity node == signoff-quality propagatable proof later in project

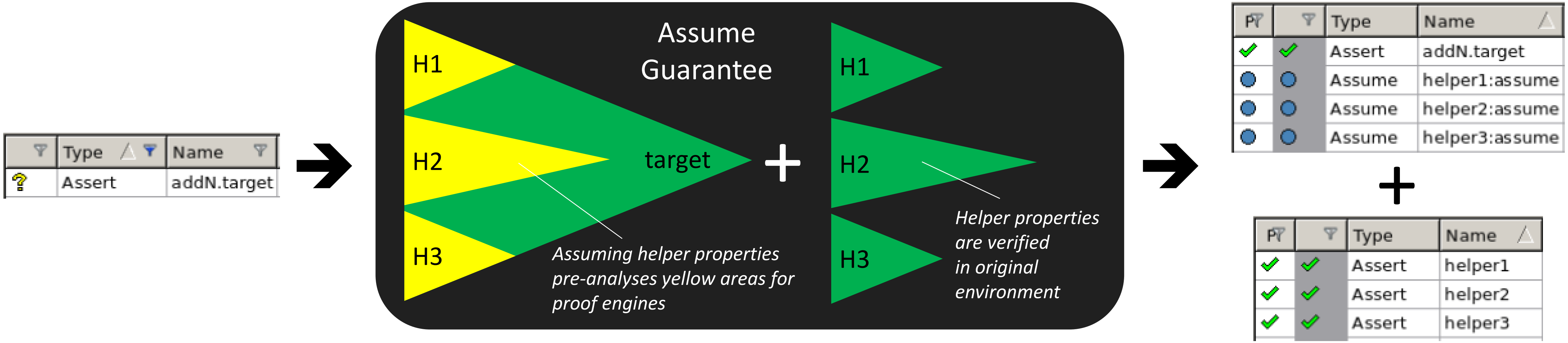
Case Split Propagation Rules



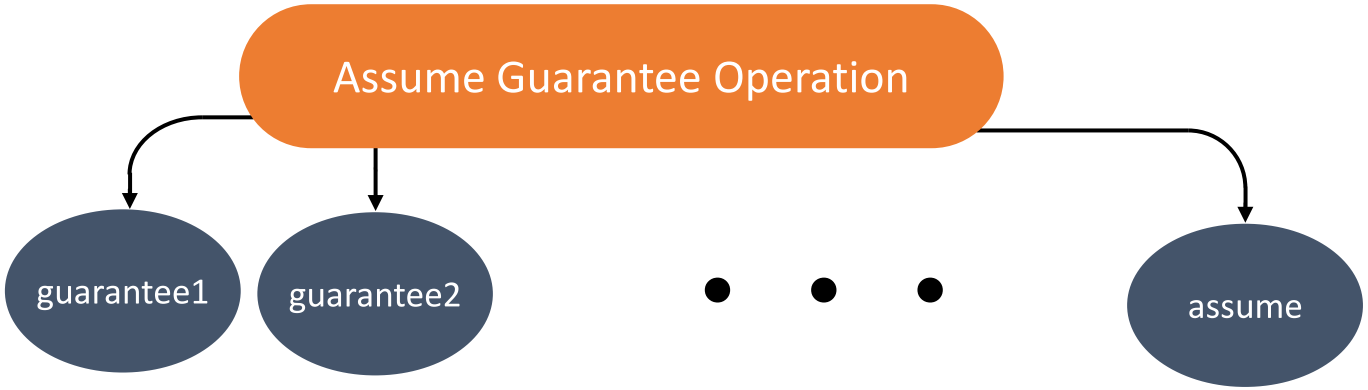
- Counterexamples always propagate from case nodes
  - If assertion is false for one case, it's false
  - From completeness/validity nodes, no propagation
    - Proof does not exist at parent
- Proofs need to integrate results from subnodes
  - Need all cases + completeness + (if hard case split) validity proofs to pass
  - Bounded proofs: min bound of {min case, completeness, validity} propagates

Assume Guarantee (“Helper Assertions”)

- Prove assertions earlier in fanin to aid target proofs
  - Simplify problem by pre-proving some logic ➔ faster convergence on target
  - Depends on helpers being valid- need to prove them
  - Can use cascading series of helper sets, or just one

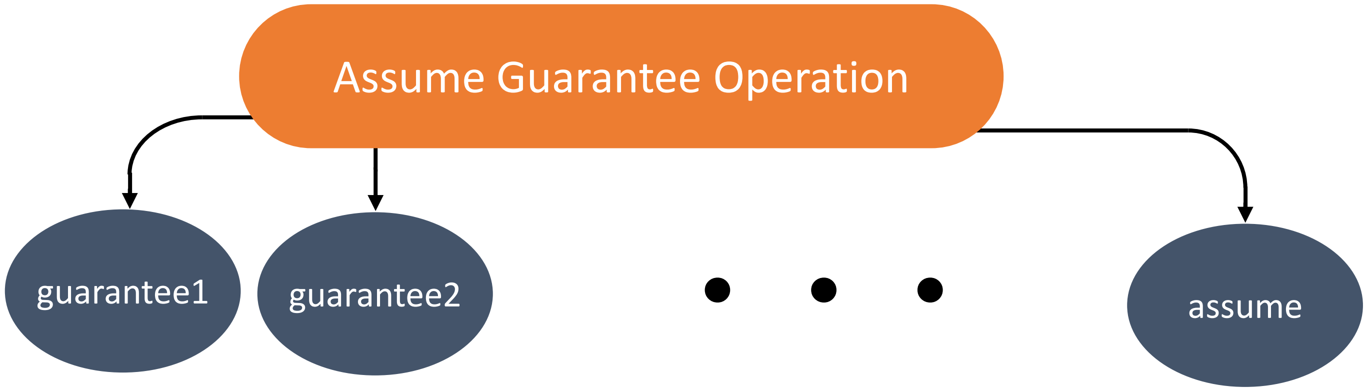


Assume Guarantee Operation in Proof Structure



- Declare set(s) of targets & order on node creation
- Properties in node <N> assume those in nodes <N-1:0>
  - Only leftmost group verified without additional assumptions

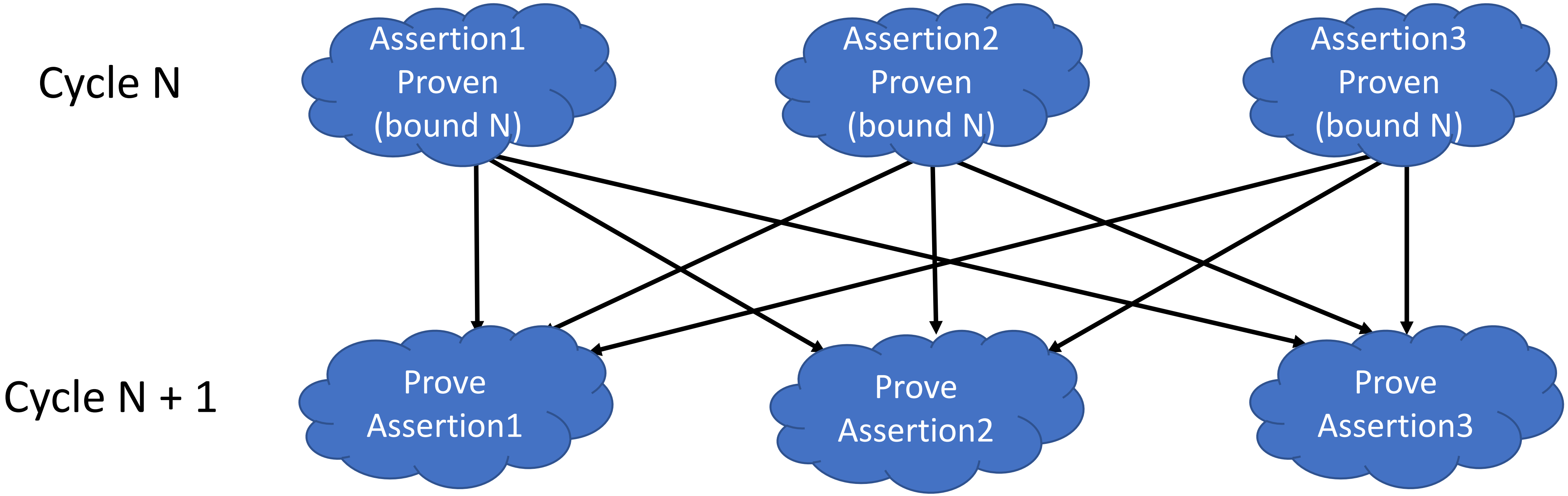
Assume Guarantee Propagation Rules



- Counterexamples from any node are always valid & propagated
- Proof in an imp node is propagated only if all nodes to left are proven
  - Except in leftmost node, where proofs always valid since no additional assumes
- Bounded proof = minimum bound of current node & all proofs to the left

Compositional Assume Guarantee (CAG): The Concept

- Use all assertions as helpers for each other
  - Inductive method – not circular reasoning!
    1. Prove all assertions true on cycle 1
    2. Prove: (all assertions true on cycle N) ➔ (each assertion true on cycle N+1)



Compositional Assume Guarantee in Proof Structure

- Only one implementation node, with all chosen properties
  - Proof Structure adds (in same node) “CAG Assumption Bundle”
  - This Bundle represents the set of mutual inductive assumes
- Propagation rules
  - Any counterexample can safely propagate
  - Proofs only propagate if all assertions pass
    - Otherwise inductive assumptions are overconstraining
  - For bounded proofs, minimum bound of any property == bound for all