A Dynamic Firing Speculation to Speedup Distributed Symbolic State-Space Generation

Ming-Ying Chung and Gianfranco Ciardo

Department of Computer Science and Engineering

University of California at Riverside

Riverside, CA 92521, USA

{chung,ciardo}@cs.ucr.edu

LaTeX

April 28, 2006

Outline

- **Introduction**
  - *Parallel and distributed model checking (PDMC)*
  - *Multi-valued Decision Diagrams (MDD)* for state-space encoding
  - Recent research on symbolic PDMC
- **Background**
  - *Saturation* state-space generation and its distributed version
  - *Speculative computing* to speedup distributed state-space generation
- **Graph-based firing speculation**
  - *Firing pattern graph*
  - *Pattern length* based speculation
  - *Weighted score* based speculation
- **Experimental results and conclusion**

Introduction

Parallel and distributed model checking

- *Formal verification* : for *quality assurance*
- *Model checking* : a model-based automatic verification approach

  E. Clarke and E. Emerson. *Synthesis of synchronization skeletons for branching time temporal logic*, Logic of Programs 1981
- *State-space generation* : the first step in model checking (memory-intensive)
- *Binary decision diagrams* (BDDs) : *symbolic* state-space construction

  R. Bryant, *Graph-based algorithms for boolean function manipulation*, IEEE TC 1986
- *Parallel and distributed model checking* (PDMC) :

  Collect computation resources via network or different computer architectures
- PDMC on *shared-memory multi-processor* : special hardware

  S. Kimura and E. M. Clarke, *A parallel algorithm for constructing BDDs*, ICCD 1990
- PDMC on *distributed shared memory* : special software

  Y. Parasuram, E. Stabler, and S.-K. Chin, *Parallel implementation of BDD algorithm using a DSM*, HICSS 1994

Structured discrete-state models

- A *structured discrete state model* is a triple $(\widehat{\mathcal{S}}, \mathcal{S}^{init}, \mathcal{N})$
  - $\widehat{\mathcal{S}}$ is the set of *potential states* of the model
  - $\mathcal{S}^{init} \subseteq \widehat{\mathcal{S}}$ is the set of *initial states*
  - $\mathcal{N} : \widehat{\mathcal{S}} \to 2^{\widehat{\mathcal{S}}}$ is the *next-state* function
  - It is actually specified implicitly at a higher level model

- The *reachable state-space* $\mathcal{S} \in \widehat{\mathcal{S}}$ is the smallest set
  - containing $\mathcal{S}^{init}$
  - closed with respect to $\mathcal{N}$

$$\mathcal{S} = \mathcal{S}^{init} \cup \mathcal{N}(\mathcal{S}^{init}) \cup \mathcal{N}^2(\mathcal{S}^{init}) \cup \mathcal{N}^3(\mathcal{S}^{init}) \cup \cdots = \mathcal{N}^*(\mathcal{S}^{init})$$

Quasi-reduced ordered MDDs

- *A global state described by K variables*

$$\boxed{\mathbf{i} = (\mathbf{i}_K, \ldots, \mathbf{i}_1)}$$

- Each variable corresponds to one level

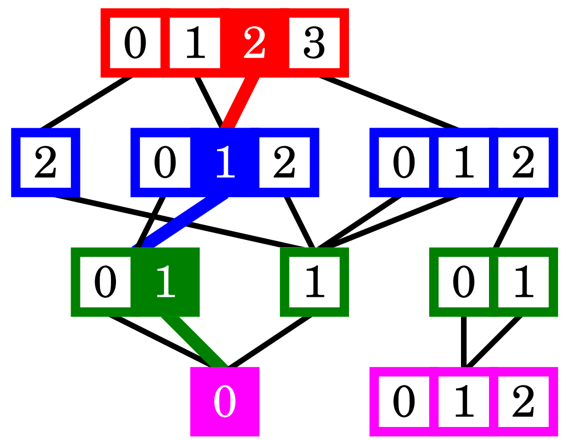
$\mathcal{S}_4 = \{0, 1, 2, 3\}$

$\mathcal{S}_3 = \{0, 1, 2\}$

$\mathcal{S}_2 = \{0, 1\}$

$\mathcal{S}_1 = \{0, 1, 2\}$

9 nodes

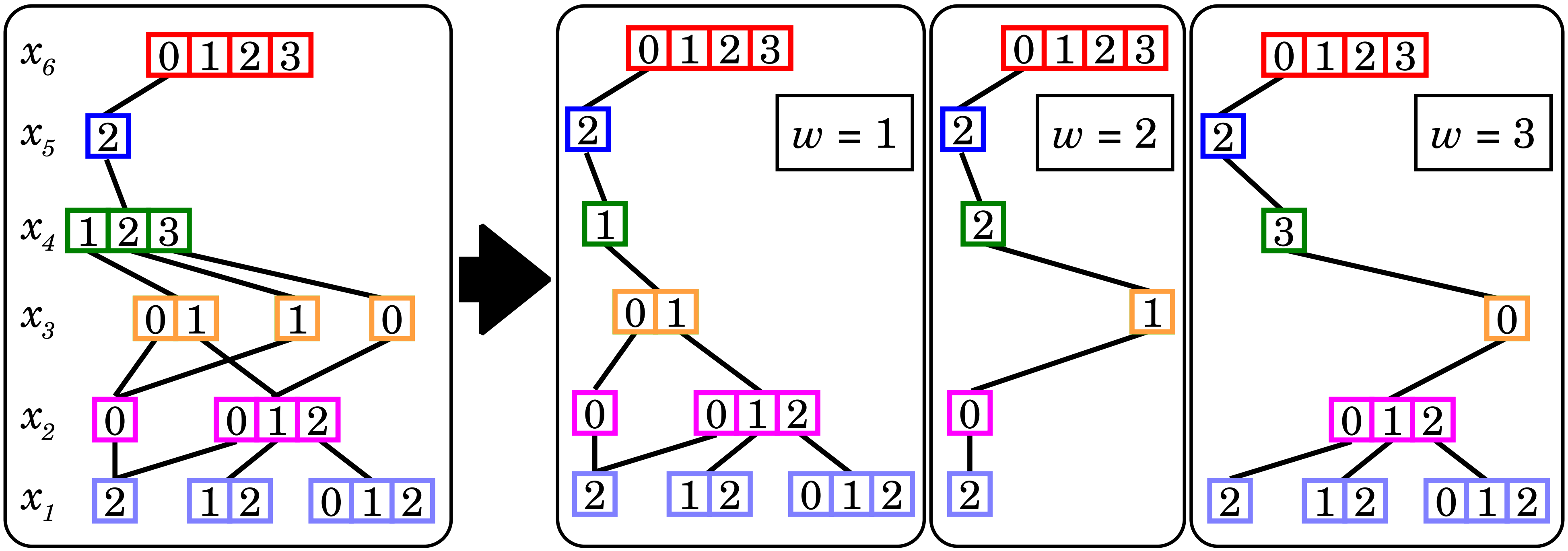


$$\mathcal{S} = \underbrace{\left\{ \begin{matrix}0\\2\\1\\0\end{matrix}, \begin{matrix}1\\0\\0\\0\end{matrix}, \begin{matrix}1\\0\\1\\0\end{matrix}, \begin{matrix}1\\1\\0\\0\end{matrix}, \begin{matrix}1\\1\\1\\0\end{matrix}, \begin{matrix}1\\2\\1\\0\end{matrix}, \begin{matrix}2\\0\\0\\0\end{matrix}, \begin{matrix}2\\0\\1\\0\end{matrix}, \begin{matrix}2\\1\\0\\0\end{matrix}, \left| \begin{matrix}2\\1\\1\\0\end{matrix} \right|, \begin{matrix}2\\2\\1\\0\end{matrix}, \begin{matrix}3\\0\\1\\0\end{matrix}, \begin{matrix}3\\1\\1\\0\end{matrix}, \begin{matrix}3\\2\\0\\0\end{matrix}, \begin{matrix}3\\2\\0\\1\end{matrix}, \begin{matrix}3\\2\\0\\2\end{matrix}, \begin{matrix}3\\2\\1\\0\end{matrix}, \begin{matrix}3\\2\\1\\1\end{matrix}, \begin{matrix}3\\2\\1\\2\end{matrix} \right\}}_{19 \text{ states}}$$

T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Multi-valued decision diagrams: theory and applications*, Journal of multiple-valued logic 1998

Symbolic PDMC on network of workstations (NOW)

Job-based slicing : natural way to parallelize distributed state-space generation



T. Stornetta and F. Brewer, *An efficient parallel BDD package*, DAC 1996

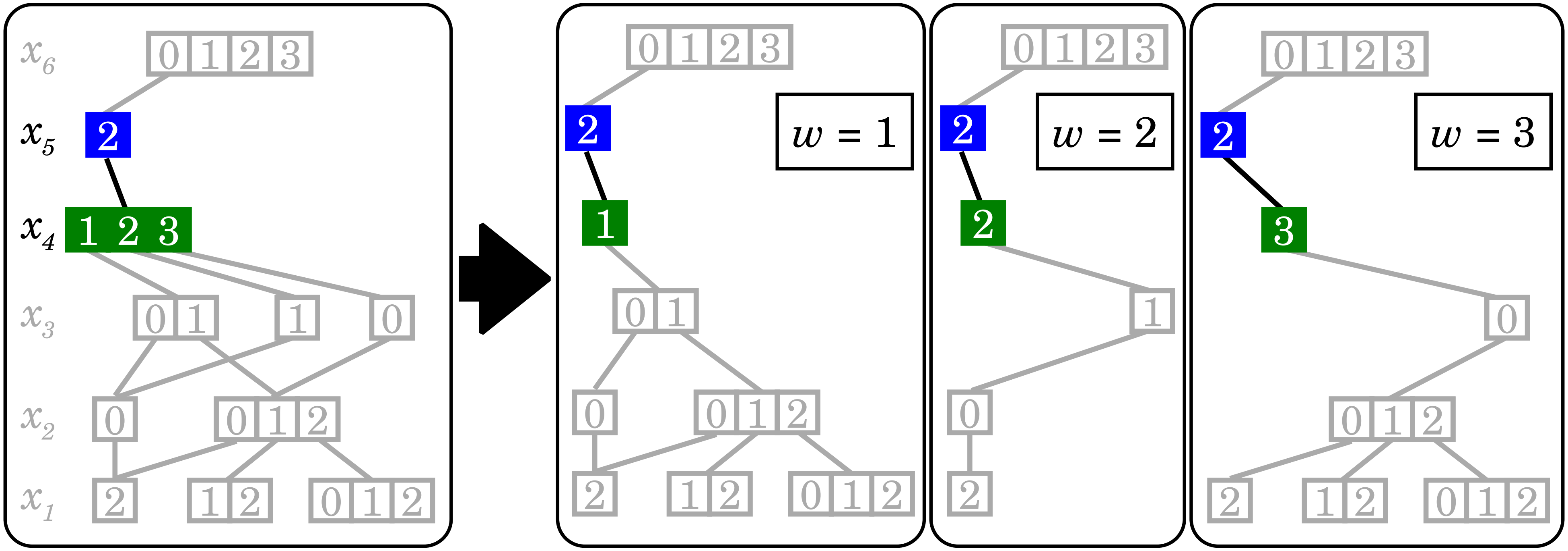
K. Milvang-Jensen and A. Hu, *BDDNOW*, FMCAD 1998

T. Heyman, D. Geist, O. Grumberg, T. Heyman, and A. Schuster, *Achieving scalability in parallel reachability analysis of very large circuits*, CAV 2000

O. Grumberg, T. Heyman, and A. Schuster, *A work-efficient distributed algorithm for reachability analysis*, CAV 2003

Symbolic PDMC on network of workstations (NOW)

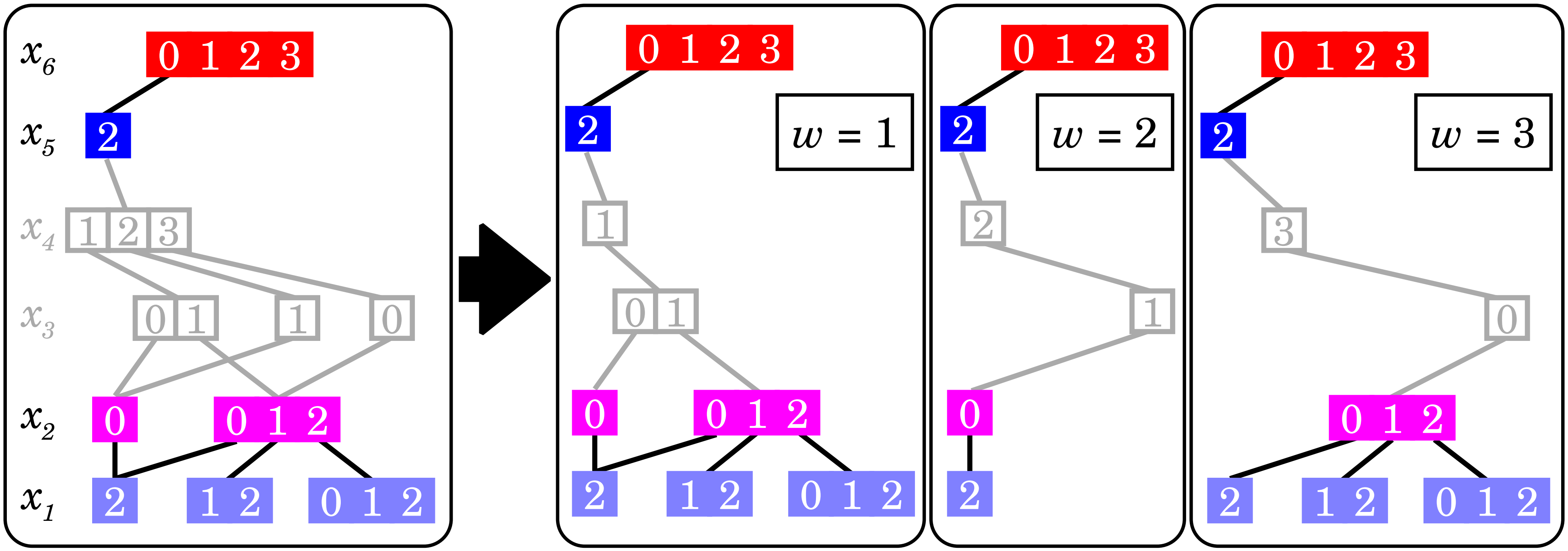
Job-based slicing : $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$ $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for $i \neq j$



- Finding the most appropriate set of variables to slice the image is not trivial
- Not only overlap image computation but also create duplicate MDD nodes
- Frequent global synchronizations are required to minimize duplicate work
- Scalability becomes an issue

Symbolic PDMC on network of workstations (NOW)

Job-based slicing : $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$ $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for $i \neq j$

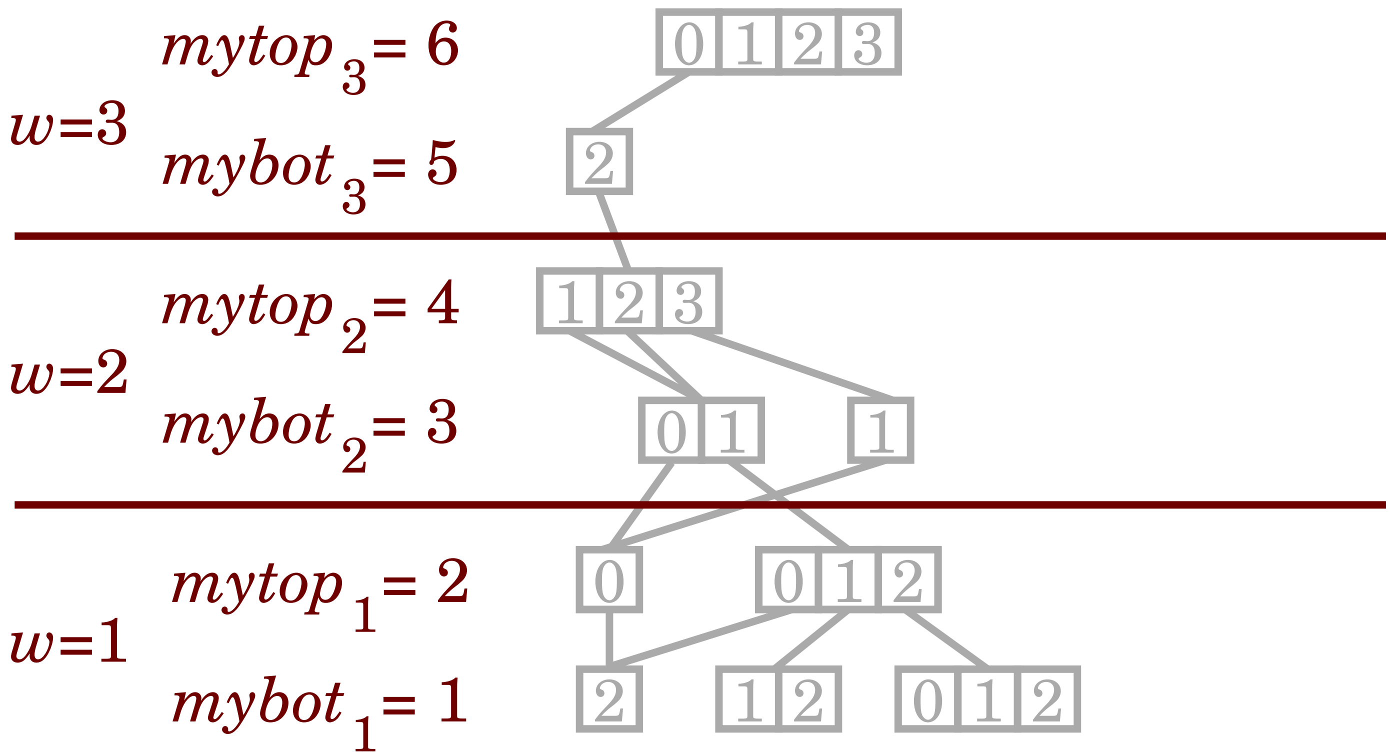


- Finding the most appropriate set of variables to slice the image is not trivial
- Not only overlap image computation but also create duplicate MDD nodes
- Frequent global synchronizations are required to minimize duplicate work
- Scalability becomes an issue

Symbolic PDMC on network of workstations (NOW)

Level-based slicing : to avoid duplicate work and global synchronization

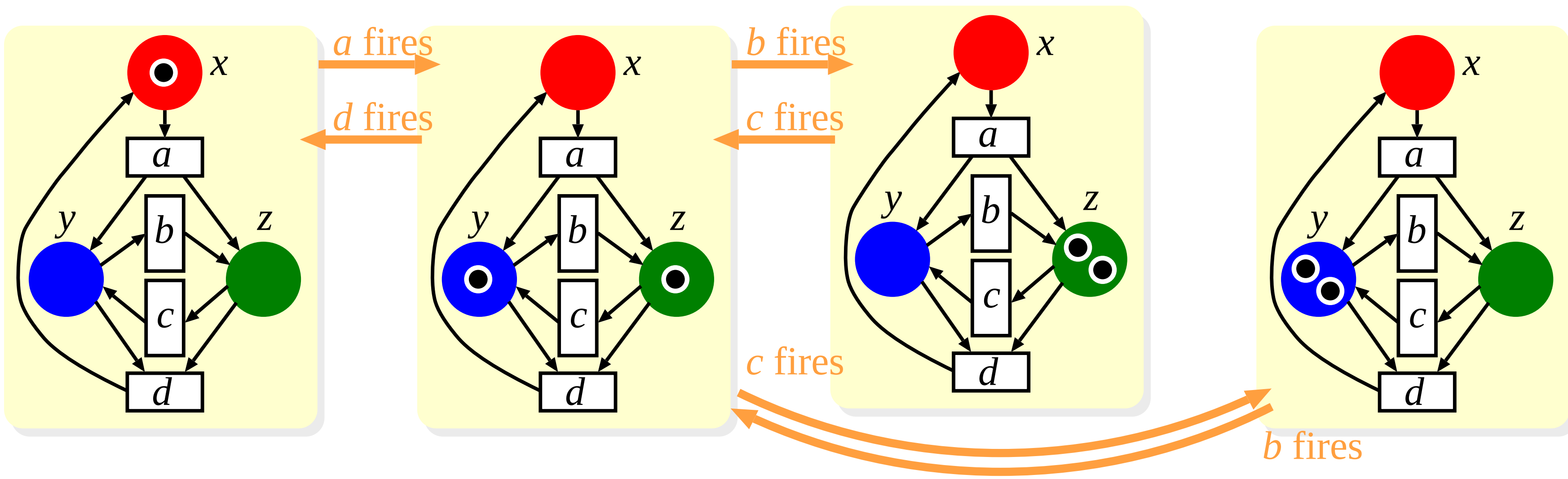
R. Ranjan, J. Snaghavi, R. Brayton, and A. Sangiovanni-Vincentelli, *BDDs on NOWs*, ICCD 1996



- Difficult to perform a good memory load balancing
- Sequentialized distributed computation is hard to parallelize

Background

Event locality

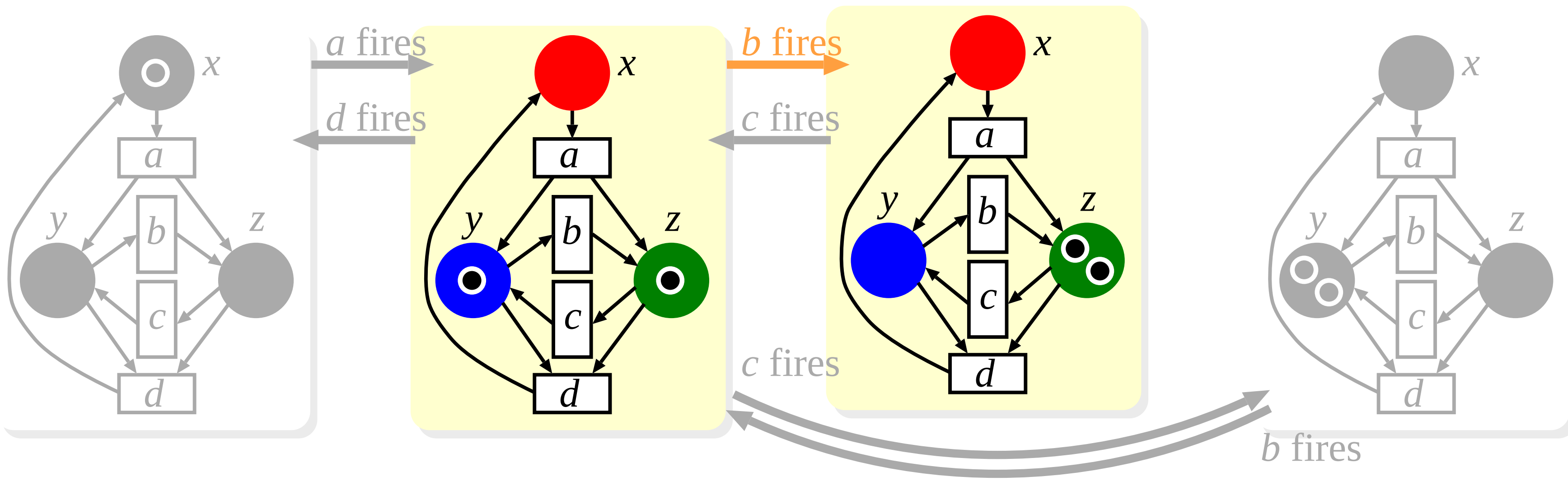


- Assuming the variable order is x, y and z (from high to low level)
- $Top(\alpha)$ *is the highest MDD level affected by event* α

  Event α is independent of any MDD level higher than $Top(\alpha)$

  - $Top(a) = Top(d) = x$
  - $Top(b) = Top(c) = y$

Event locality

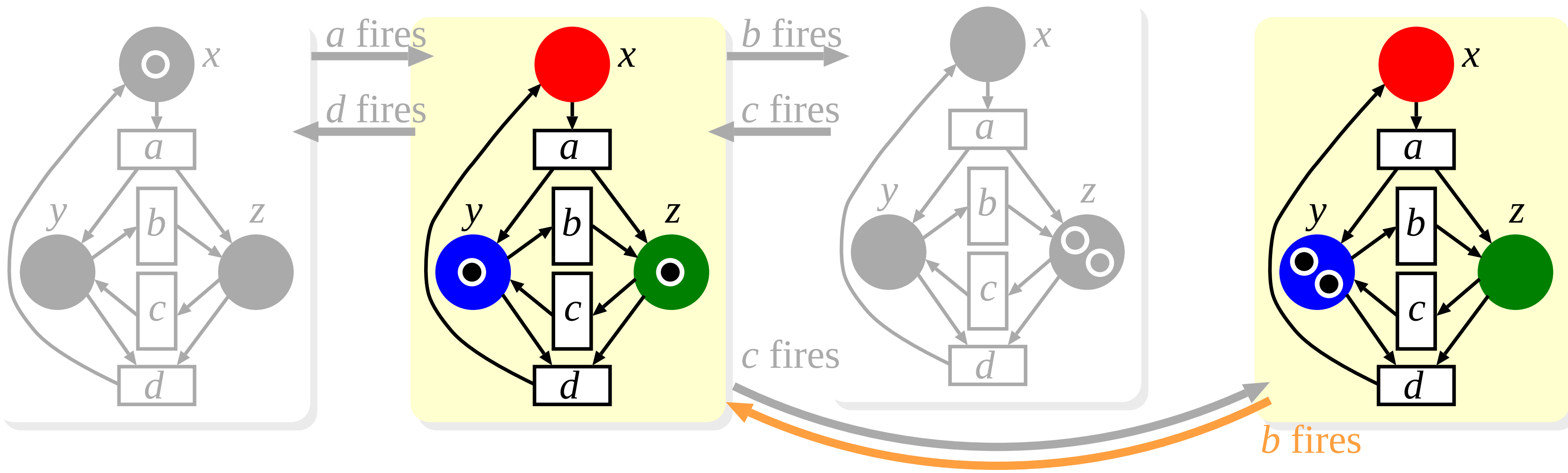


- Assuming the variable order is x, y and z (from high to low level)
- *$Top(\alpha)$ is the highest MDD level affected by event α*

  Event α is independent of any MDD level higher than $Top(\alpha)$

  - $Top(a) = Top(d) = x$
  - $Top(b) = Top(c) = y$

Event locality



- Assuming the variable order is x, y and z (from high to low level)
- *$Top(\alpha)$ is the highest MDD level affected by event α*

  Event α is independent of any MDD level higher than $Top(\alpha)$

  - $Top(a) = Top(d) = x$
  - $Top(b) = Top(c) = y$

Saturation-style state-space generation

An MDD node at level k is called *saturated* if the state set it encodes is a fixed point with respect to any event α such that $Top(\alpha) \leq k$ where α is independent of any level higher than k

1. Build the MDD encoding of initial state set

2. From $k=1$ to K, saturate each node at k by firing all α s.t. $Top(\alpha)=k$

2.1 *If this creates nodes below k, saturate them immediately upon creation*

3. When the root node is saturated, all reachable states have been discovered

Saturation has *enormous time and memory efficiency* compared to traditional symbolic breath-first iteration for asynchronous models

G. Ciardo, G. Lüettgen, and R. Siminiceanu, *Saturation : an efficient iteration strategy for symbolic state-space generation*, TACAS 2001

Saturation NOW

- **Problem** :

  We target relatively large models where state-space generation with a single workstation must rely on virtual memory or runs out of memory

- **Solution** :

  - Apply saturation algorithm on a NOW to increase the available memory
  - *Level-based MDDs slicing for distributed state-space construction*

    No overlapped image computation

    No duplicate MDD node

    $\longrightarrow$ no global synchronization
  - *A nested approach of dynamic memory load balancing*

    Only pairwise communication is enough to achieve nearly ideal memory scalability

M.-Y. Chung and G. Ciardo, *Saturation NOW*, QEST 2004

Speculative firing prediction

- **Problem** :
  - *Only one workstation is active* at any time

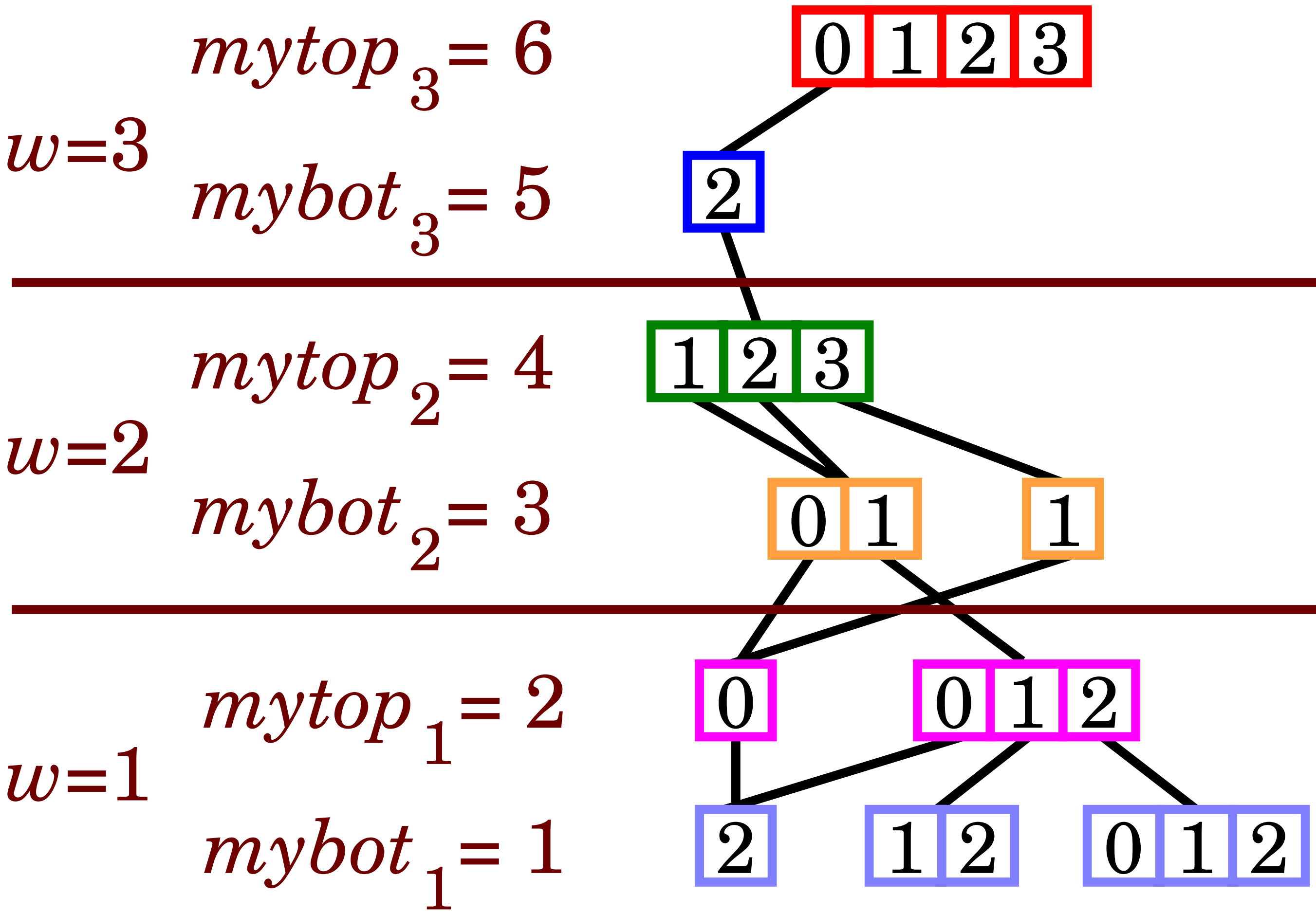
    No theoretical speedup is achieved
  - The level-based slicing scheme *sequentializes the distributed computation*

    Parallelization becomes a challenge
- **Solution** :
  - *Idle workstations perform some event firings in advance* and cache the results of those firings
  - One firing cache per MDD level

    A hash used to cache firing results
  - Some later firing requests might retrieve those results from firing caches and can be resolved quickly

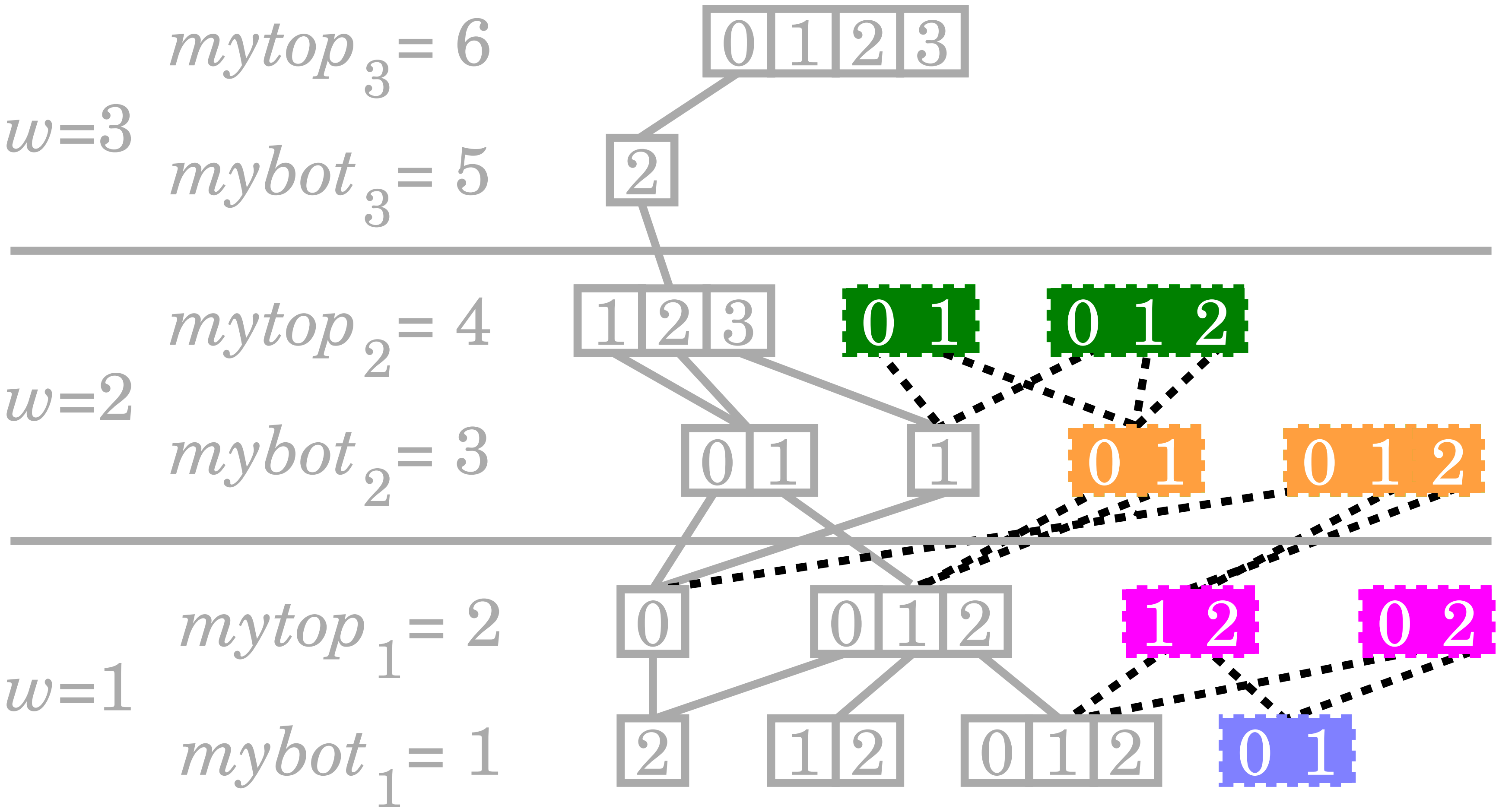
M.-Y. Chung and G. Ciardo, *A Pattern Recognition Approach for Speculative Firing Prediction in Distributed Saturation State-Space Generation*, PDMC 2005

Idea of speculative firing



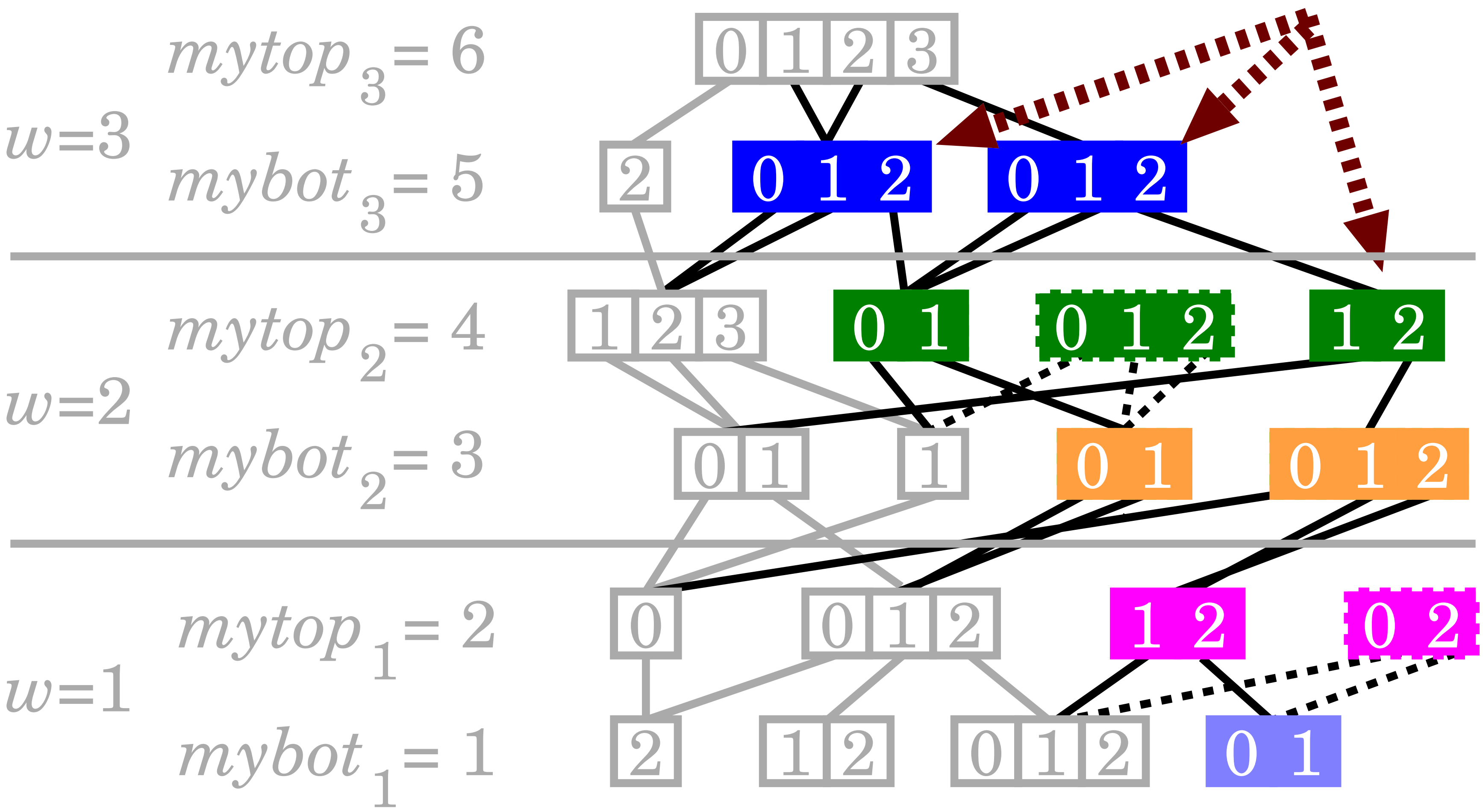
w_3, w_2, and w_1 perform distributed state-space generation

Idea of speculative firing



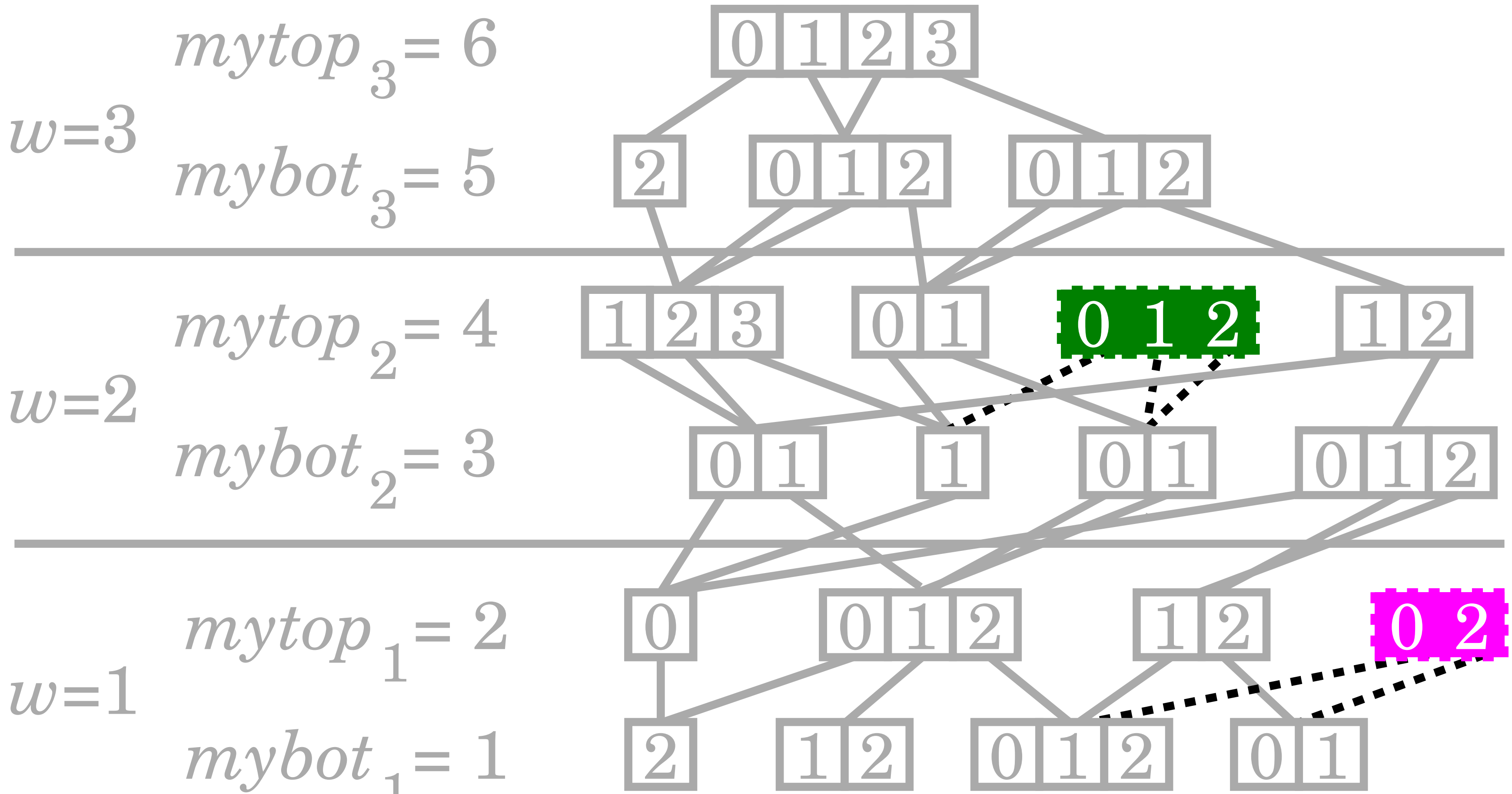
w_2 and w_1 perform speculative computing together or individually

Idea of speculative firing



Some of the speculative firing results were retrieved from the firing caches

Idea of speculative firing



Many useless MDD nodes $\longrightarrow$ high memory consumption

Pattern recognition approach

- **Problem** :
  - Do not know a priori whether an event will be fired on some MDD node
  - A NAÏVE approach : *idle workstations exhaust all possible firings*
    - It might create many useless nodes and firing cache entries
    - It will not stop when the prediction does not help much
- **Observation** :
  - $\mathcal{E}_{patt}(p) = \{e$: event e was fired on node p during generation $\}$
  - Some MDD nodes might have similar or the same set of events fired on
- **Solution** :
  - A more informed firing speculation based on recognized *firing patterns*

M.-Y. Chung and G. Ciardo, *A Pattern Recognition Approach for Speculative Firing Prediction in Distributed Saturation State-Space Generation*, PDMC 2005

Graph-based speculative firing

- **Problem** : Firing patterns are stored explicitly
  - The information of pattern evolution cannot be preserved
  - Storing this auxiliary information is somewhat expensive

- **Solution** : *A graph to encode firing patterns implicitly for each level*
  - The information of the evolution of firing patterns can be utilized to improve the accuracy of firing speculation
  - MDD nodes can share the encoding of the same firing patterns which can reduce the memory overhead

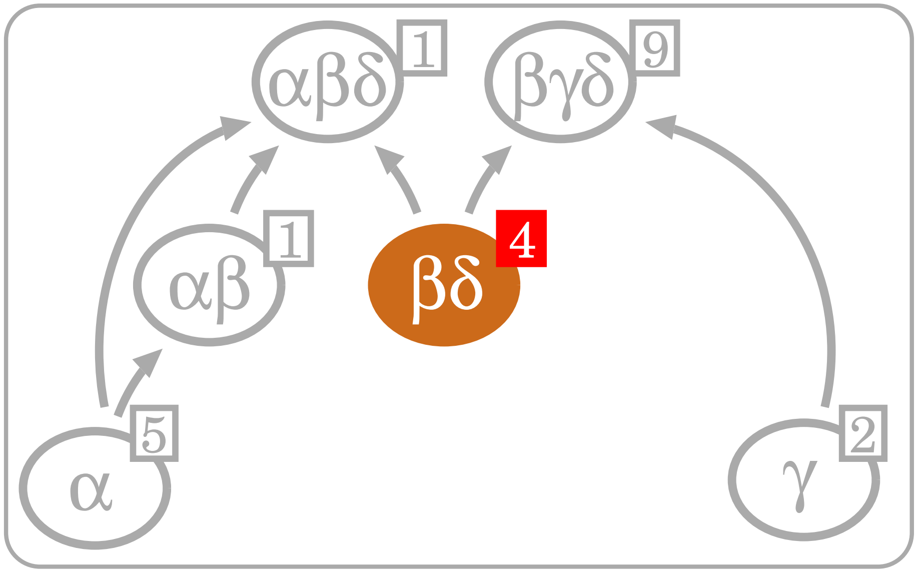
Graph-based Speculative Firing

Firing pattern graph

- Firing patterns are stored in pattern graphs, one per level

  Directed acyclic graph $G_k = (V_k, E_k)$ for $K \geq k \geq 1$
- Each pattern graph node has an associated reference counter

  Remove the pattern graph node $v \in V_k$ for some k whenever $v.ref = 0$



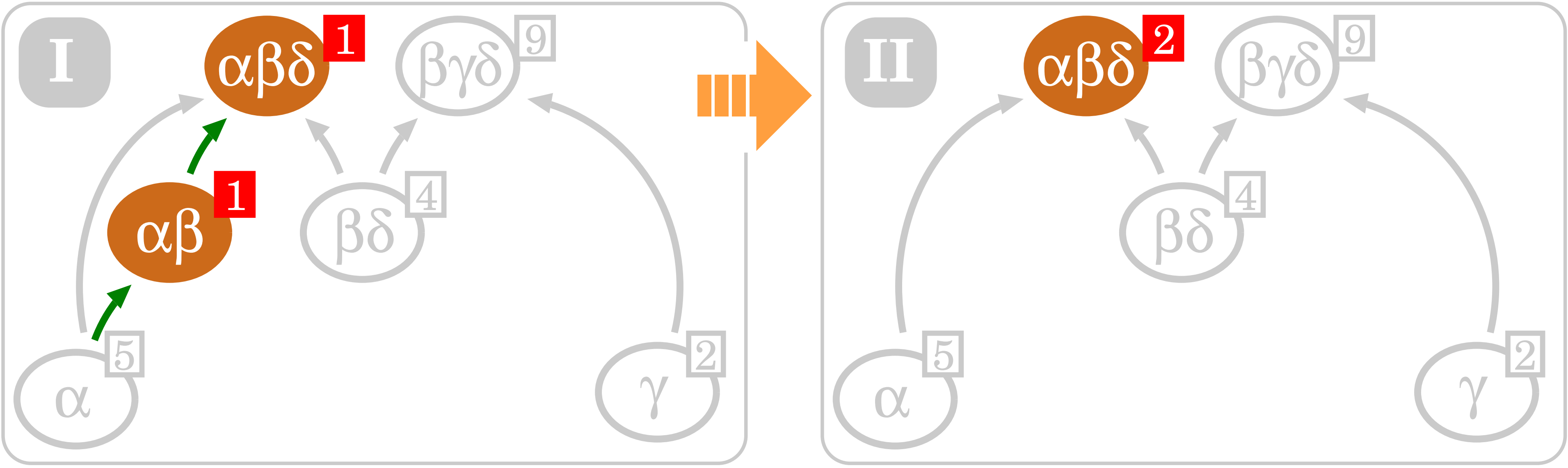
Events β and δ have been fired on four MDD nodes at some level k

Firing pattern graph

- Firing patterns are stored in pattern graphs, one per level

  Directed acyclic graph $G_k = (V_k, E_k)$ for $K \geq k \geq 1$
- Each pattern graph node has an associated reference counter

  Remove the pattern graph node $v \in V_k$ for some k whenever $v.ref = 0$



Update 1. **Firing event δ on an MDD node with current firing pattern $\{\alpha, \beta\}$**

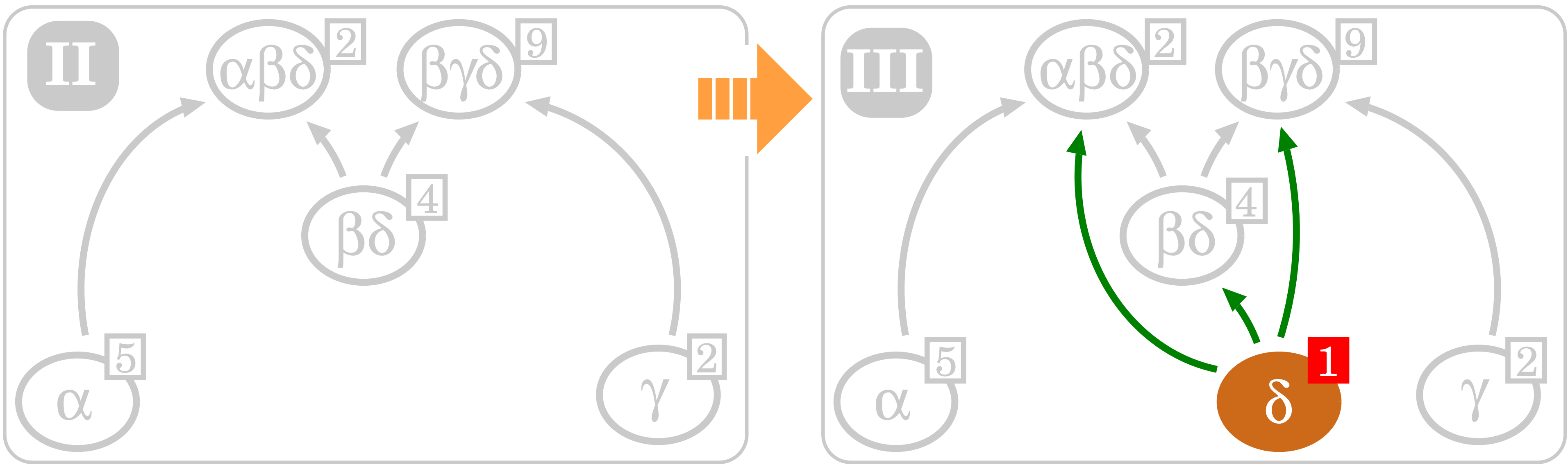
Firing pattern graph

- Firing patterns are stored in pattern graphs, one per level

  Directed acyclic graph $G_k = (V_k, E_k)$ for $K \geq k \geq 1$

- Each pattern graph node has an associated reference counter

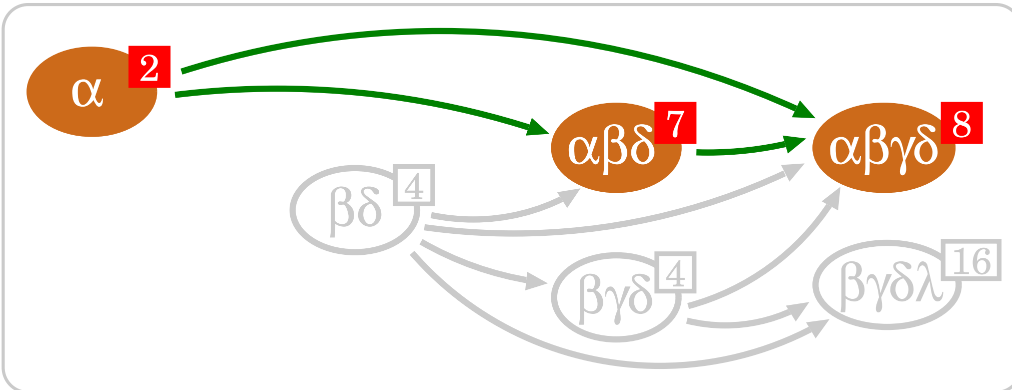
  Remove the pattern graph node $v \in V_k$ for some k whenever $v.ref = 0$



Update 2. **Firing event δ on a newly created MDD node**

Graph-based speculative firing

- Every path in a pattern graph reveals a possible evolution of some pattern
- There might be multiple ways to grow a pattern



- **Goal** :
    - Maximize the usefulness of the speculative firing results
    - Minimize the space and time overhead
- **Solution** :

- Speculate only when it is idle
- *Dynamically adjust the speculation aggressiveness* according to

$$SpecHitRate = \frac{\text{number of speculative results used}}{\text{number of speculative results created}}$$

Each workstation initializes a variable $\mathit{MaxDiff}$

- *Increase* $\mathit{MaxDiff}$ whenever $\mathit{SpecHitRate}$ increase
- Speculatively fire event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying

  $|\mathcal{E}_u| - |\mathcal{E}_v| \leq \mathit{MaxDiff}$

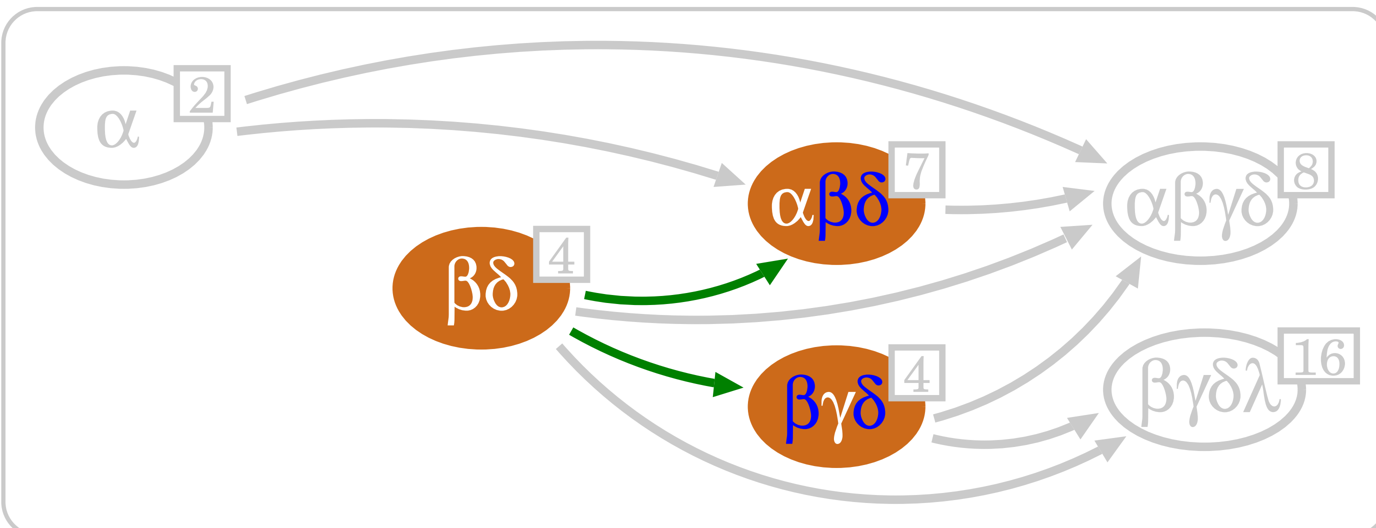
Pattern length based speculation

Each workstation initializes a variable $MaxDiff$

- *Increase* $MaxDiff$ whenever $SpecHitRate$ increase
- Speculatively fire event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any PG node $u \in v.parent$ satisfying

  $|\mathcal{E}_u| - |\mathcal{E}_v| \leq MaxDiff$

Example : MDD node p referencing pattern graph node encoding pattern $\{\beta, \delta\}$



$MaxDiff = 1$: speculatively fire events α and γ on MDD node p

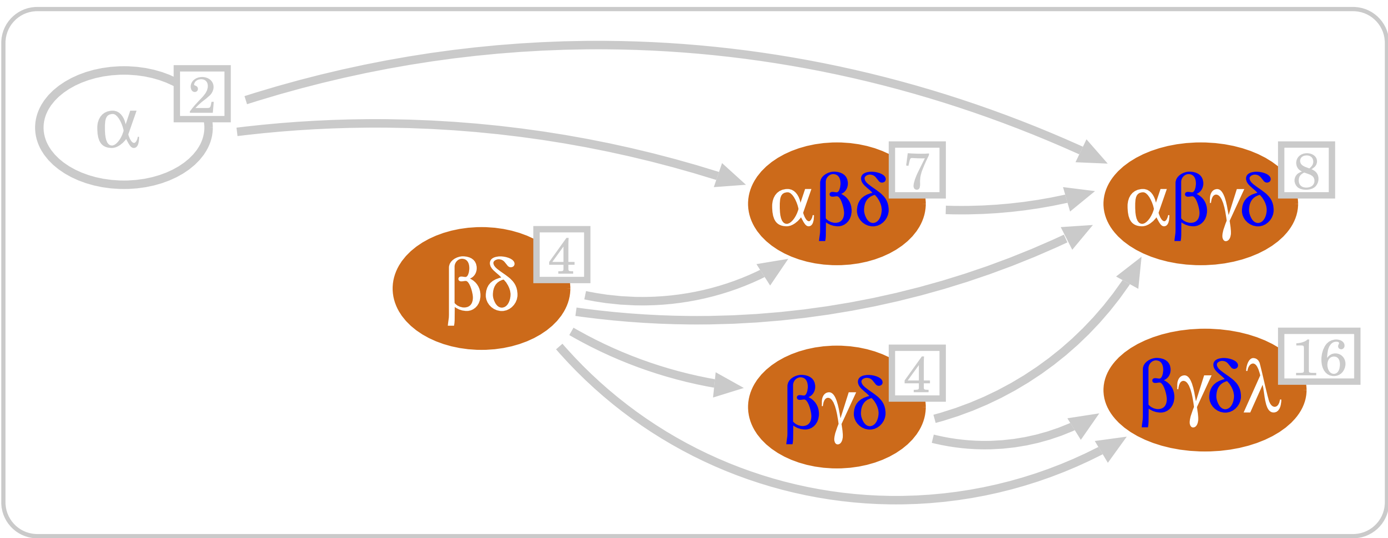
Pattern length based speculation

Each workstation initializes a variable $\mathit{MaxDiff}$

- *Increase* $\mathit{MaxDiff}$ whenever $\mathit{SpecHitRate}$ increase
- Speculatively fire event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying

  $|\mathcal{E}_u| - |\mathcal{E}_v| \le \mathit{MaxDiff}$

Example : MDD node p referencing pattern graph node encoding pattern $\{\beta, \delta\}$



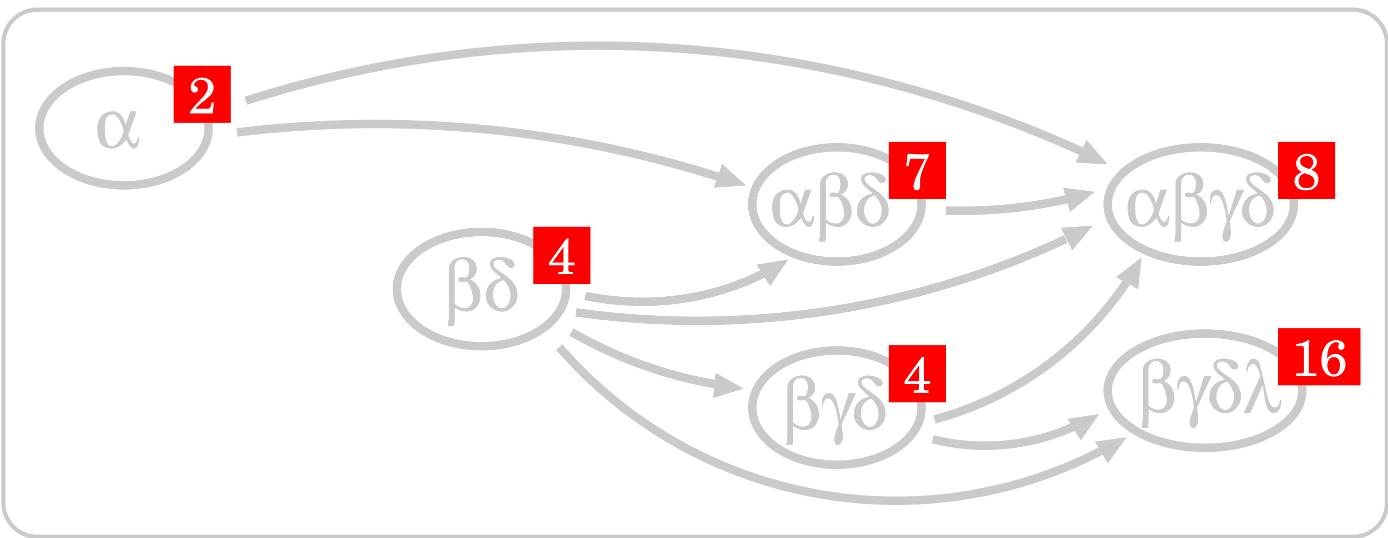
$\mathit{MaxDiff} = 2$: speculatively fire events α, γ, and λ on MDD node p

Pattern length based speculation

Each workstation initializes a variable $MaxDiff$

- *Increase* $MaxDiff$ whenever $SpecHitRate$ increase
- Speculatively fire event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying

  $|\mathcal{E}_u| - |\mathcal{E}_v| \leq MaxDiff$



Reference counters are only used to discard useless patterns graph nodes

Weighted score based speculation

Each workstation initializes a variable $MinScore$

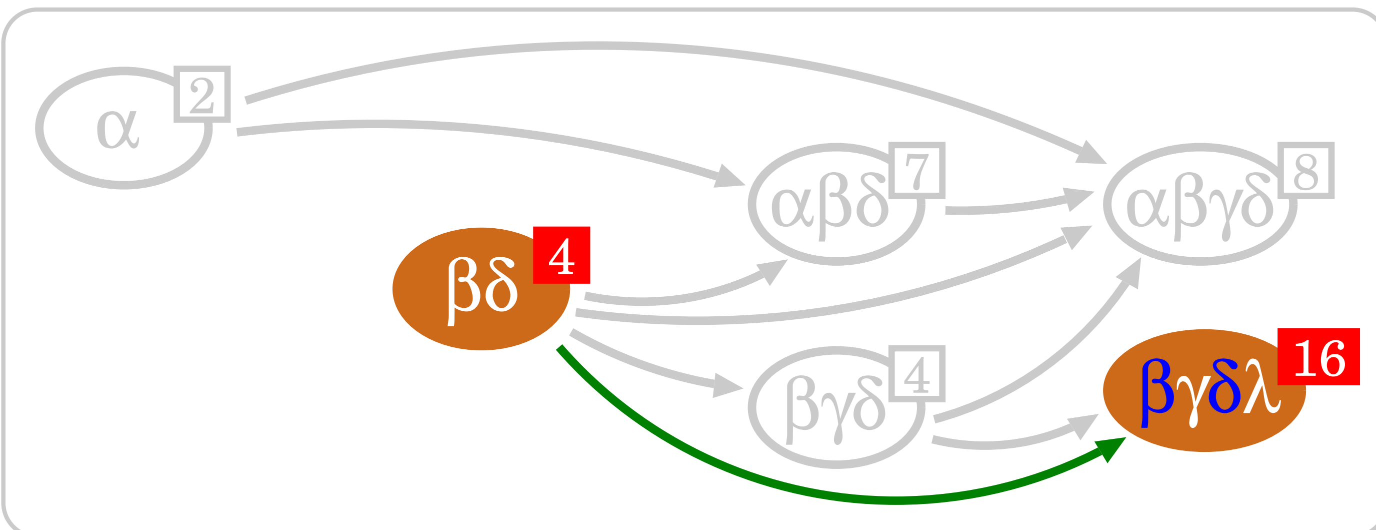
- *Decrease* $MinScore$ whenever $SpecHitRate$ increase
- Speculatively fire event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying $u.ref/(|\mathcal{E}_u| - |\mathcal{E}_v|) \geq MinScore$

Weighted score based speculation

Each workstation initializes a variable $MinScore$

- *Decrease* $MinScore$ whenever $SpecHitRate$ increase
- Speculatively fire event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying $u.ref/(|\mathcal{E}_u| - |\mathcal{E}_v|) \geq MinScore$

Example : MDD node p referencing pattern graph node encoding pattern $\{\beta, \delta\}$



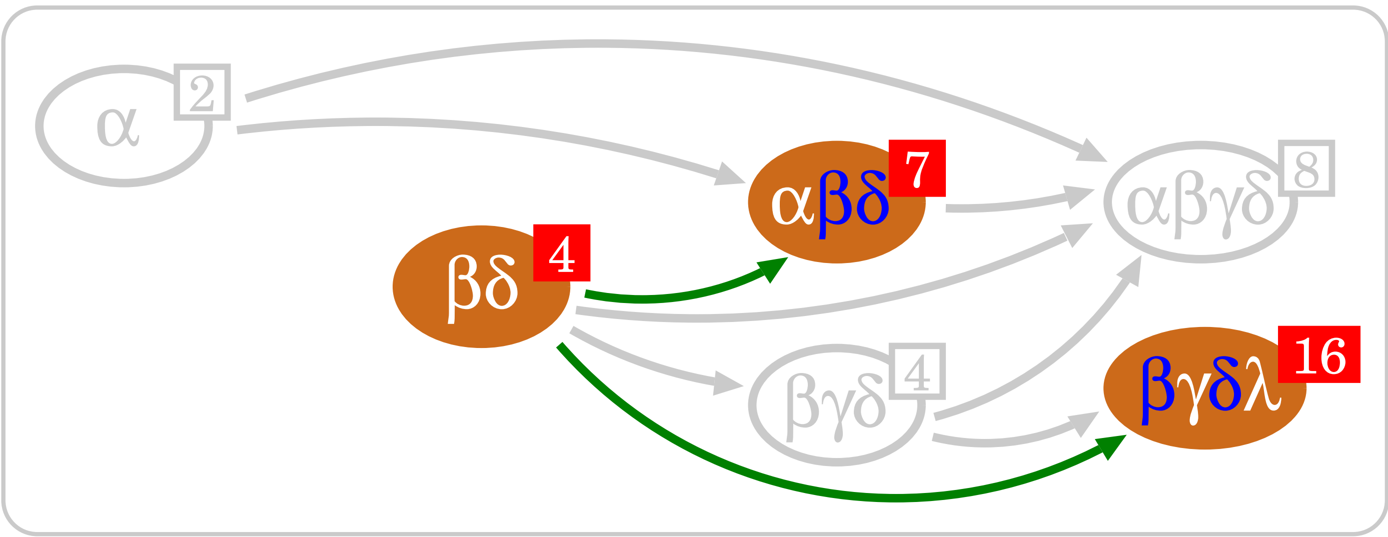
$MinScore = 8$: speculatively fire events γ and λ on MDD node p

Weighted score based speculation

Each workstation initializes a variable $MinScore$

- *Decrease* $MinScore$ whenever $SpecHitRate$ increase
- Speculatively fire event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying $u.ref/(|\mathcal{E}_u| - |\mathcal{E}_v|) \geq MinScore$

Example : MDD node p referencing pattern graph node encoding pattern $\{\beta, \delta\}$



$MinScore = 6$: speculatively fire events α, γ, and λ on MDD node p

Experimental Results and Conclusion

Flexible manufacturing system

- System with three machines to process three different types of parts : N is the number of each type of parts

W	**Time** (sec)					**Total Memory** (MB)				
	DISTR	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**	**DISTR**	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**
$N = 300$ $\lvert\mathcal{S}\rvert = 3.64 \cdot 10^{27}$						**SEQ** completes in 55 sec using 241MB				
2	79	-8%	-8%	-8%	-11%	243	$+12\%$	$+24\%$	$+3\%$	$+5\%$
4	91	${}^{d}+67\%$	-9%	-13%	-20%	243	$+102\%$	$+30\%$	$+11\%$	$+14\%$
8	260	-	-30%	-44%	-50%	243	-	$+42\%$	$+16\%$	$+22\%$
$N = 450$ $\lvert\mathcal{S}\rvert = 6.90 \cdot 10^{29}$						**SEQ** does not complete in 5 hrs using 512MB				
2	${}^{s}257$	${}^{s}+12\%$	${}^{s}-14\%$	${}^{s}-10\%$	${}^{s}-14\%$	826	$+16\%$	$+5\%$	$+4\%$	$+4\%$
4	${}^{d}311$	> 5hrs	${}^{d}-18\%$	${}^{d}-29\%$	${}^{d}-30\%$	826	-	$+33\%$	$+9\%$	$+17\%$
8	959	> 5hrs	-25%	-34%	-38%	826	-	$+61\%$	$+24\%$	$+39\%$

(W : number of workstations) (^{s} : memory swapping) (^{d} : dynamic memory load balancing)

- The best case for both **LENGTH** and **SCORE**
- Both outperform **HIST** in terms of runtime and memory consumption

Slotted ring network protocol

- Protocol for local area networks : N is the number of nodes within the network

W	**Time** (sec)					**Total Memory** (MB)				
	DISTR	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**	**DISTR**	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**
$N = 200$ $\lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$						**SEQ** completes in 108 sec using 284MB				
2	119	-24%	-13%	-5%	-8%	286	$+3\%$	$+45\%$	$+12\%$	$+14\%$
4	139	-27%	-15%	-6%	-9%	286	$+11\%$	$+51\%$	$+17\%$	$+26\%$
8	182	-32%	-24%	-14%	-20%	286	$+129\%$	$+62\%$	$+20\%$	$+29\%$
$N = 300$ $\lvert\mathcal{S}\rvert = 8.38 \cdot 10^{211}$						**SEQ** does not complete in 5 hrs using 512MB				
2	$^{s}552$	$^{s}{+5\%}$	$^{s}{-5\%}$	$^{s}{-10\%}$	$^{s}{-7\%}$	962	$+25\%$	$+11\%$	$+6\%$	$+10\%$
4	$^{d}490$	> 5hrs	$^{d}{-16\%}$	$^{d}{-18\%}$	$^{d}{-14\%}$	962	-	$+34\%$	$+13\%$	$+19\%$
8	564	> 5hrs	-39%	-24%	-30%	962	-	$+50\%$	$+21\%$	$+29\%$

(W : number of workstations) (^{s} : memory swapping) (^{d} : dynamic memory load balancing)

- Neither **LENGTH** nor **SCORE** outperforms **HIST** in terms of runtime
- Both consume less memory than **HIST**

Round robin mutex protocol

- Round robin mutual exclusion algorithm : N is the number of processes

W	Time (sec)					Total Memory (MB)				
	DISTR	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**	**DISTR**	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**
$N = 800$ $\lvert\mathcal{S}\rvert = 1.20 \cdot 10^{196}$						**SEQ** completes in 27 sec using 290MB				
2	29	+37%	+6%	+0%	+0%	293	+110%	+85%	+21%	+27%
4	36	+33%	+8%	+0%	+0%	293	+348%	+109%	+28%	+37%
8	51	+33%	+5%	+0%	+0%	293	+807%	+148%	+36%	+49%
$N = 1100$ $\lvert\mathcal{S}\rvert = 3.36 \cdot 10^{334}$						**SEQ** does not complete in 5 hrs using 512MB				
2	$^{d}65$	$^{s}+62\%$	$^{s}+18\%$	$^{d}+5\%$	$^{d}+9\%$	794	+46%	+6%	+2%	+3%
4	47	$^{s}+131\%$	$^{d}+10\%$	+2%	+2%	794	+119%	+38%	+3%	+5%
8	56	$^{d}+164\%$	+7%	+2%	+2%	794	+299%	+50%	+9%	+10%

(W : number of workstations) (^{s} : memory swapping) (^{d} : dynamic memory load balancing)

- The worst case for all four speculation approaches
- Either **LENGTH** or **SCORE** exhibits only a small worsening of their runtimes
- Both control well the memory overhead of speculation

Runway monitoring system

- Avionics system : monitors T targets with S speeds on a $X \times Y \times Z$ runway

W	Time (sec)					Total Memory (MB)				
	DISTR	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**	**DISTR**	**NAÏVE**	**HIST**	**LENGTH**	**SCORE**
$Z = 2 \quad \lvert\mathcal{S}\rvert = 1.51 \cdot 10^{15}$					**SEQ** completes in 236 sec using 314MB					
2	731	> 10hrs	-2%	-3%	-5%	332	-	$+39\%$	$+4\%$	$+9\%$
4	938	> 10hrs	-8%	-16%	-18%	332	-	$+88\%$	$+20\%$	$+28\%$
8	1480	> 10hrs	-22%	-27%	-31%	332	-	$+128\%$	$+67\%$	$+90\%$
$Z = 3 \quad \lvert\mathcal{S}\rvert = 5.07 \cdot 10^{15}$					**SEQ** does not complete in 10 hrs using 512MB					
2	$^{s}11280$	> 10hrs	$^{s}-1\%$	$^{s}-2\%$	$^{s}-3\%$	962	-	$+10\%$	$+3\%$	$+4\%$
4	$^{d}9762$	> 10hrs	$^{d}-15\%$	$^{d}-19\%$	$^{d}-26\%$	962	-	$+31\%$	$+4\%$	$+13\%$
8	$^{d}14101$	> 10hrs	$^{d}-17\%$	$^{d}-29\%$	$^{d}-31\%$	962	-	$+58\%$	$+8\%$	$+35\%$

(W : number of workstations) (^{s} : memory swapping) (^{d} : dynamic memory load balancing)

- Both **LENGTH** and **SCORE** work on a real application

Conclusions and future work

- **Conclusions** :
  - We have showed the potential of low-overhead firing speculation to accelerate distributed state-space generation
  - Both **LENGTH** and **SCORE** heuristics work not only on theoretical models but also on some real world applications

- **Future research directions** :
  - We are applying the speculative firing idea to distributed state-space generation on an SMP architecture
  - We plan to apply this idea to symbolic distributed CTL model checking

Thank You!