

**A New Partial-Order Reduction Algorithm**  
**for Concurrent System Verification**

**Ratan Nalumasu**

**Ganesh Gopalakrishnan**

**Department of Computer Science,  
University of Utah, Salt Lake City**

# Overview of the talk

- Explicit enumeration
- Partial order reduction
- Proviso algorithms (**SPIN**, PO-Package)
- Non-Proviso algo (**two-phase**, stackmark)
- Experimental results
- Conclusions

# Explicit Enumeration

- Construct FSM model (Interleaving semantics)
- Execute model, checking for LTL properties
- Example applications:
  - Cache coherence protocols (UltraSparc design)
  - Requirements validation (Shuttle ascent ctrl.)
  - Telecommunication Protocols
  - Security protocols
- Often outperforms implicit enumeration

# An Example of Explicit Enumeration

**process p**

```

{ do
  :: x++ → c1 ? x
  :: (x == 0) → c2 ! y+1
  :: y-- → ...
od }

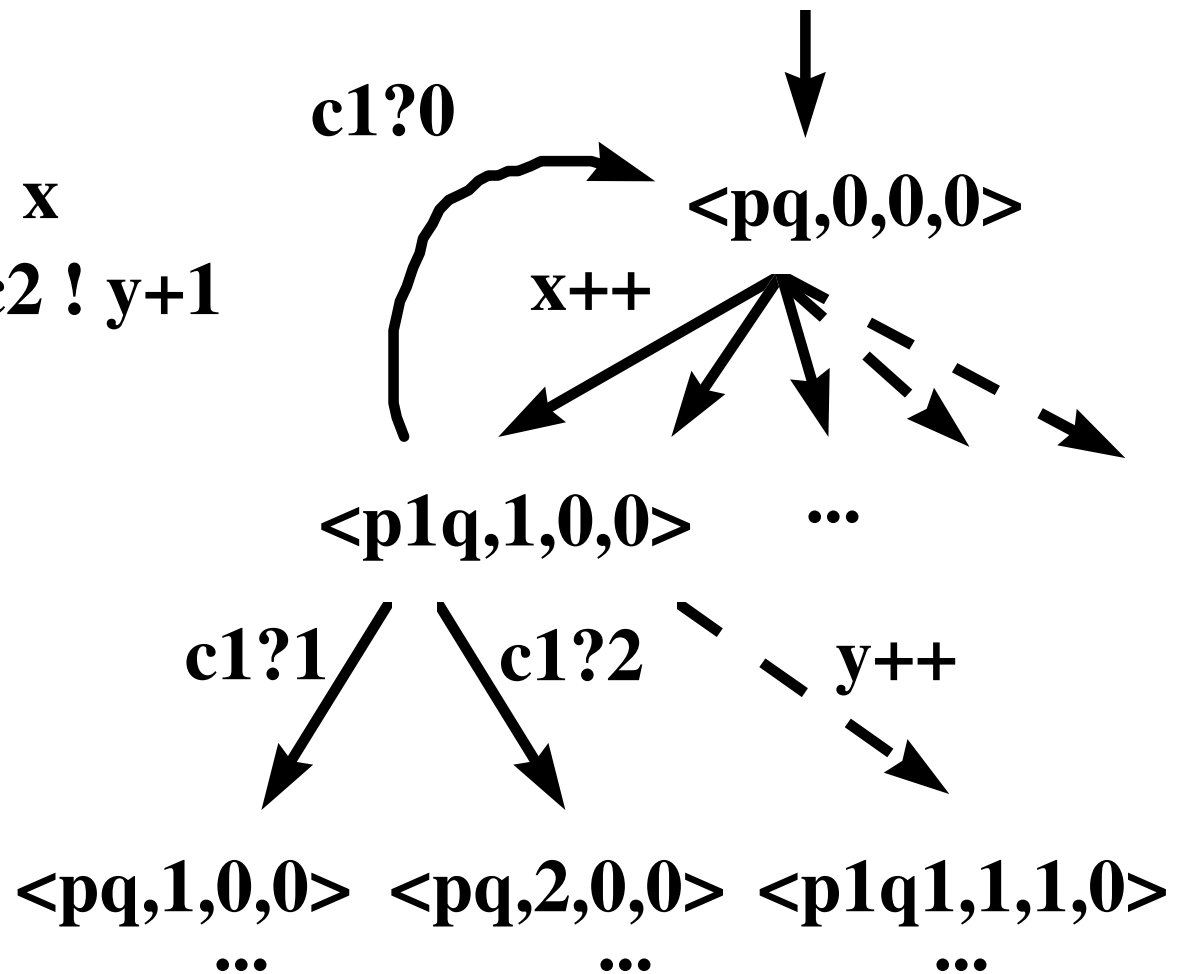
```

**process q**

```

{ do
  :: y++ → ...
  :: z++ → ...
od }

```



# State Explosion & Ways to Combat it

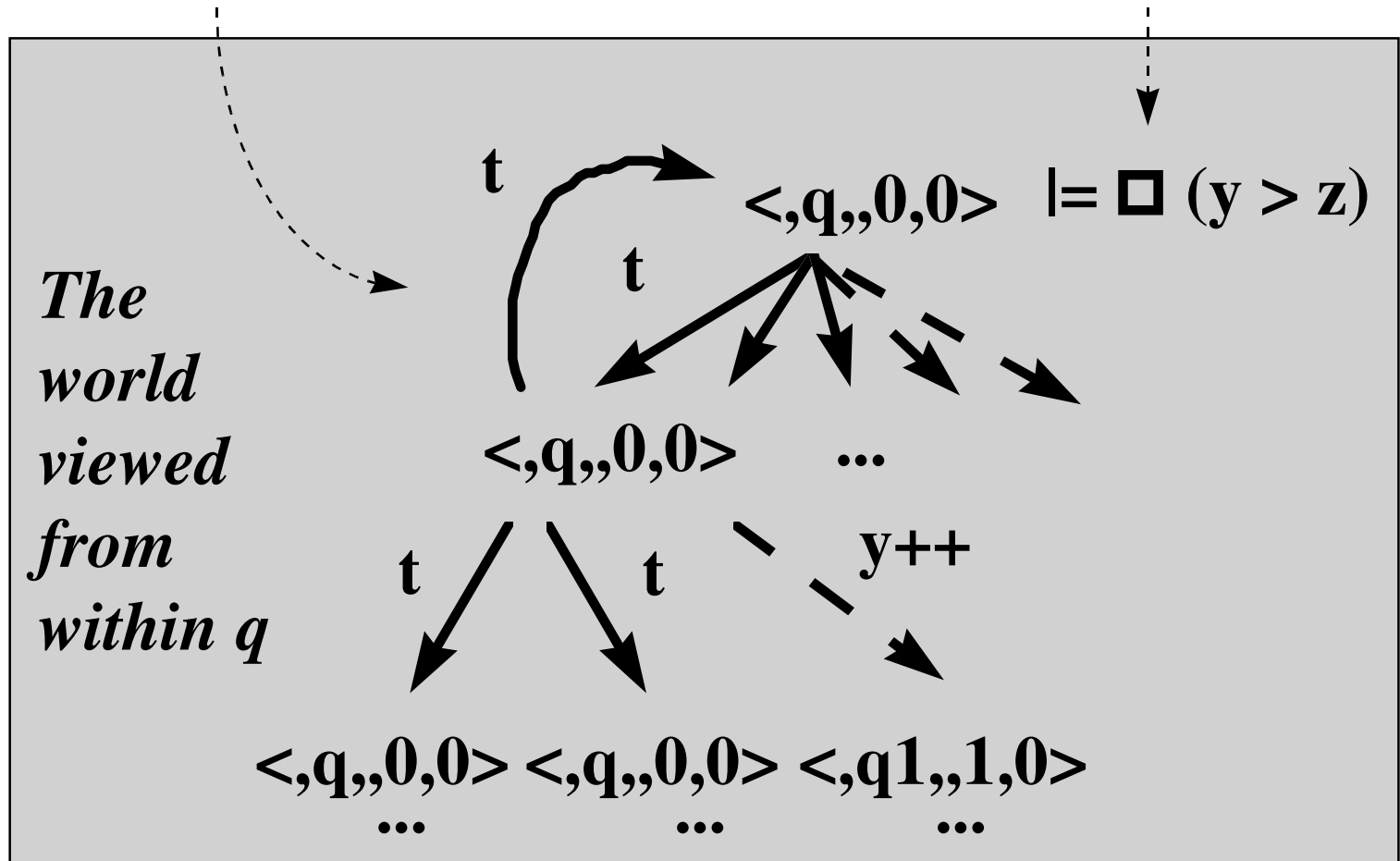
- Symmetry Reduction
- Selective State Caching
- Resetting “dead variables”
- Use of Online Algorithms
- **Partial Order Reduction**

# Basic Definitions

## Stuttering steps and Stuttering-invariant formulas

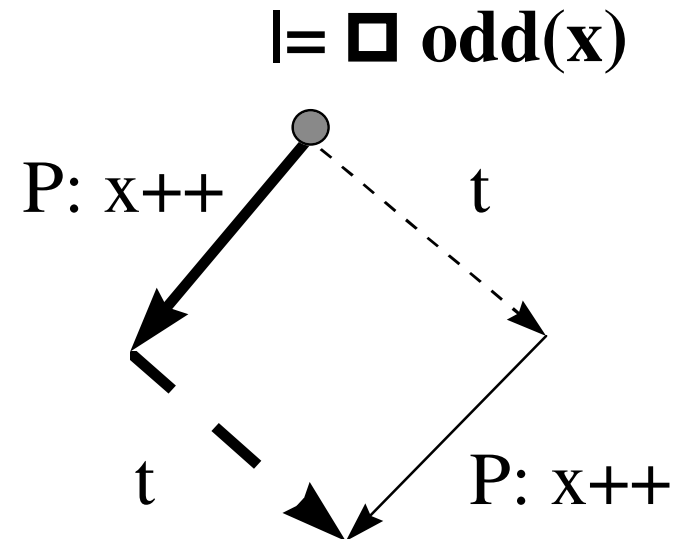
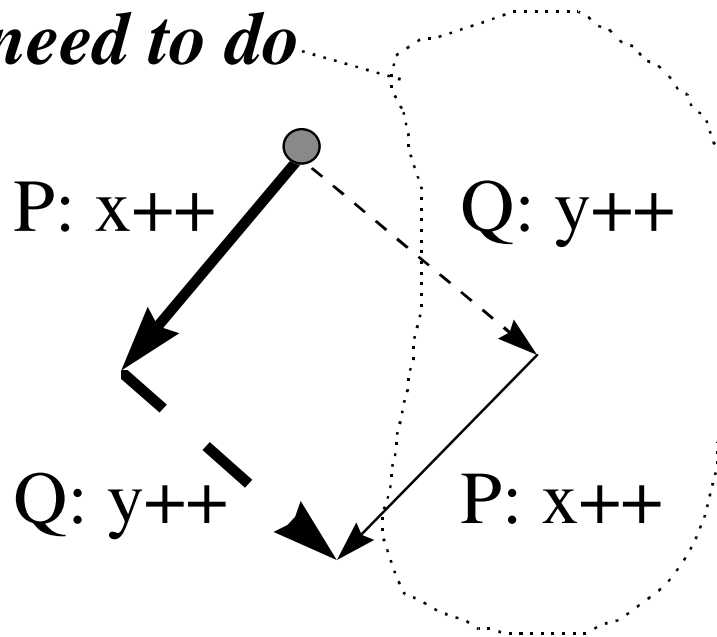
**global y**  
**proc p**  
 {local x  
 ... }

**proc q**  
 {local z  
 ...}



# PO Redn Idea #1 : Exploit stuttering eq.

*No need to do*

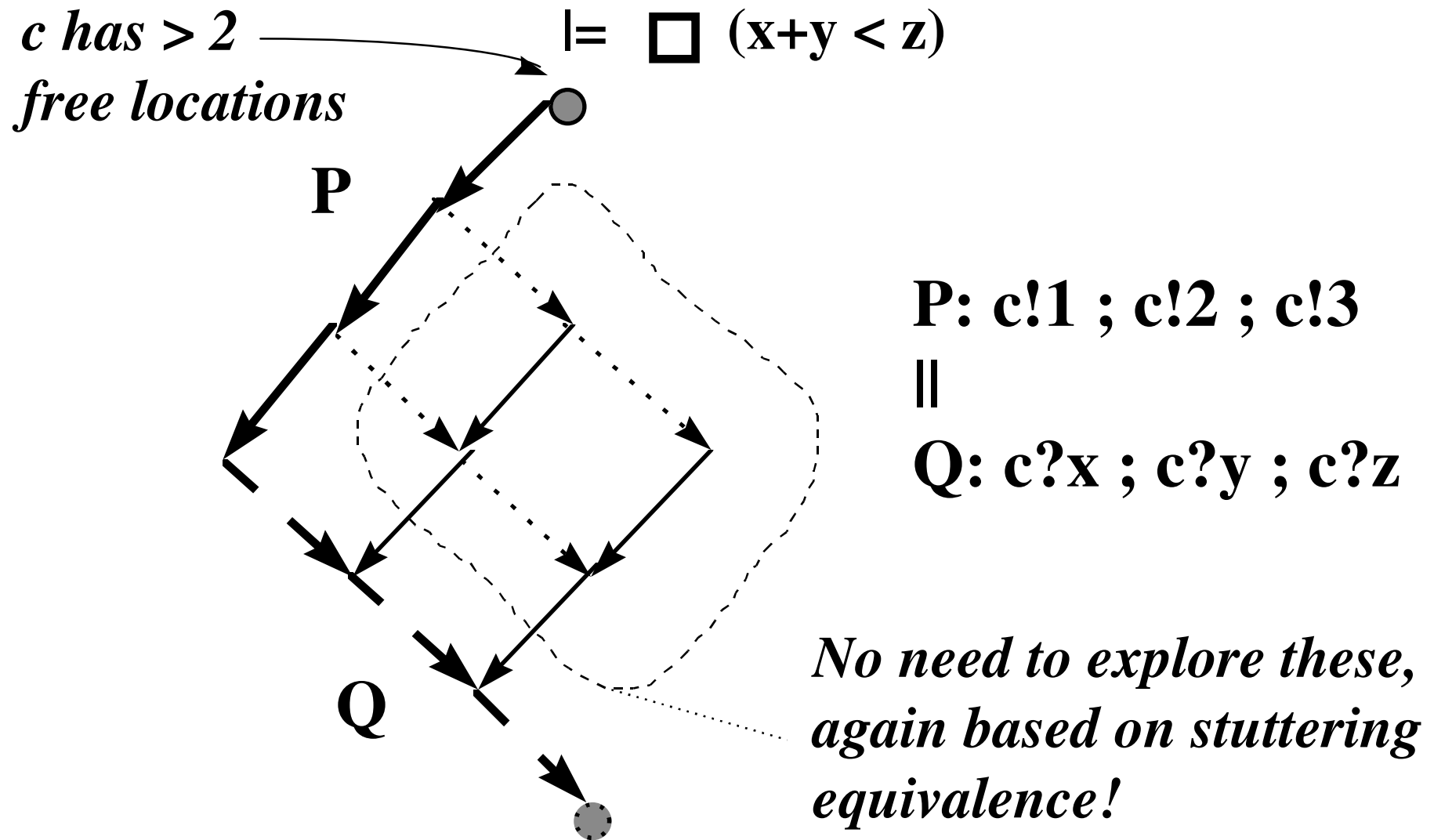


$\models \square \text{ odd}(x)$

**Processes P and Q have x and y as locals**

**Properties Stuttering Invariant & depend on x or y**

# Idea #1 Works Even with Channel I/O



# Idea #1 is Insufficient...

**P: c!1 ; c!2 ; c!3**

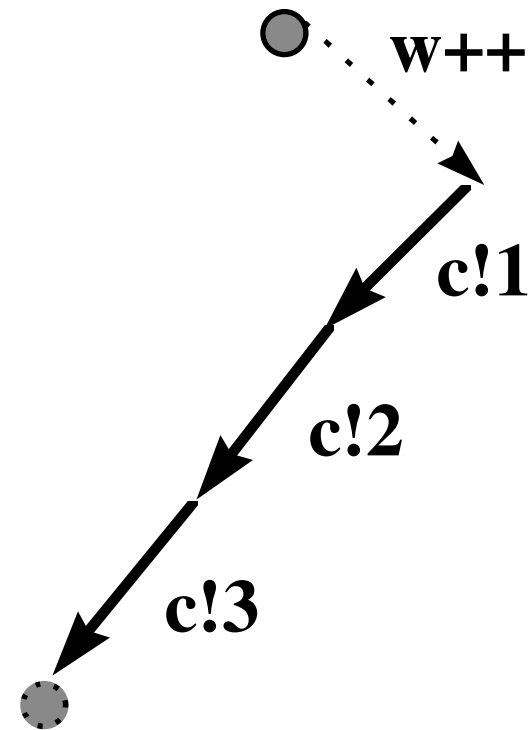
**||**

**Q: c?x ;  c?y ; c?z**

**| w++**

**heh! heh!**

*c has > 2 free*



# Why Idea #1 is Insufficient

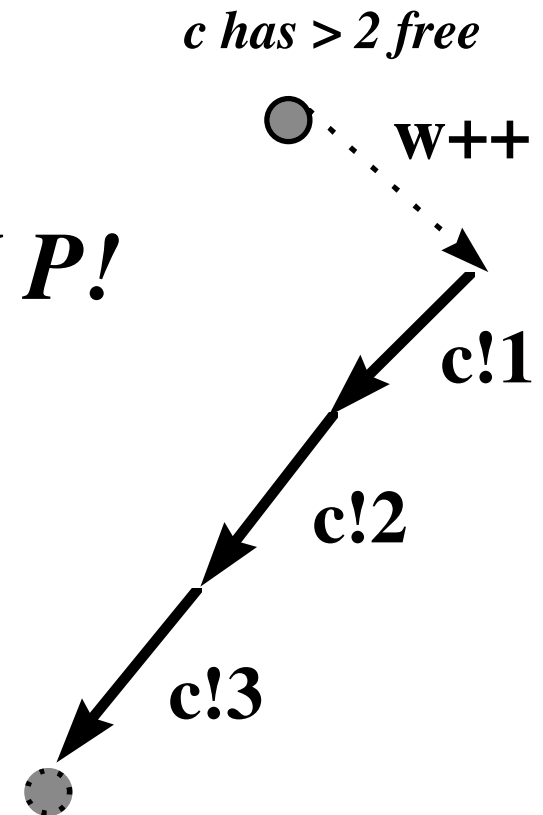
- *P can enable Q*
- *P can't disable Q*
- *Q can't affect P*
- ***MUST HAVE CHOSEN P!***

**P: c!1 ; c!2 ; c!3**

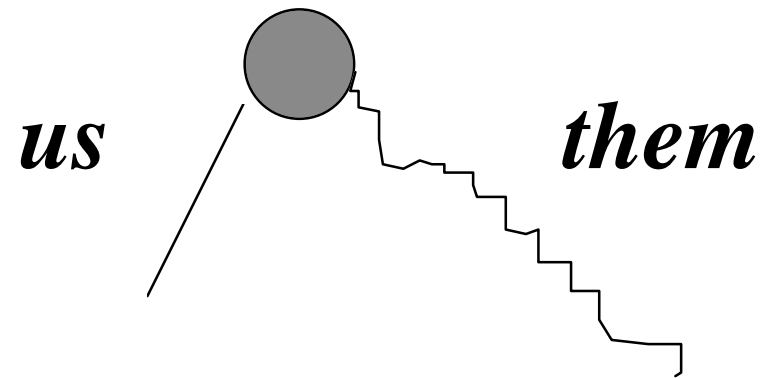
**||**

**Q: c?x ; c?y ; c?z**

**| w++**



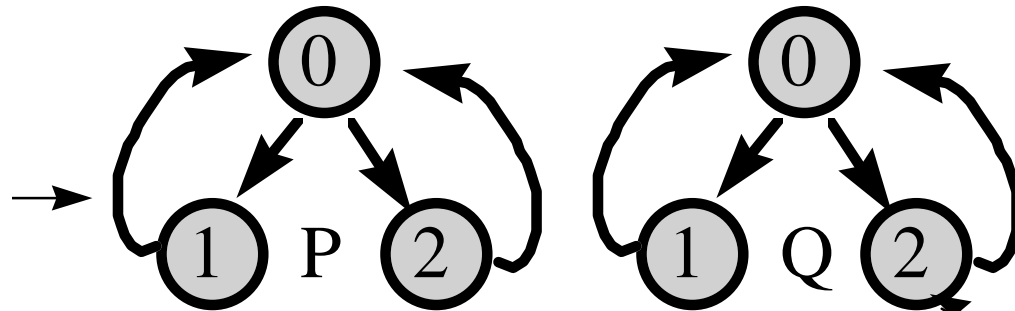
## Idea #2: Stuttering eql + Persistence



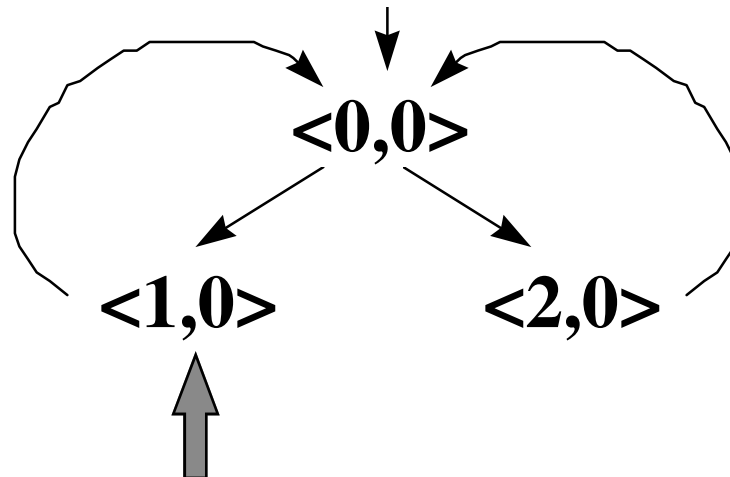
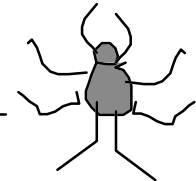
- **Our moves never disable their's**
- **Their moves don't affect us**
- **All moves involve locals or pt-pt chan**
- **Follow our moves (*Persistent Sets*)**

# Idea #2 Won't Do: Avoid "ignoring"


All  
local (t)  
actions



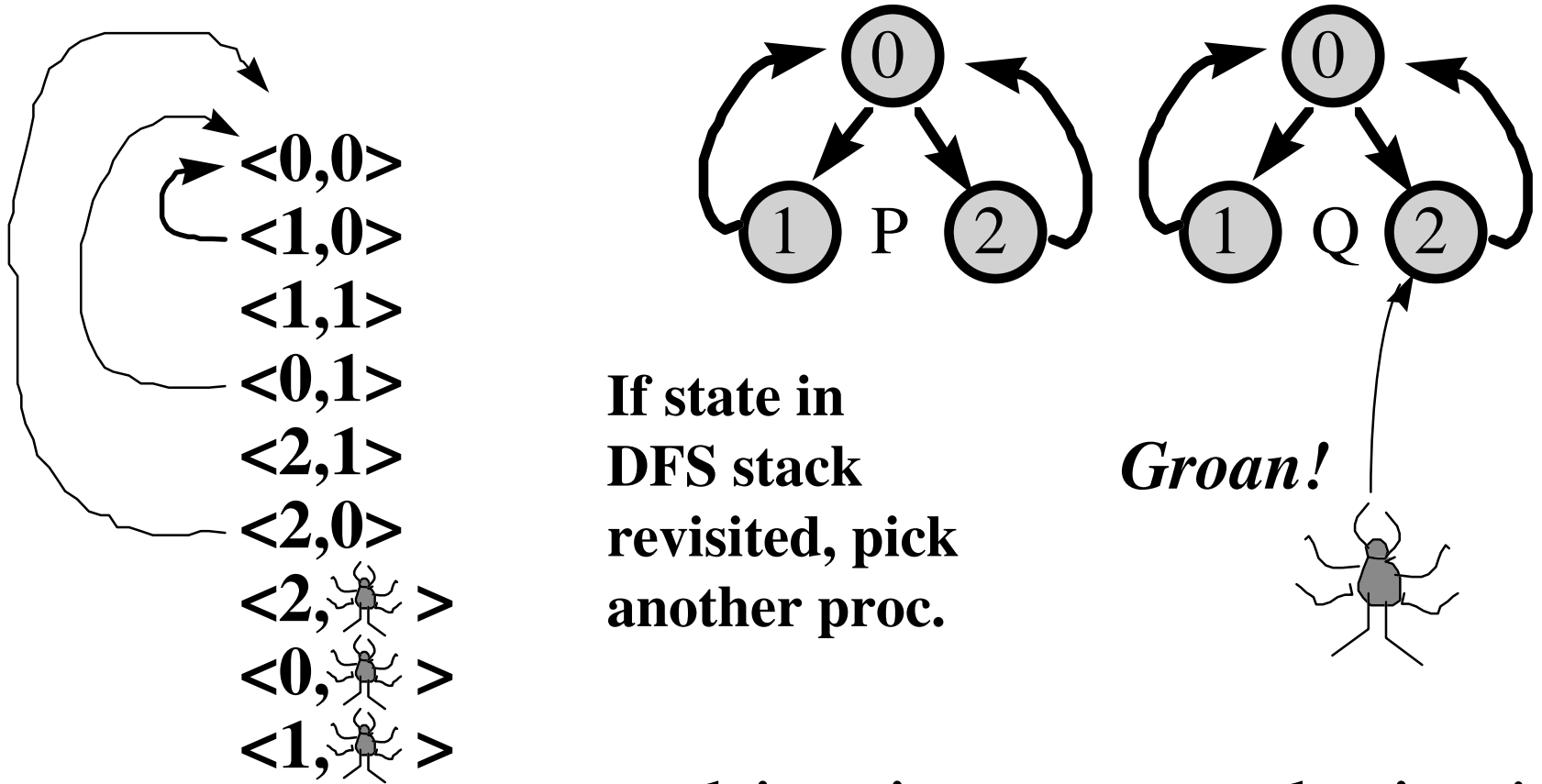
*heh! heh!*



*We aren't done here!*

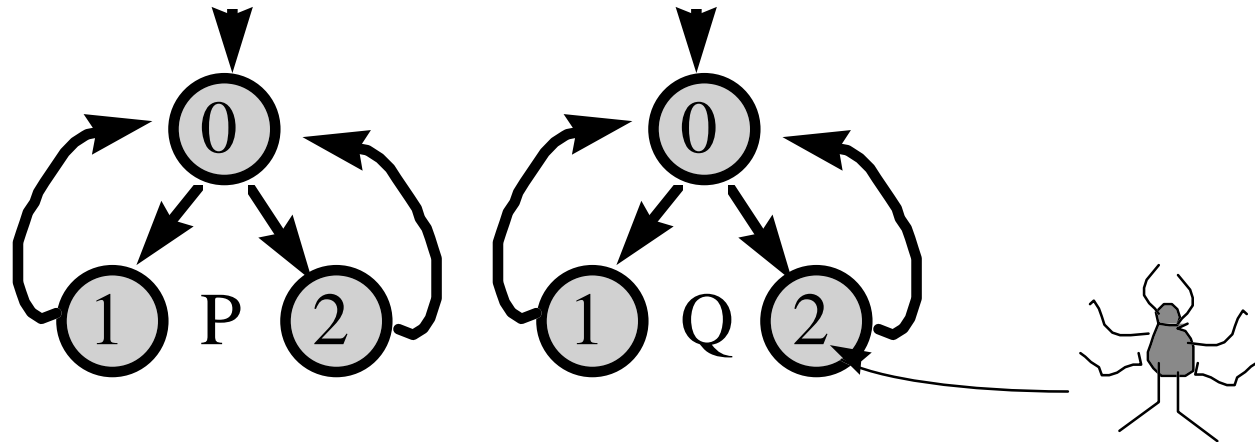
*No <\*, >*  
*generated!*

# Idea #3: Stut.eq. + Persist. + *Proviso*



... resulting in state explosion!

# Why Idea #3 is Not Always Optimal



**5** states suffice to reveal the error:

(  $\langle 0,0 \rangle$ ,  $\langle 1,0 \rangle$ ,  $\langle 2,0 \rangle$ ,  $\langle 0,1 \rangle$ ,  $\langle 0, \text{spider} \rangle$  )

***Proviso algos generate all 9!***

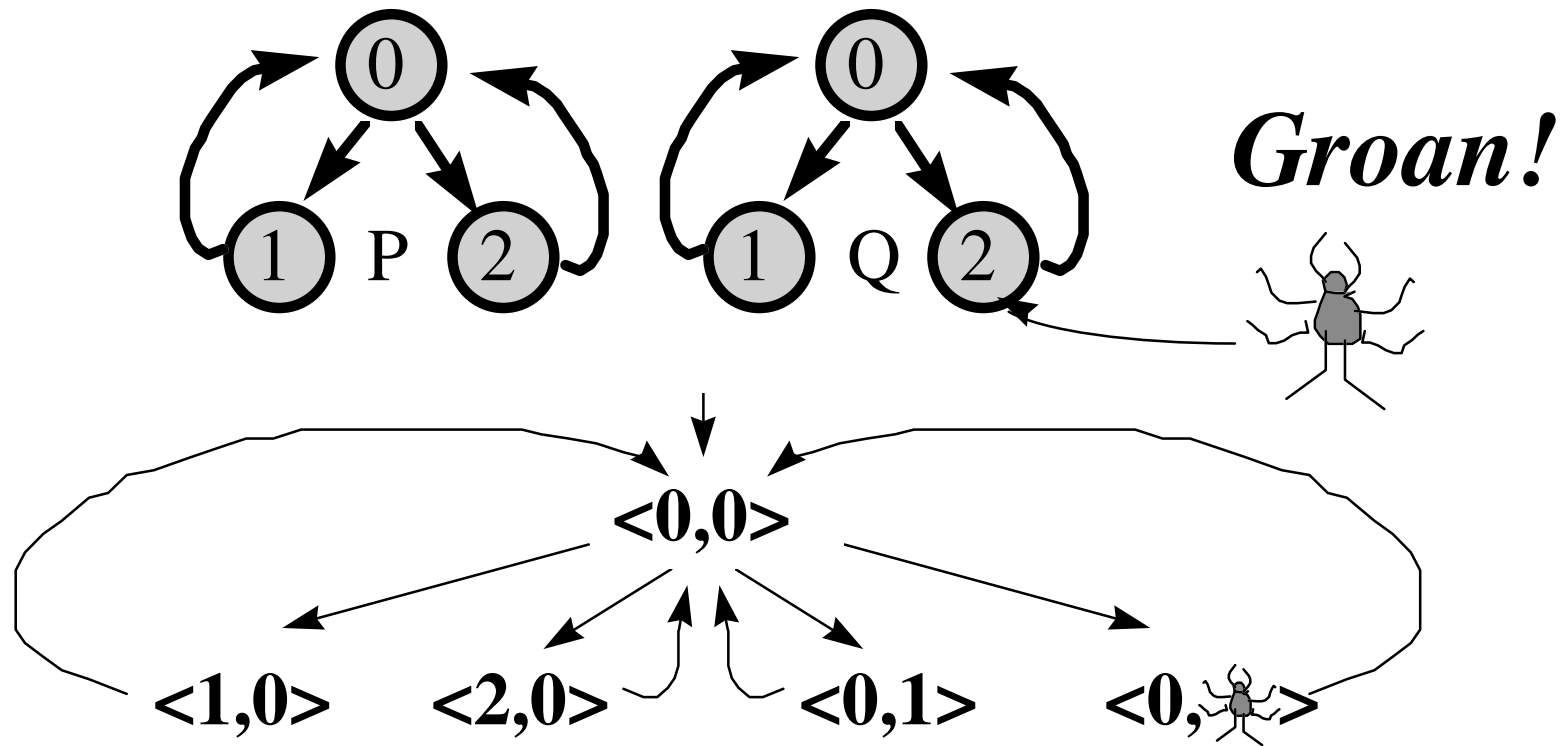
# Proviso Does “Show Up” in Practice...

*Many protocols in our domain (e.g. cache coherence) seemed to be taking too long or not finishing...*

	<i>States</i>	<i>Time (s)</i>
Mig (spin)	113,628	13.6
Inv (spin)	> 620,446	DNF --► Missed bug?

*Similar effects observed even in PO-Package*

Idea #4 (Two-Phase Algorithm):  
Stut.eq. + Persist. + *NON PROVISO!*

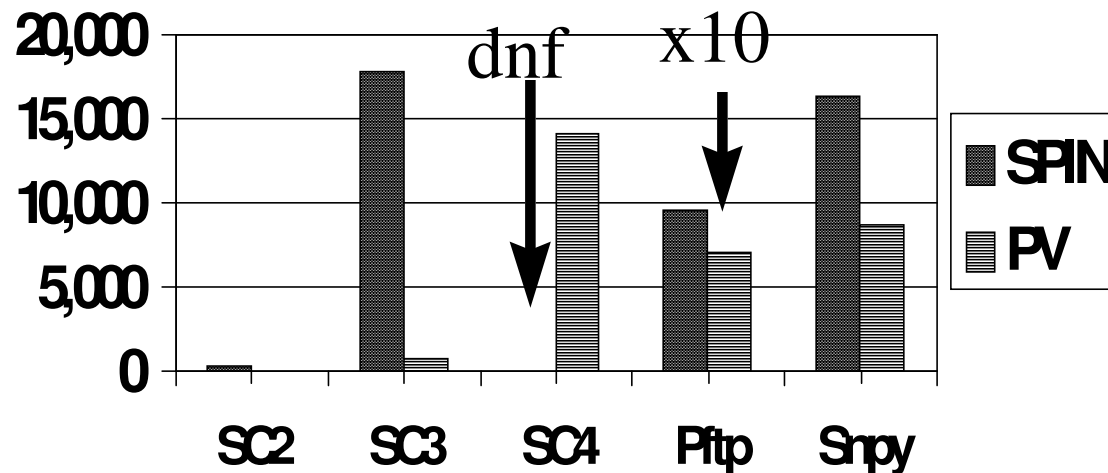


*PO Redn when deterministic (singleton persistent sets)*  
*Keep “mini hash-table” to avoid  $\langle 00 \rangle$  revisit confusion!*

# Proviso (SPIN, *PO-PACKAGE*) vs. Non-Proviso (2-Phase, *Stackmark*)

	<i>States</i>	<i>Time (s)</i>	
Mig (spin)	113,628	13.6	
Mig (2Ph. PV tool)	9,185	1.7	
Inv (spin)	> 620,446	DNF	--▶ Missed bug
Inv (2Ph. PV tool)	135,404	21.2	--▶ Found bug

States



# Conclusions

- Demonstrated that Proviso can be harmful thru SPIN, *PO-PACKAGE*, Two-Phase, and *Stackmark*
- Two-phase is implemented in our model-checker PV
- Accepts a subset of Promela
- Performs stutter-free safety-checking
- Supports selective caching
- Recently Proved to preserve **stutter-free LTL**
- PV is in use in verifying cache coherence protocols
- PV is available upon request