Distributed Symbolic Reachability Analysis

Ming-Ying Chung

Department of Computer Science and Engineering
University of California, Riverside

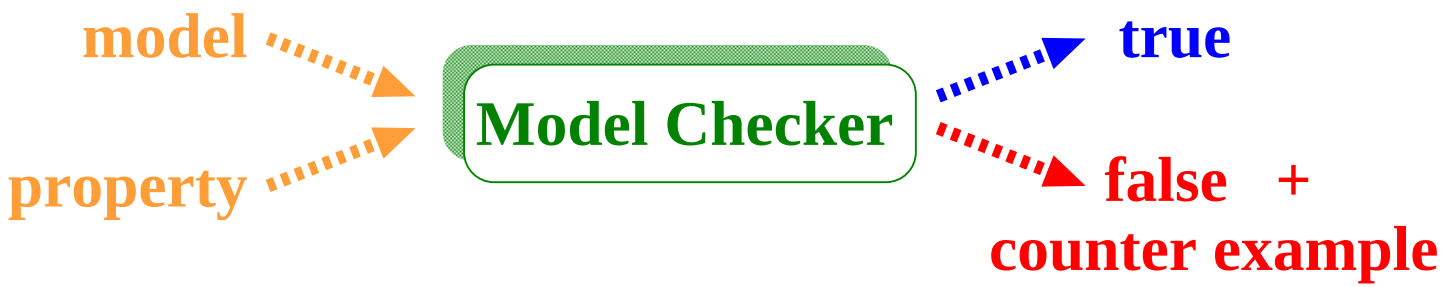
Dissertation Committee: **Dr. Gianfranco Ciardo** **Chair**
Dr. Tevfik Bultan
Dr. Harry Hsieh

December 13th, 2007

Formal Verification

- Mechanisms for assuring the quality of hardware or software designs:
  - Simulation (incomplete).
  - Testing (incomplete).
  - Formal verification (complete):

    model checking, theorem proving, equivalence checking, etc.



- Features of model checking: **completeness**, **counter example(s)**.

Motivation

Motivation I - Intel Pentium FDIV Bug

- Faulty implementation of FDIV was used in Pentium 60/90MHz processors (disclosed by Dr. Thomas Nicely in Oct. 1994).
- Certain floating point division operations resulted incorrectly:
  $\frac{5505001}{294911} = 18.66600093$ **which should be** 18.6665197.
- Some missing entries in the lookup table used by the FDIV implementation.
- Intel replaced all defective processors in Dec. 1994:
  costing 475 million dollars.

Motivation II - ESA Ariane 5 Failure

- ESA lauched their Ariane 5 rocket for the first time in Jun. 1996.
- The rocket exploded 37 seconds after its lift-off from French Guiana.
- Software bug in the inertial reference system:

  a 64-bit floating point number was converted to a 16-bit signed integer.
- The number never become larger than $2^{15}-1$ during any Ariane launch.
- **This failure cost ESA 500 million dollars and ten years of development**.

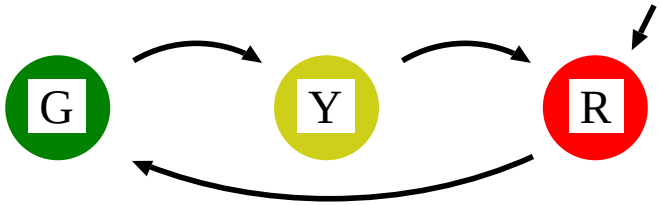
Model Checking

E. Clarke and E. Emerson. *Synthesis of synchronization skeletons for branching time temporal logic*. Logic of Programs 1981.

J.-P. Queille and J. Sifakis. *Specification and verification of concurrent systems in CESAR*. Symposium on Programming 1982

Modelling

- US traffic light described by a state transition graph.



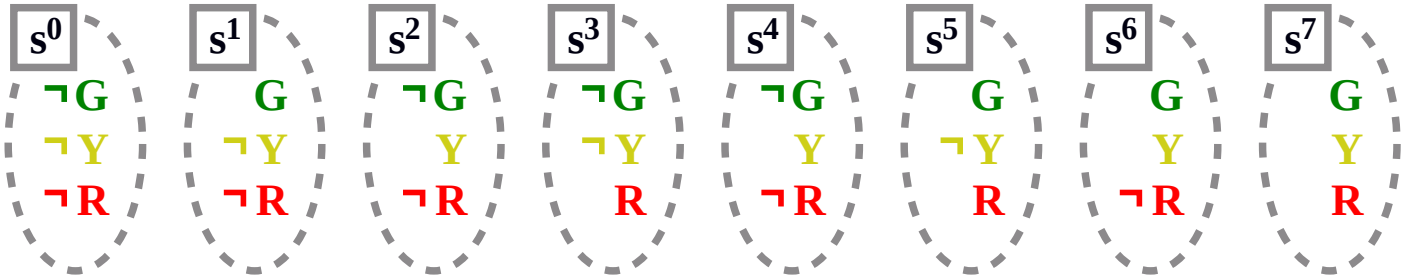
- The **Kripke structure**: $M = (\widehat{\mathcal{S}}, \mathbf{s}^{init}, \mathcal{R}, \mathcal{L})$

$\widehat{\mathcal{S}} = \{\mathbf{s}^0, \mathbf{s}^1, \mathbf{s}^2, \mathbf{s}^3, \mathbf{s}^4, \mathbf{s}^5, \mathbf{s}^6, \mathbf{s}^7\}$

$\mathbf{s}^{init} = \mathbf{s}^3$

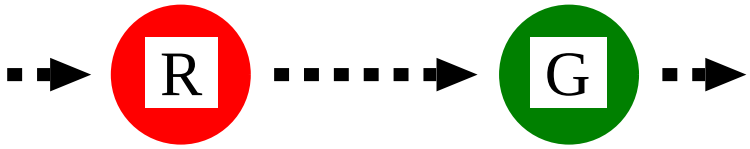
$\mathcal{R} = \{\langle \mathbf{s}^1 \mathbf{s}^2 \rangle, \langle \mathbf{s}^2 \mathbf{s}^3 \rangle, \langle \mathbf{s}^3 \mathbf{s}^1 \rangle\}$

$\mathcal{L} = \{\langle \neg\mathbf{G}, \neg\mathbf{Y}, \neg\mathbf{R} \rangle_{\mathbf{s}^0}, \langle \mathbf{G}, \neg\mathbf{Y}, \neg\mathbf{R} \rangle_{\mathbf{s}^1}, \langle \neg\mathbf{G}, \mathbf{Y}, \neg\mathbf{R} \rangle_{\mathbf{s}^2}, \langle \neg\mathbf{G}, \neg\mathbf{Y}, \mathbf{R} \rangle_{\mathbf{s}^3}, \langle \neg\mathbf{G}, \mathbf{Y}, \mathbf{R} \rangle_{\mathbf{s}^4}, \langle \mathbf{G}, \neg\mathbf{Y}, \mathbf{R} \rangle_{\mathbf{s}^5}, \langle \mathbf{G}, \mathbf{Y}, \neg\mathbf{R} \rangle_{\mathbf{s}^6}, \langle \mathbf{G}, \mathbf{Y}, \mathbf{R} \rangle_{\mathbf{s}^7}\}$



Specification

- Whenever we encounter a **RED** light, we will see a **GREEN** light eventually.



- Desired properties can be expressed in **temporal logics**:
  - **Computational Tree Logic** (**CTL**):

    AG(**R** $\longrightarrow$ **AF G**) $=$ $\neg$ **E**(true **U** (**R** $\wedge$ **EG** $\neg$ **G**)

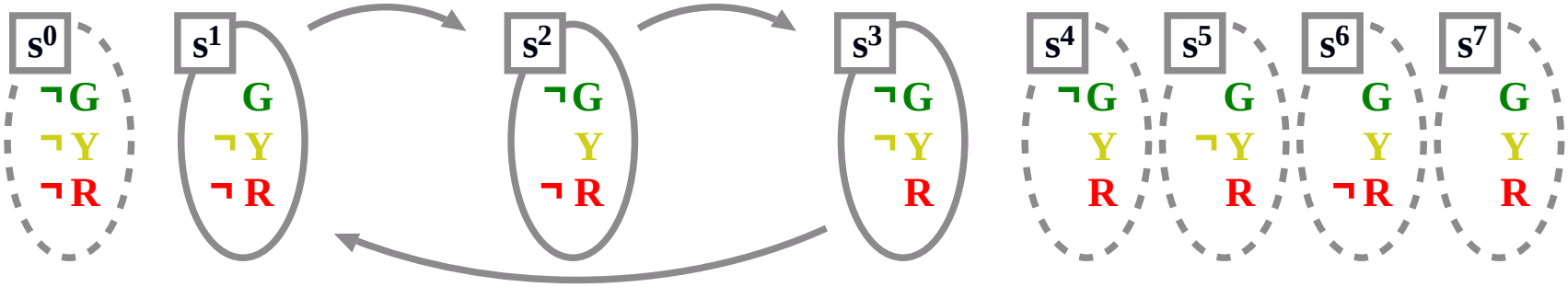
  - **Linear-time Temporal Logic** (**LTL**):

    G(**R** $\longrightarrow$ **F G**)

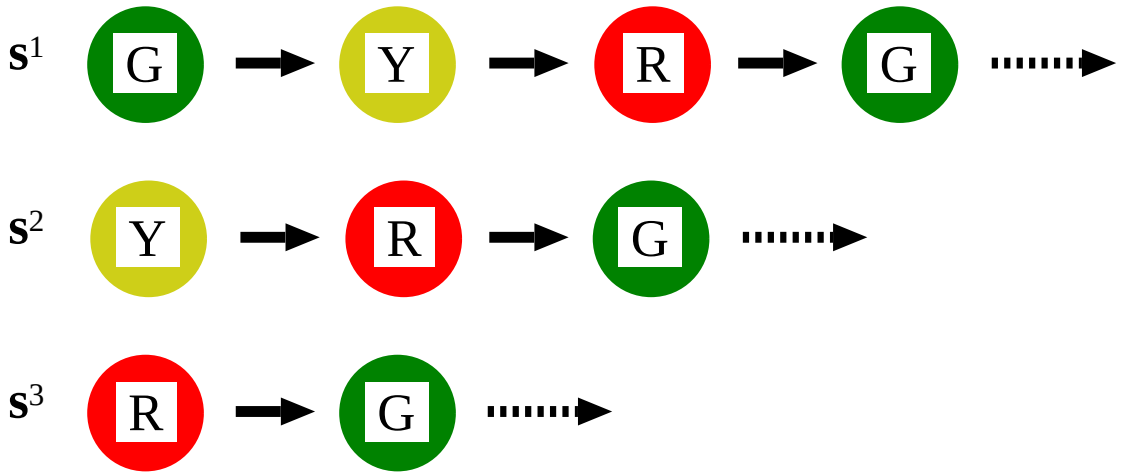
Verification

Looking into the system to see whether the property hold forever:

(1) generating the state space of the system, $\mathcal{S} = \{\mathrm{s}^1, \mathrm{s}^2, \mathrm{s}^3\}$.



(2) checking whether the desired property hold in each state.

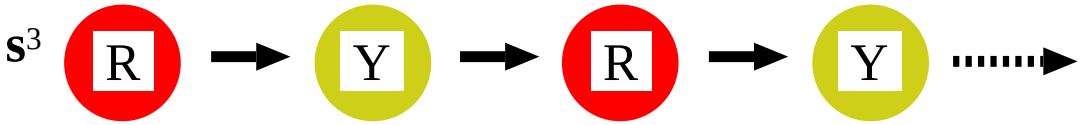


Verifying a Redesigned System

- We now redefine the purpose of **YELLOW** so that it can be used to **notify drivers that it is time to stop or move their cars**.

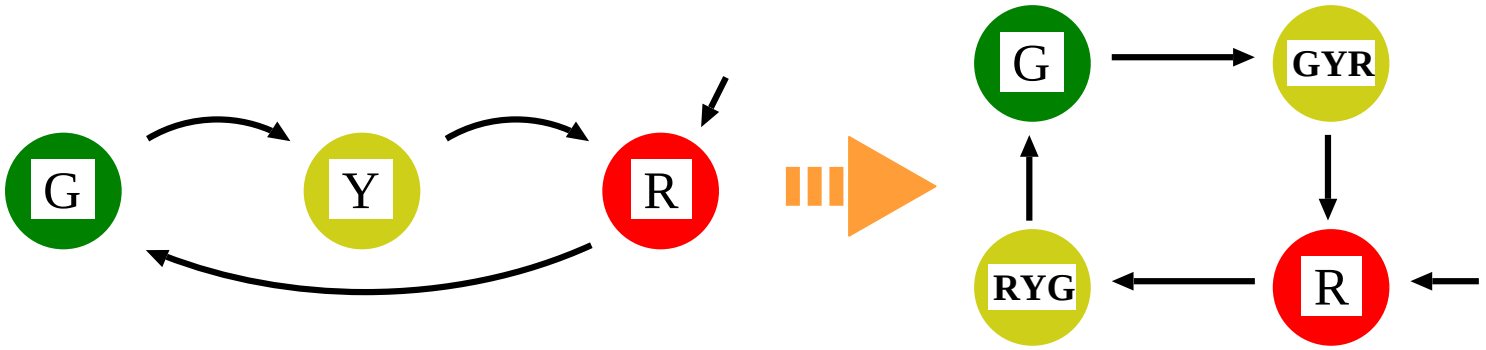
- Can we always see a **GREEN** eventually whenever encountering a **RED**?

  NO!



Verifying a Redesigned System

- Correct redesign should be as follow.



- Can we always see a **GREEN** eventually whenever encountering a **RED**?

  YES!

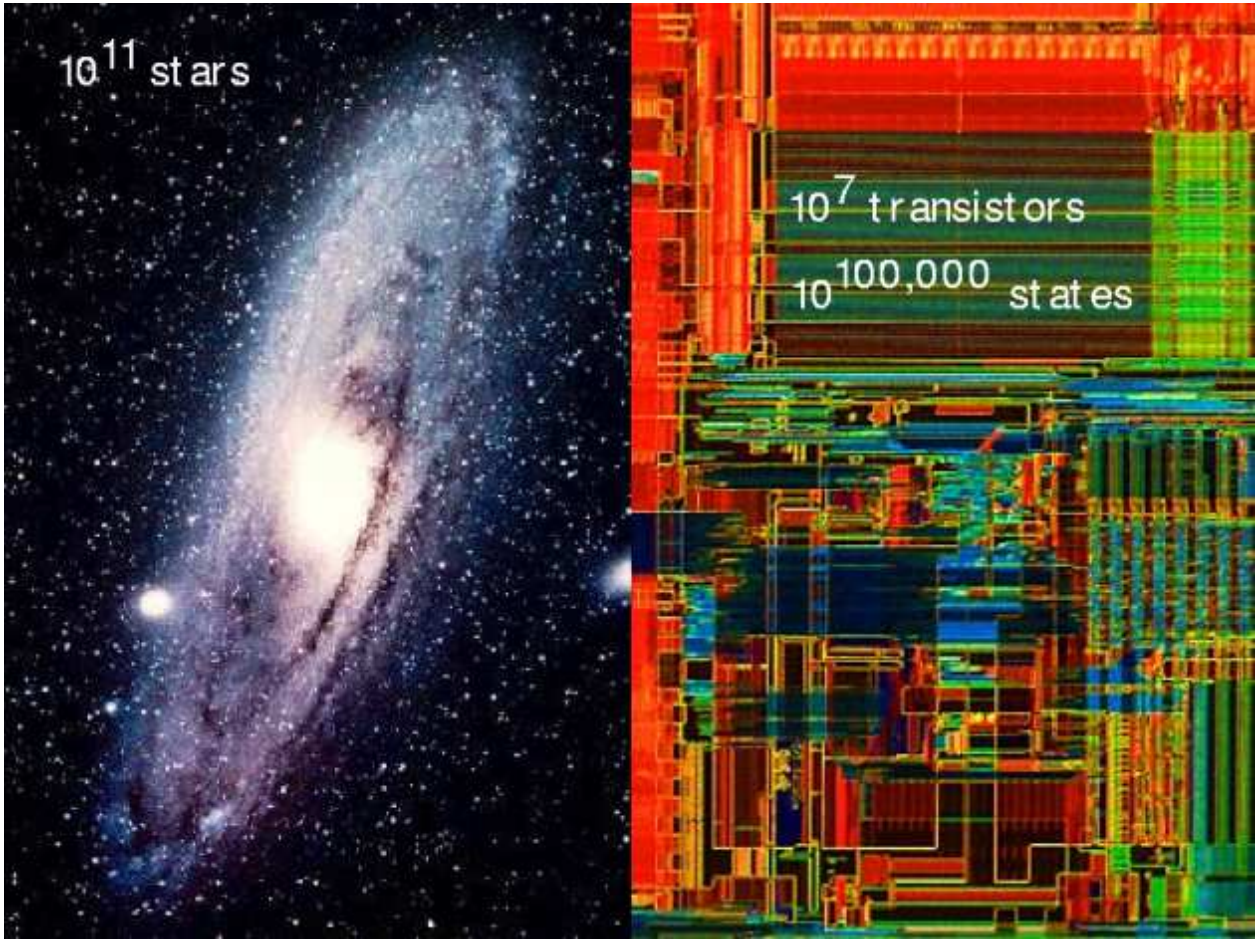
- Traffic light in Germany.

Bottleneck - State-Space Explosion

- **State-space generation** (**reachability analysis**):

  the first step in model checking.

- **Very memory-intensive**.



Symbolic Reachability Analysis

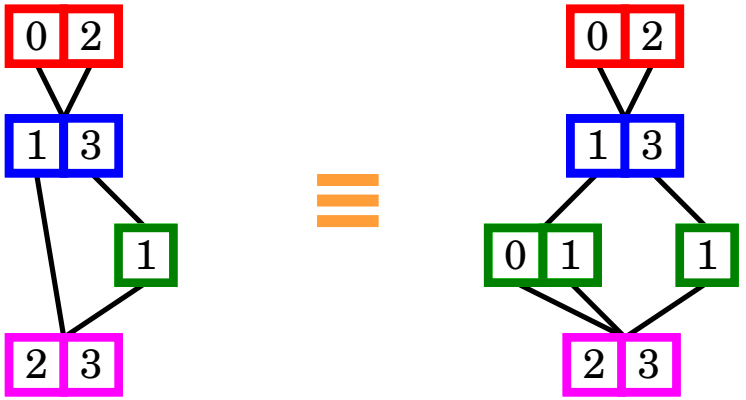
R. Bryant. *Graph-based algorithms for boolean function manipulation*. IEEE Trans 1986.

K. McMillan. *The SMV system - Symbolic Model Checking*, Carnegie Mellon University TR 1992.

Quasi-Reduced Ordered MDDs

- Fully-reduced MDD: nodes are **canonical**.
- Quasi-reduced MDD: nodes are **canonical level-wise**.
- Canonicity allows nodes to show the state encoding and computation.

$$\mathcal{S}_4 = \{0, 1, 2\}$$
$$\mathcal{S}_3 = \{0, 1, 2, 3\}$$
$$\mathcal{S}_2 = \{0, 1\}$$
$$\mathcal{S}_1 = \{0, 1, 2, 3\}$$



$$\mathcal{X} = \left\{ \begin{array}{cccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 3 & 3 & 1 & 1 & 1 & 1 & 3 & 3 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 \end{array} \right\}$$

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

Quasi-Reduced Ordered MDDs

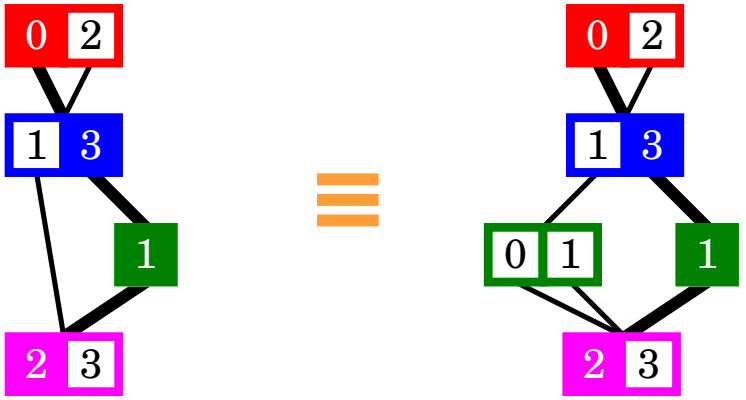
- Fully-reduced MDD: nodes are **canonical**.
- Quasi-reduced MDD: nodes are **canonical level-wise**.
- Canonicity allows nodes to show the state encoding and computation.

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

$\mathcal{S}_3 = \{0, 1, 2, 3\}$

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

$\mathcal{S}_1 = \{0, 1, 2, 3\}$

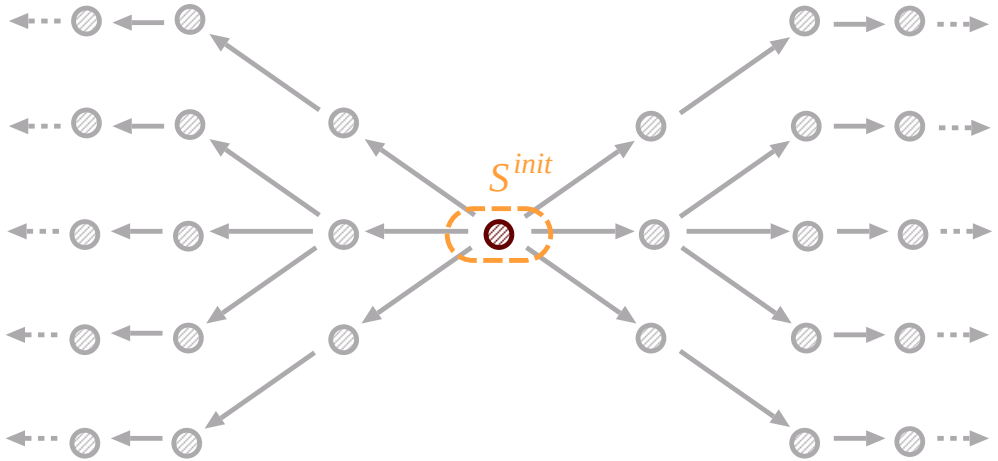


$$\mathcal{X} = \left\{ \begin{array}{cccc|c|cccccc} 0 & 0 & 0 & 0 & (0) & 0 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & (3) & 3 & 1 & 1 & 1 & 1 & 3 & 3 \\ 0 & 0 & 1 & 1 & (1) & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 2 & 3 & 2 & 3 & (2) & 3 & 2 & 3 & 2 & 3 & 2 & 3 \end{array} \right\}$$

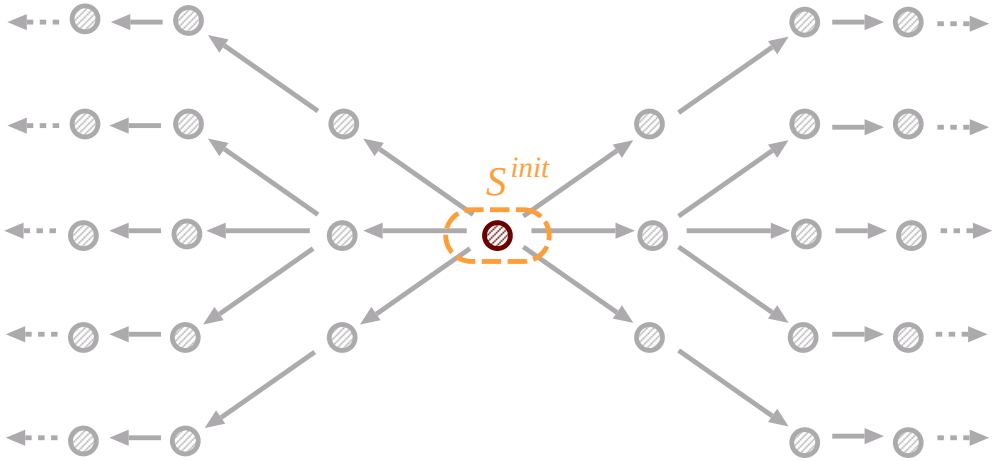
T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Multi-valued decision diagrams: theory and applications*, Journal of multiple-valued logic 1998

Explicit vs. Symbolic Reachability Analysis

- Explicit approach:

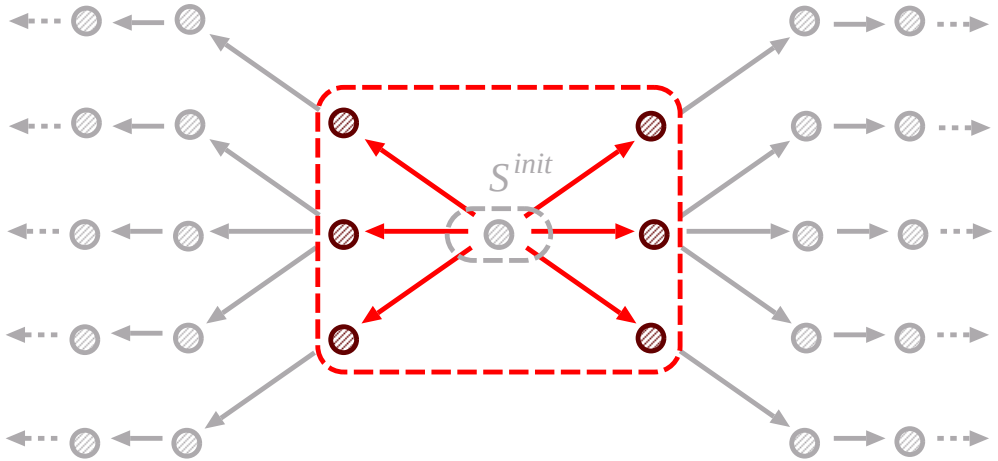


- Symbolic approach:

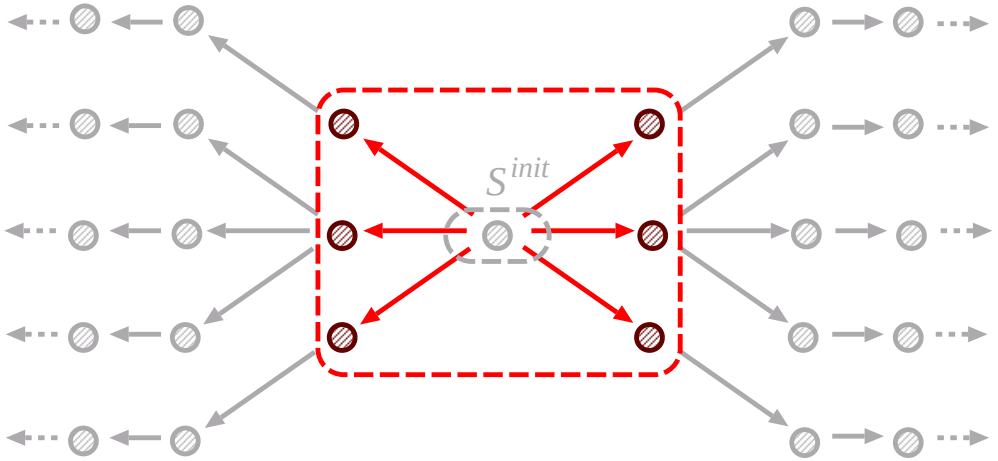


Explicit vs. Symbolic Reachability Analysis

- Explicit approach:

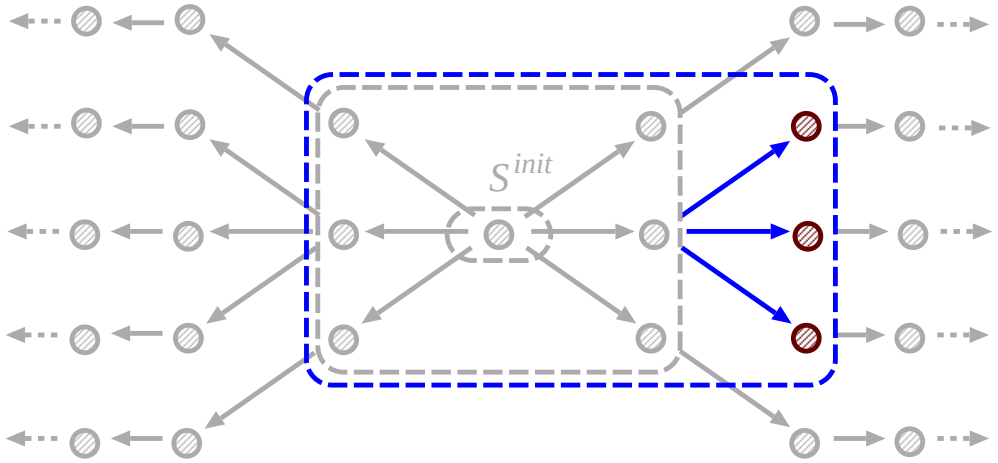


- Symbolic approach:

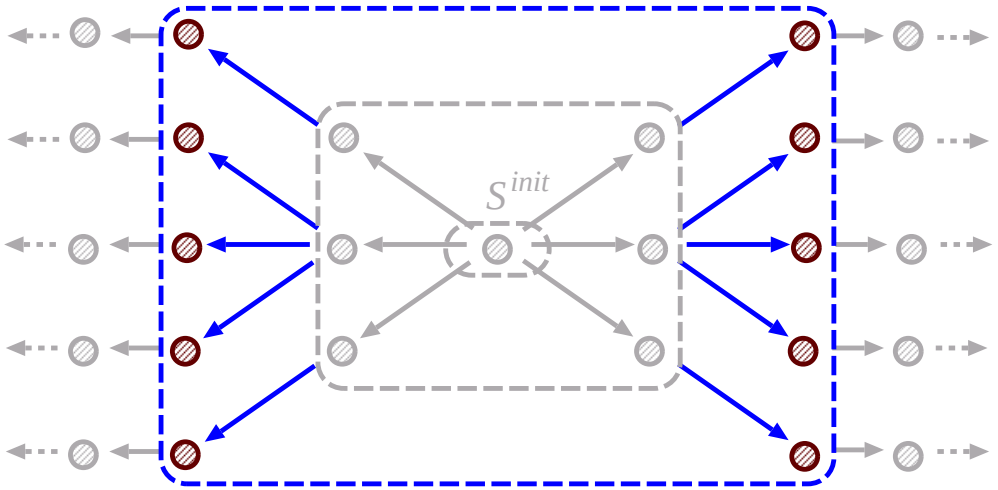


Explicit vs. Symbolic Reachability Analysis

- Explicit approach: **from one state at a time**.

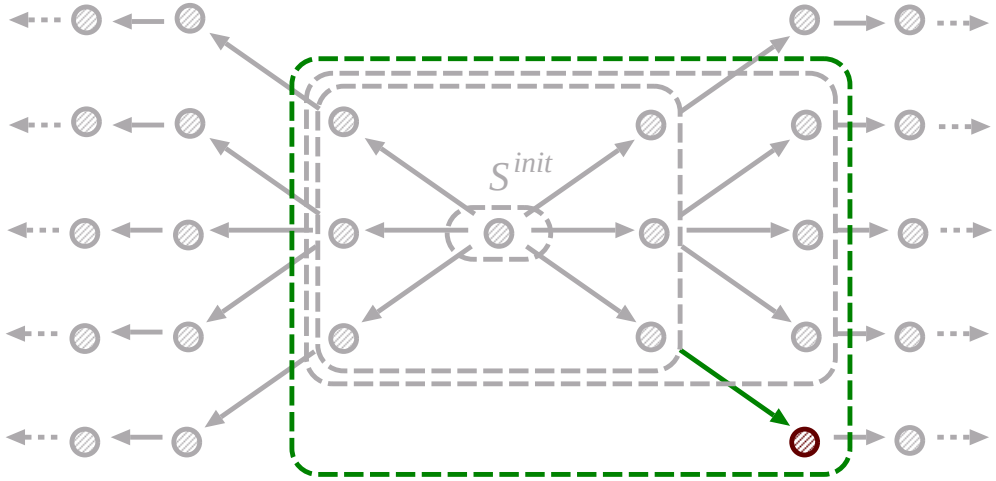


- Symbolic approach: **from a set of states at a time** (**image computation**).

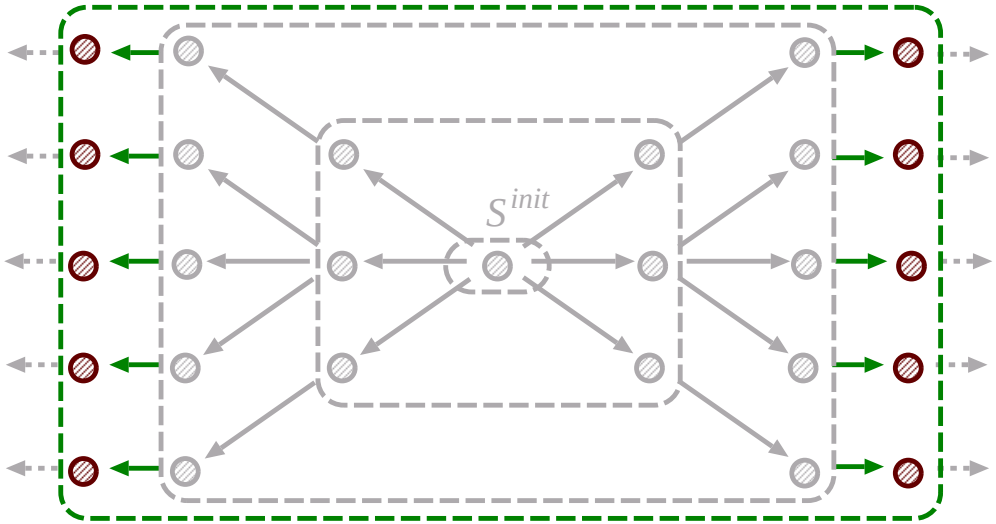


Explicit vs. Symbolic Reachability Analysis

- Explicit approach: from one state at a time.



- Symbolic approach: from a set of states at a time (image computation).



To Distribute the Memory Load of Symbolic Reachability Analysis

Chapter 3

Ming-Ying Chung and Gianfranco Ciardo.

Saturation NOW. QEST 2004.

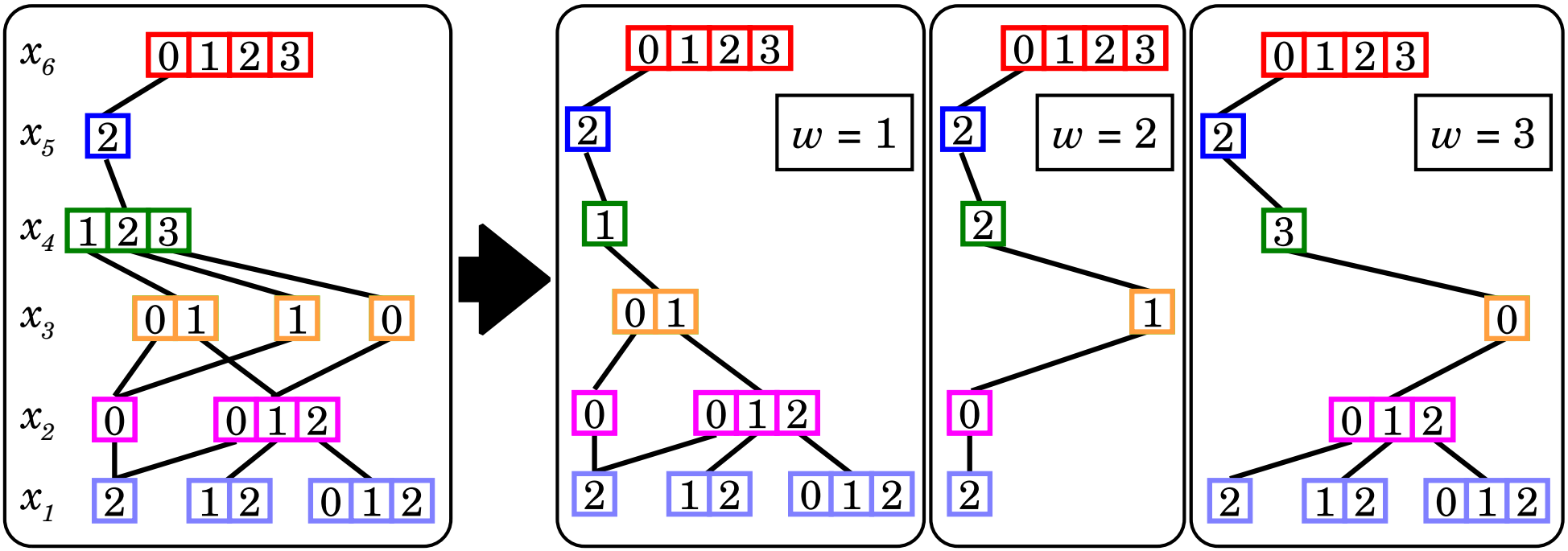
Chapter 4

Ming-Ying Chung, Gianfranco Ciardo, and Radu Siminiceanu.

Caching, hashing, and garbage collection for distributed state-space construction. PDMC 2007.

Vertical Image Slicing

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



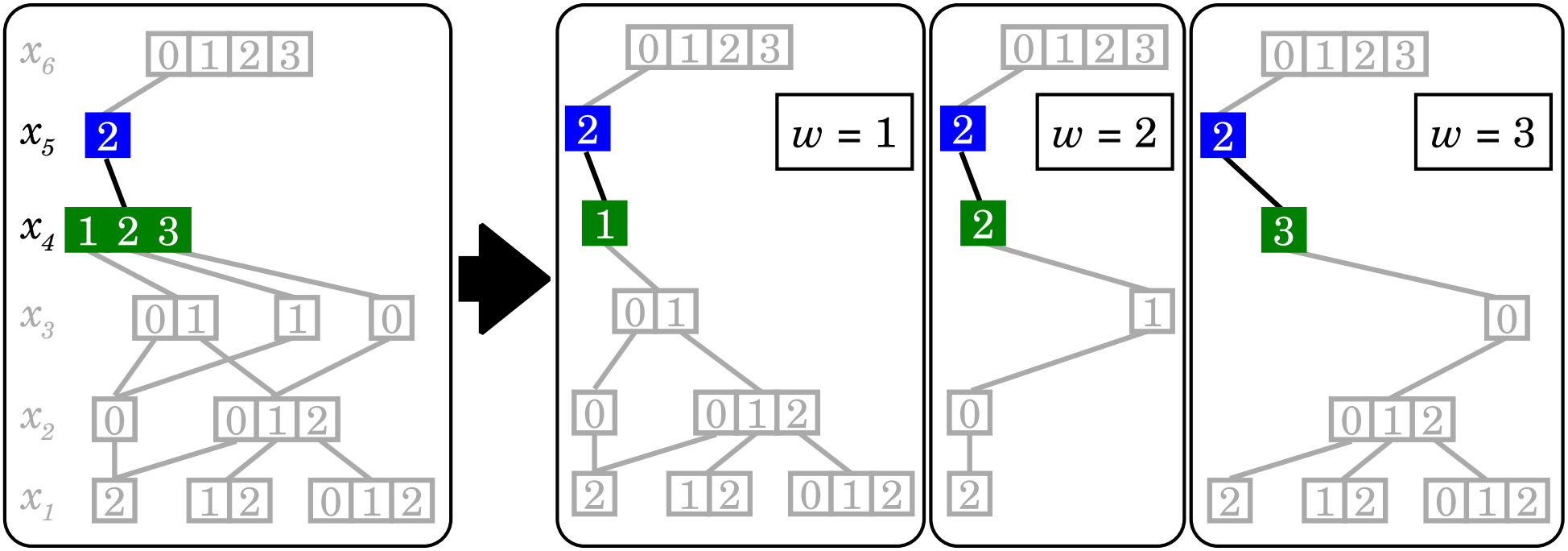
T. Heyman, D. Geist, O. Grumberg, T. Heyman, and A. Schuster, *Achieving scalability in parallel reachability analysis of very large circuits*, CAV 2000

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

O. Grumberg, T. Heyman, N. Ifergan, and A. Schuster, *Achieving speedups in distributed symbolic reachability analysis through asynchronous computation, CHARME 2005*

Slicing Variables

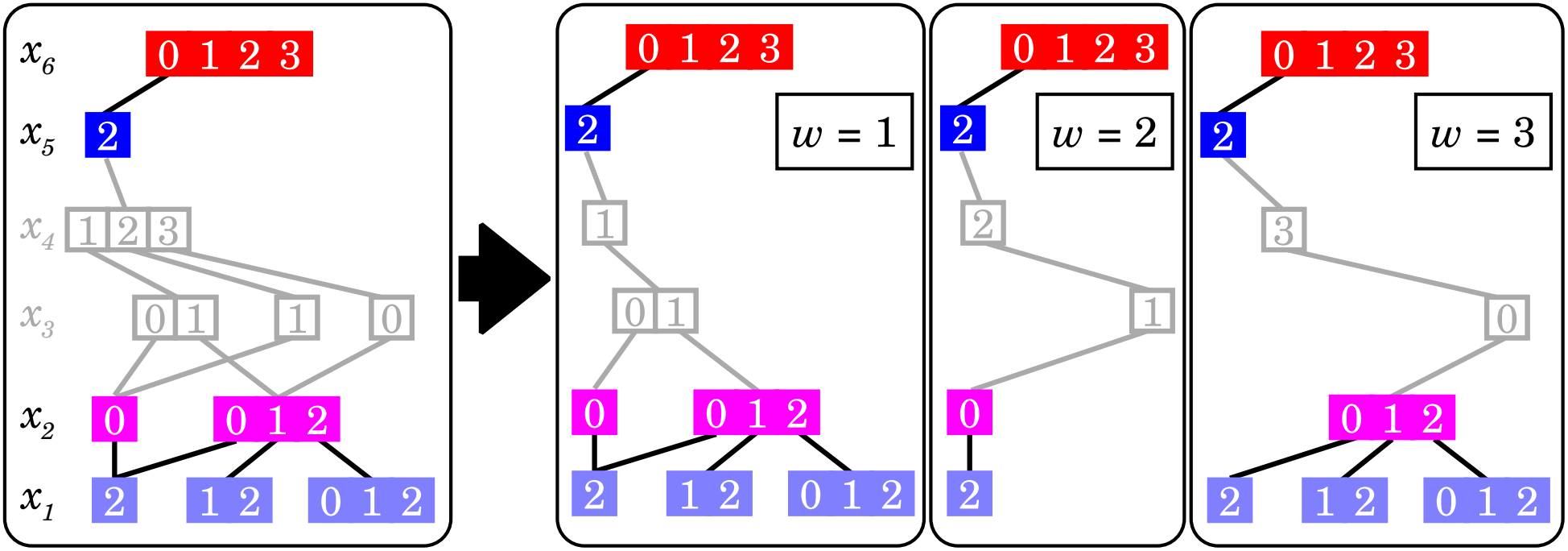
Job-based slicing: $\mathcal{S} = \mathcal{S}^1 \cup \mathcal{S}^2 \cup \mathcal{S}^3$ $\mathcal{S}^i \cap \mathcal{S}^j = \emptyset$ **for any** $i \neq j$



- **Selecting variables to slice an image is not trivial** (reodering).
- Duplicate MDD nodes and image computation. (synchronization): scalability then becomes an issue.

Duplicate Work

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 any $i \neq j$

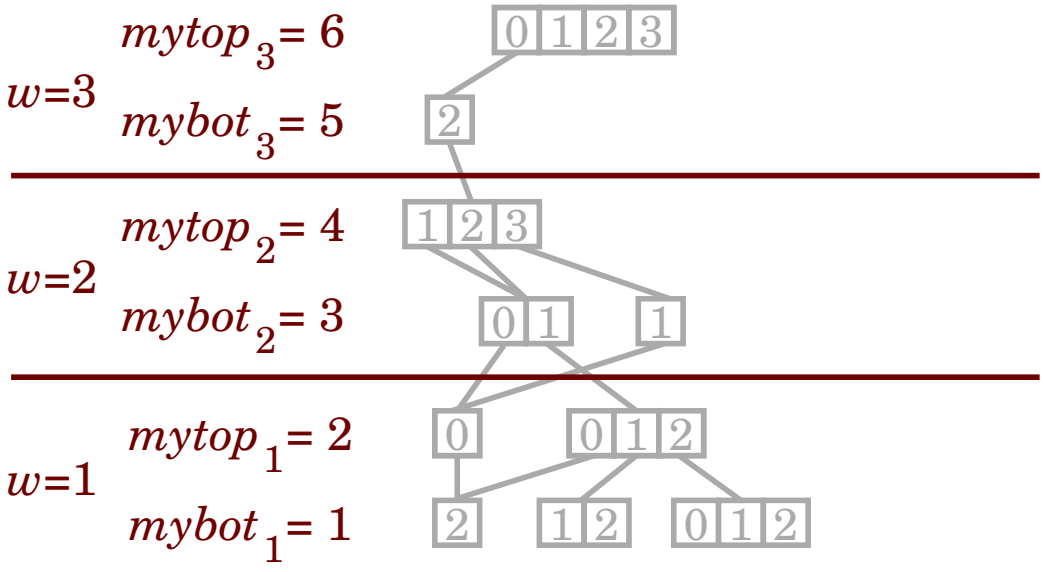


- Selecting variables to slice an image is not trivial (reodering).
- **Duplicate MDD nodes and image computation.** (synchronization): scalability then becomes an issue.

Horizontal Image Slicing

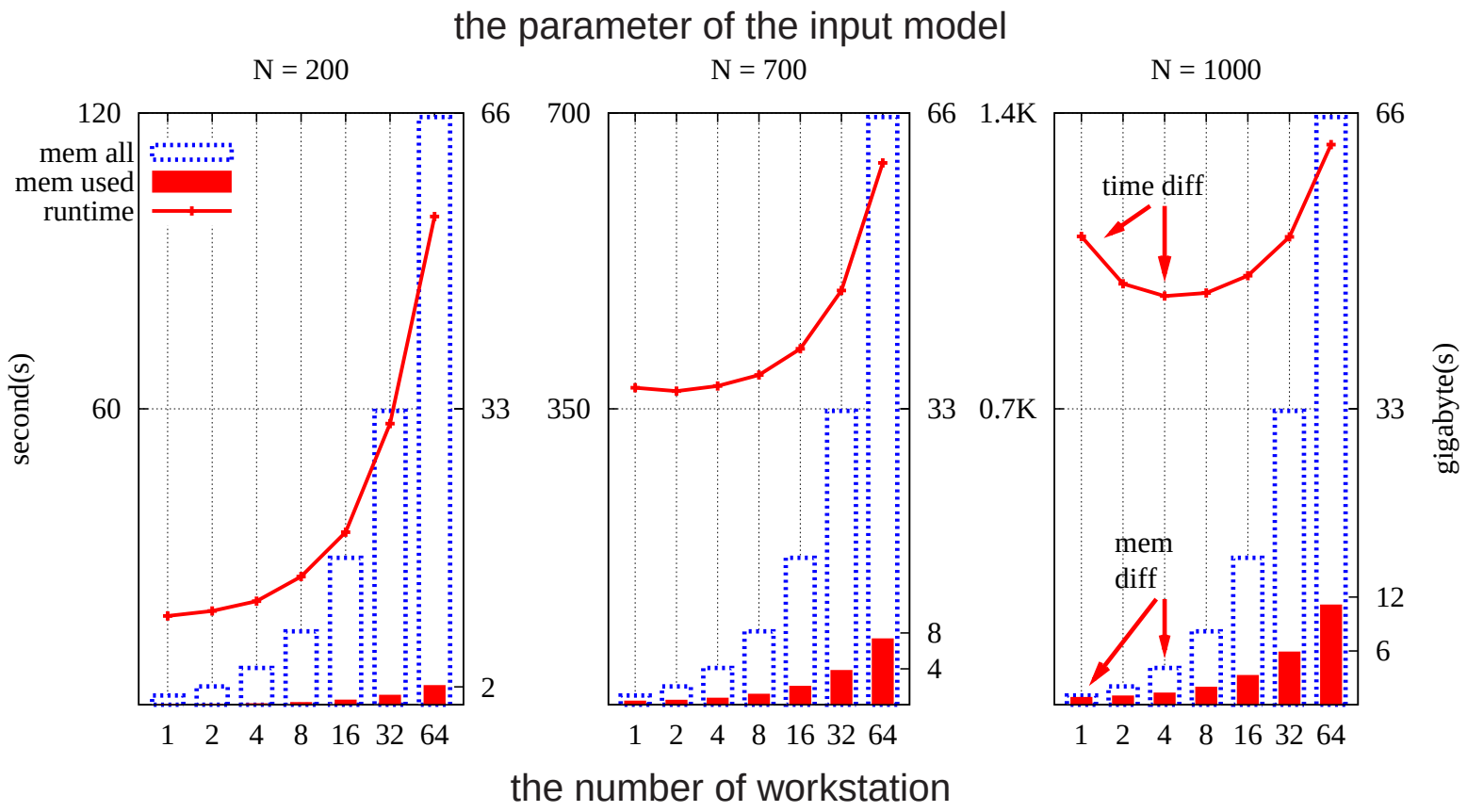
- **Level-based slicing**:
  - The node canonicity can be maintained without message passing.
  - Duplicate work (synchronization among workstations) is avoided.

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



- Issues:
  - **Difficult to perform a good memory load balancing** (Chapter 3).
  - **Sequentialized computation is hard to parallelize** (Chapters 5 and 6).

Experiment - Round Robin Mutex Protocol



- Message-passing overhead is not trivial:

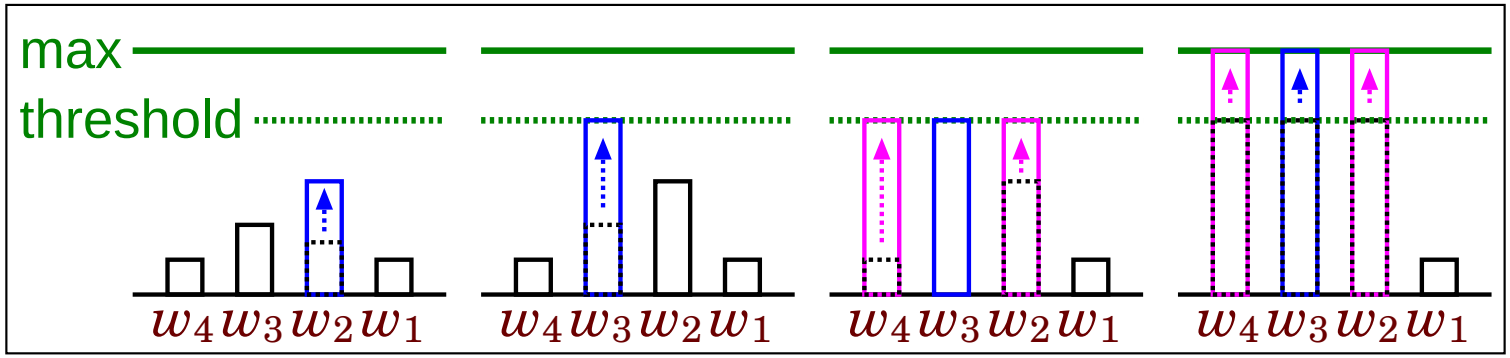
  runtime difference diminishes as the model size grows.

- W_{opt} cannot be known a priori:

  the penalty when $W < W_{opt}$ is larger than when $W > W_{opt}$.

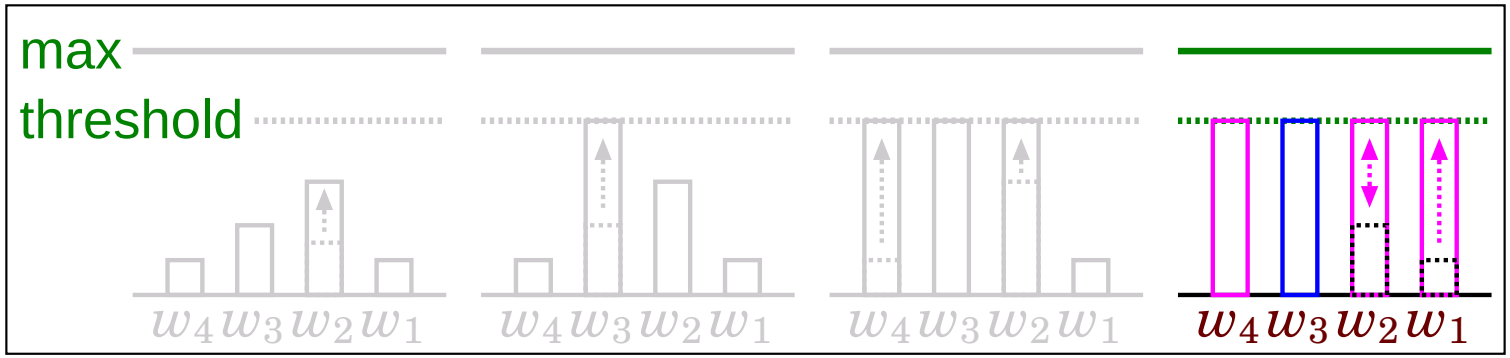
Dynamic Memory Load Balancing (DMLB)

- Pairwise DMLB might not utilize the overall NOW memory well.

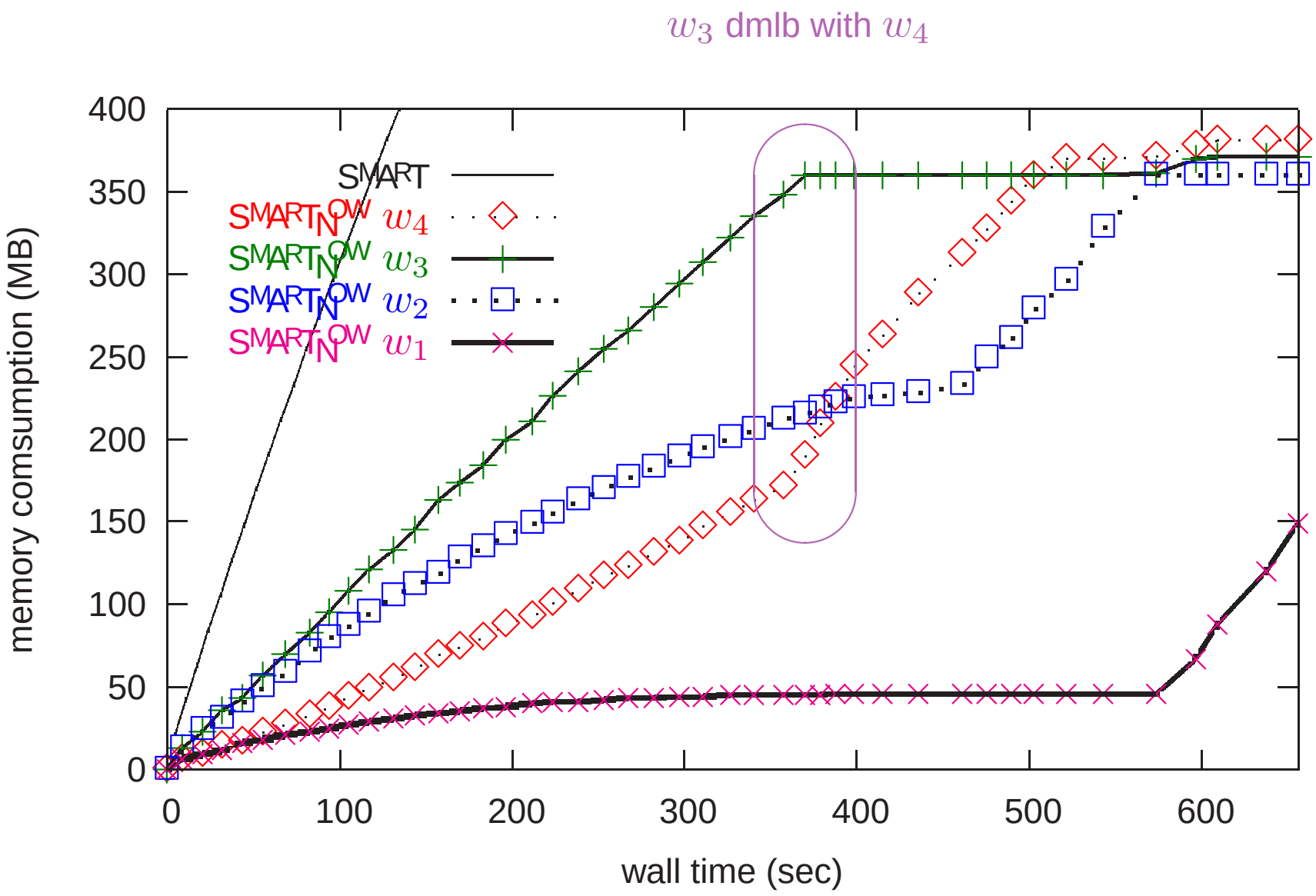


- **Nested DMLB scheme**:

  workstations freeze image computation and propagate DMLB.



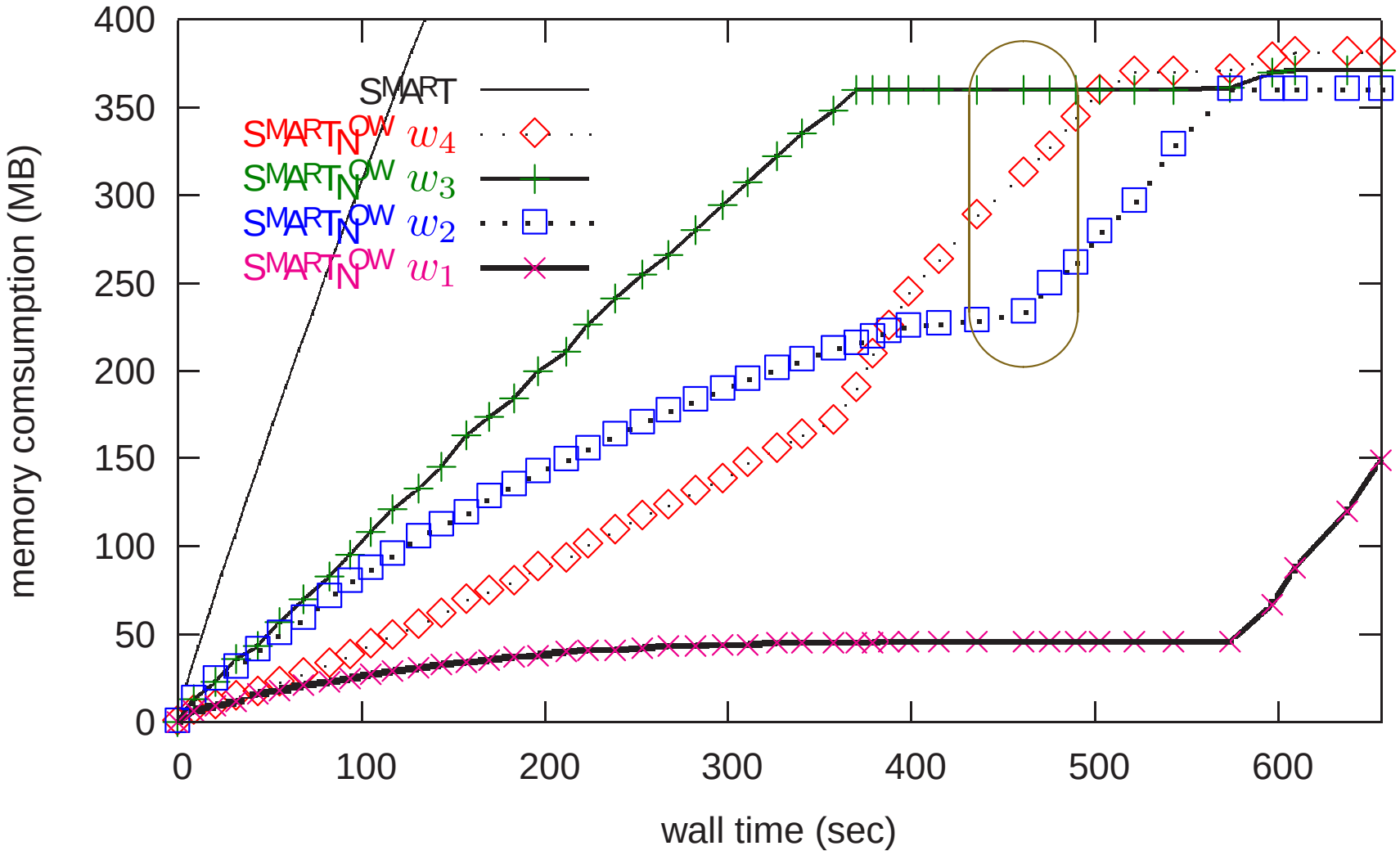
Experiment - Slotted Ring Network Protocol



w_3 DMLB with the workstation below it.

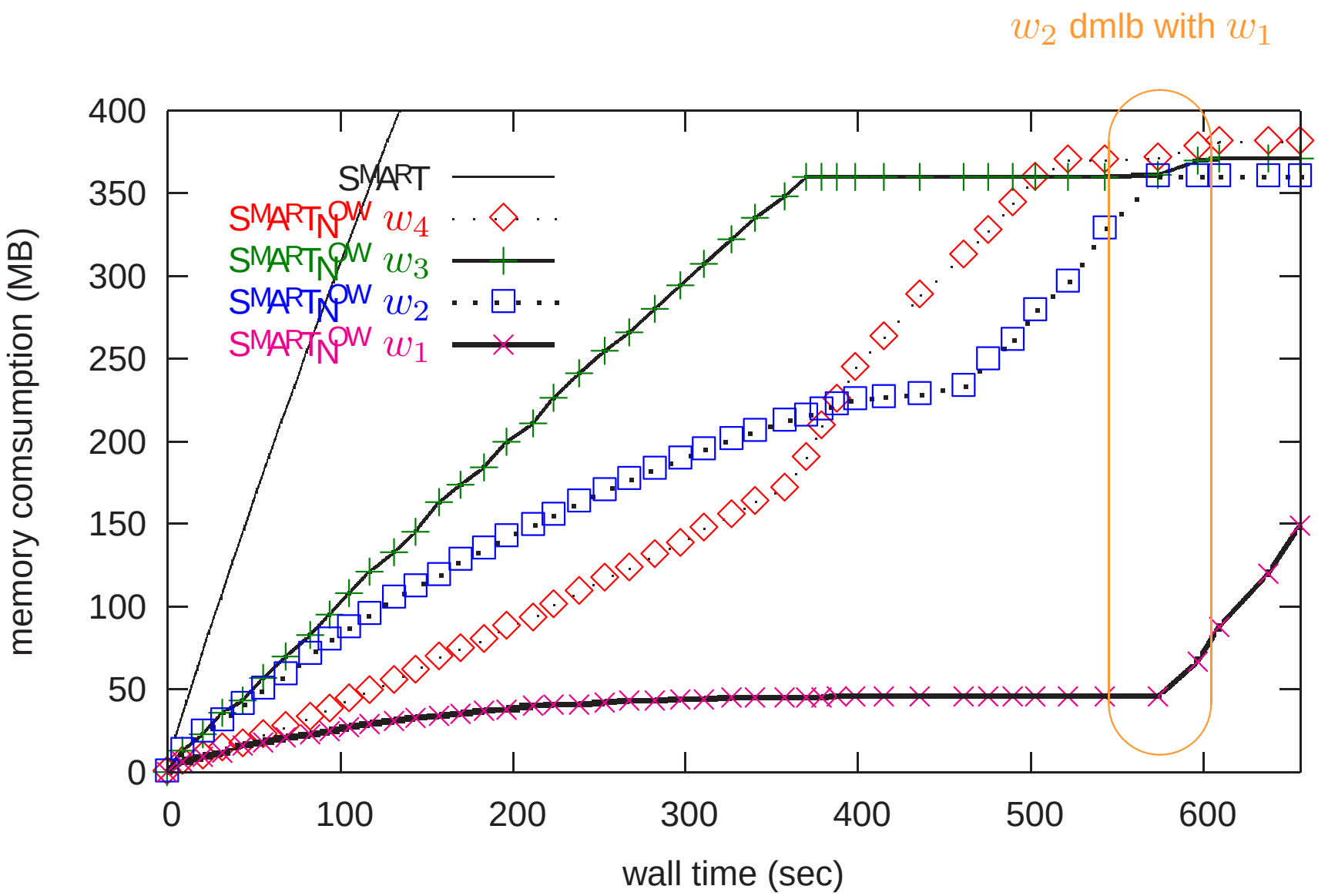
Experiment - Slotted Ring Network Protocol

w_3 also dmlb with w_2



w_3 DMLB with the workstation above it.

Experiment - Slotted Ring Network Protocol



w_3 asks the workstation below it to propagate the DMLB.

To Speedup the Computation of Distributed Symbolic Reachability Analysis

Chapter 5

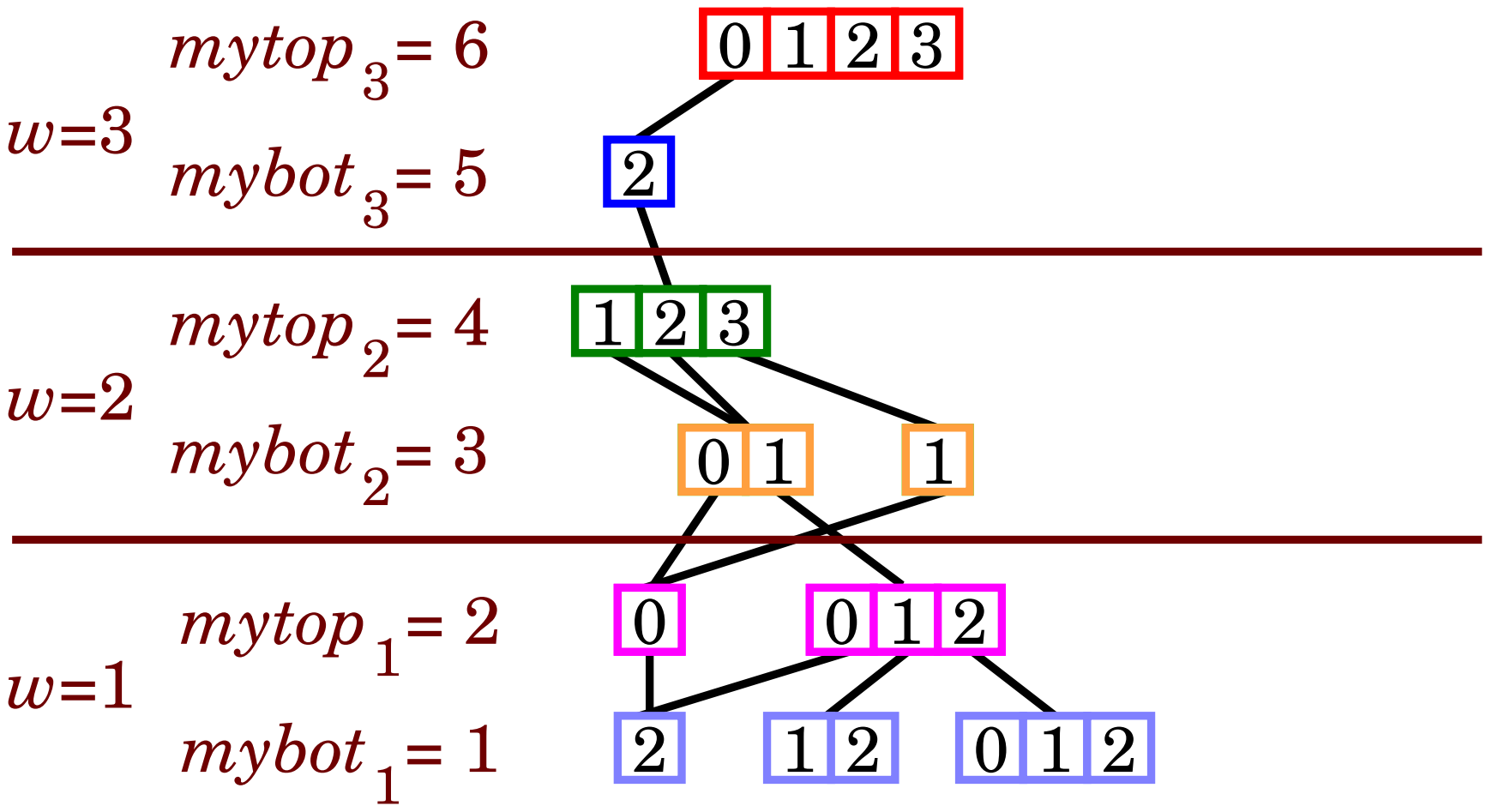
Ming-Ying Chung and Gianfranco Ciardo.
A pattern recognition approach for speculative firing prediction in distributed Saturation state-space generation. PDMC 2005.

Chapter 6

Ming-Ying Chung and Gianfranco Ciardo.
A dynamic firing speculation to speedup distributed symbolic state-space generation. IPDPS 2006.

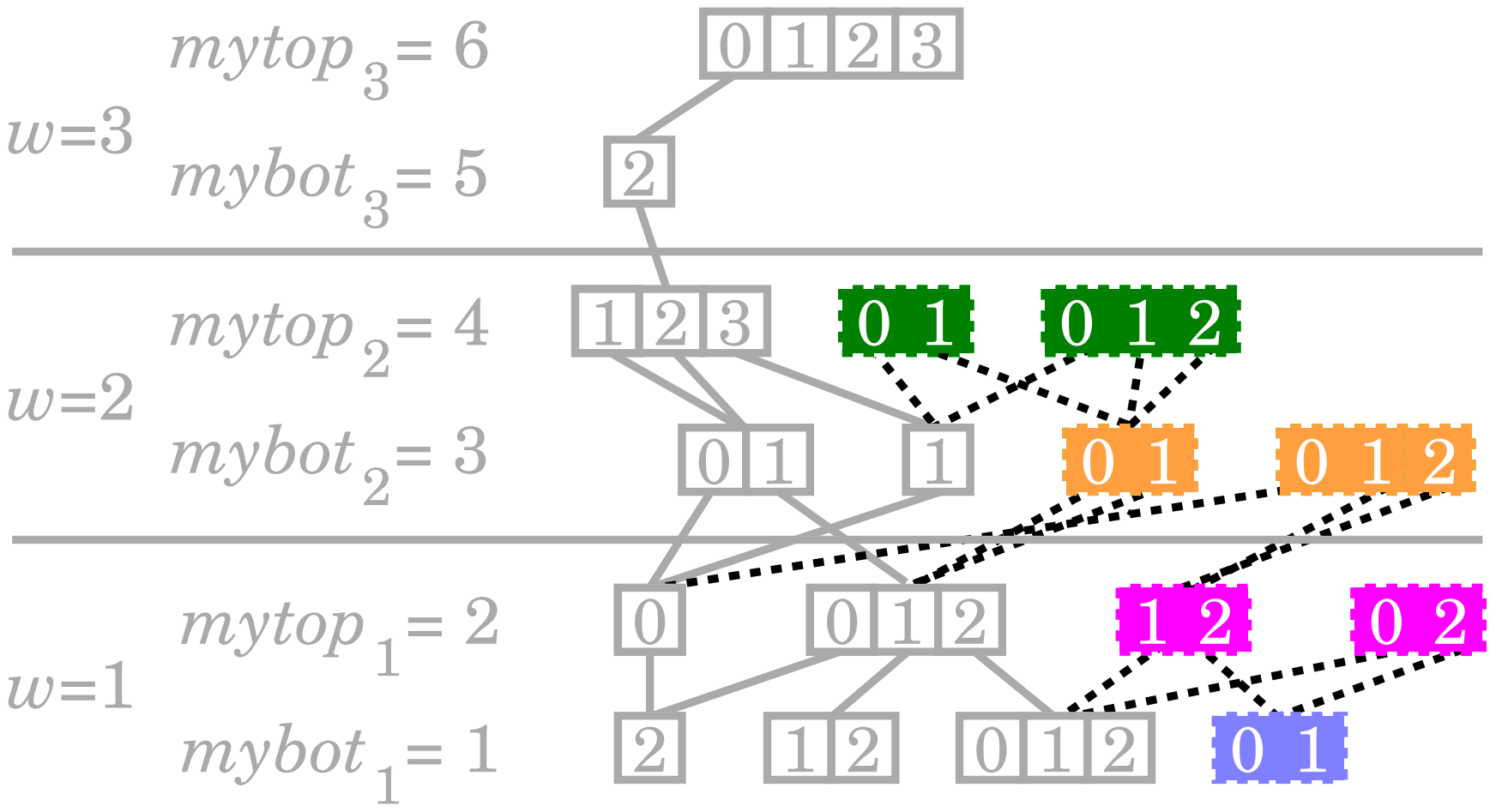
Invited submission to The Journal of Logic and Computation

Idea of Speculative Image Computation



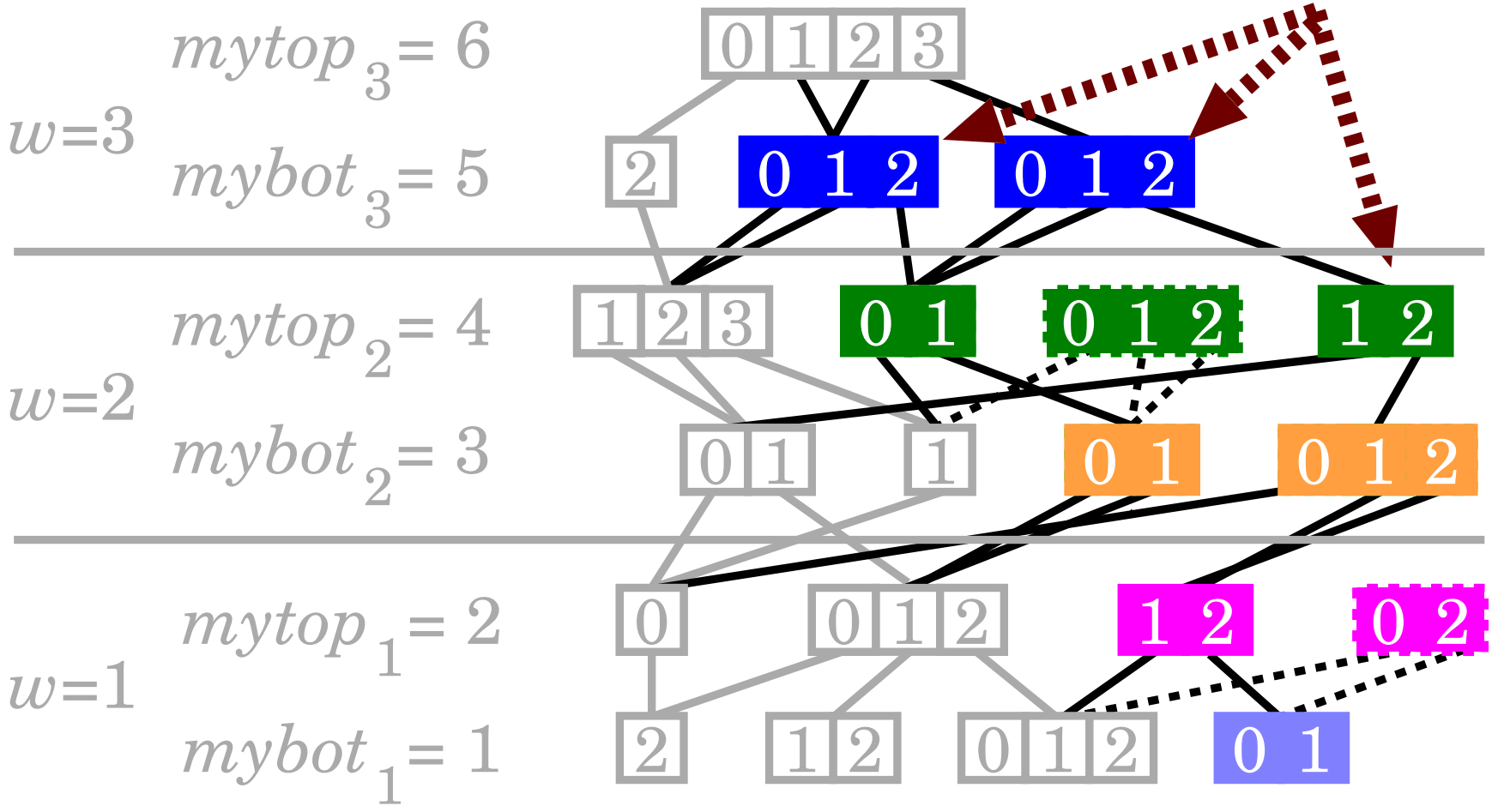
w_3, w_2, **and** w_1 **perform distributed state-space generation**.

Idea of Speculative Image Computation



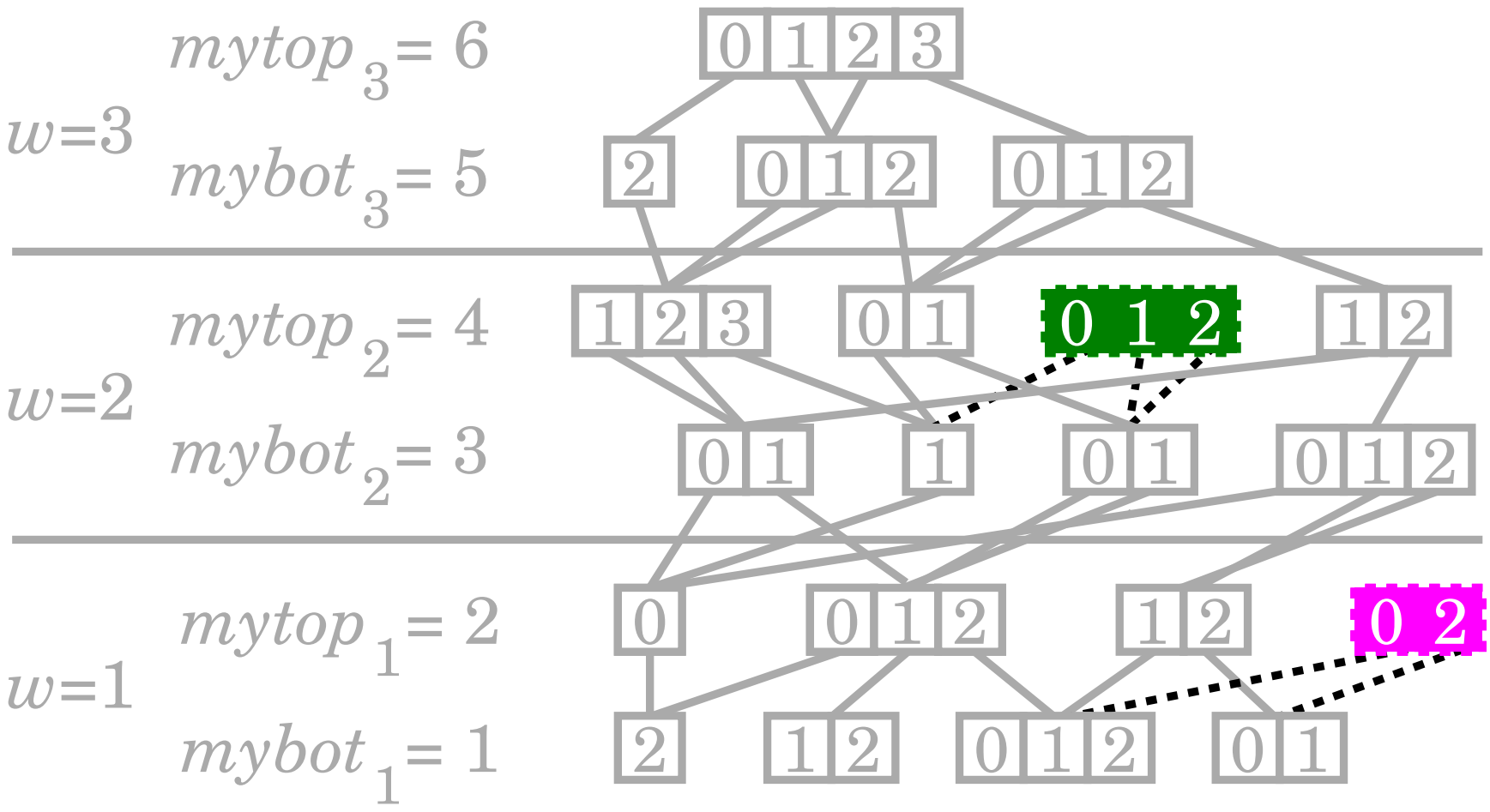
w_2 **and** w_1 **perform speculative computing together or individually**.

Idea of Speculative Image Computation



Some of the results speculatively created were retrieved from caches.

Idea of Speculative Image Computation



Many useless MDD nodes → high memory consumption.

Study on Speculative Image Computation

- **Problem** :
  - Do not know a priori whether an event will be fired on some MDD node (whether an image will be computed).
  - A NAÏVE approach: idle workstations exhaust all possible firings.
    - **Too many useless nodes and operation cache entries**.
    - **The prediction might not help at all**.

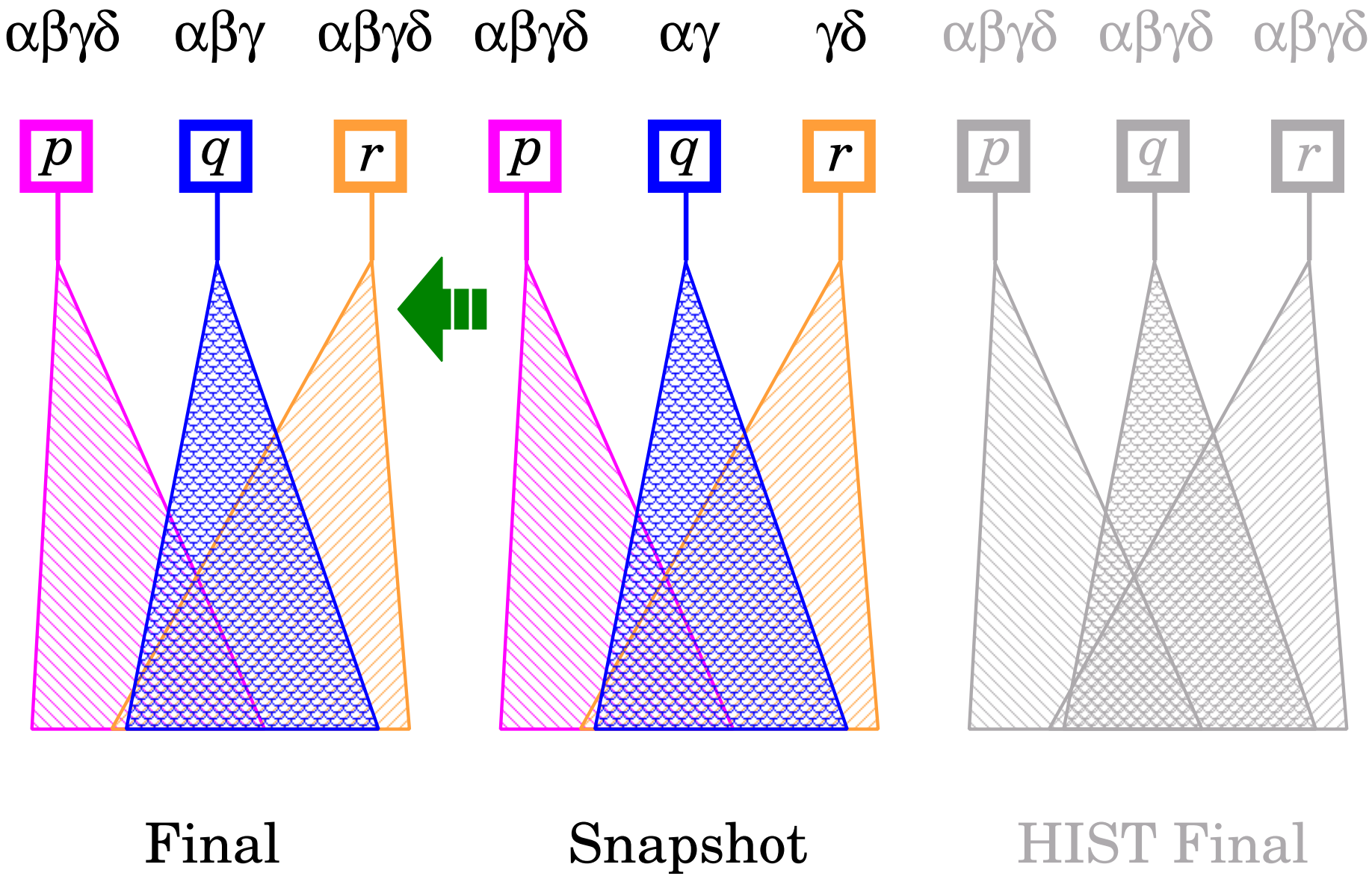
- **Observation** :

  Some MDD nodes have similar or even the same set of events fired on.

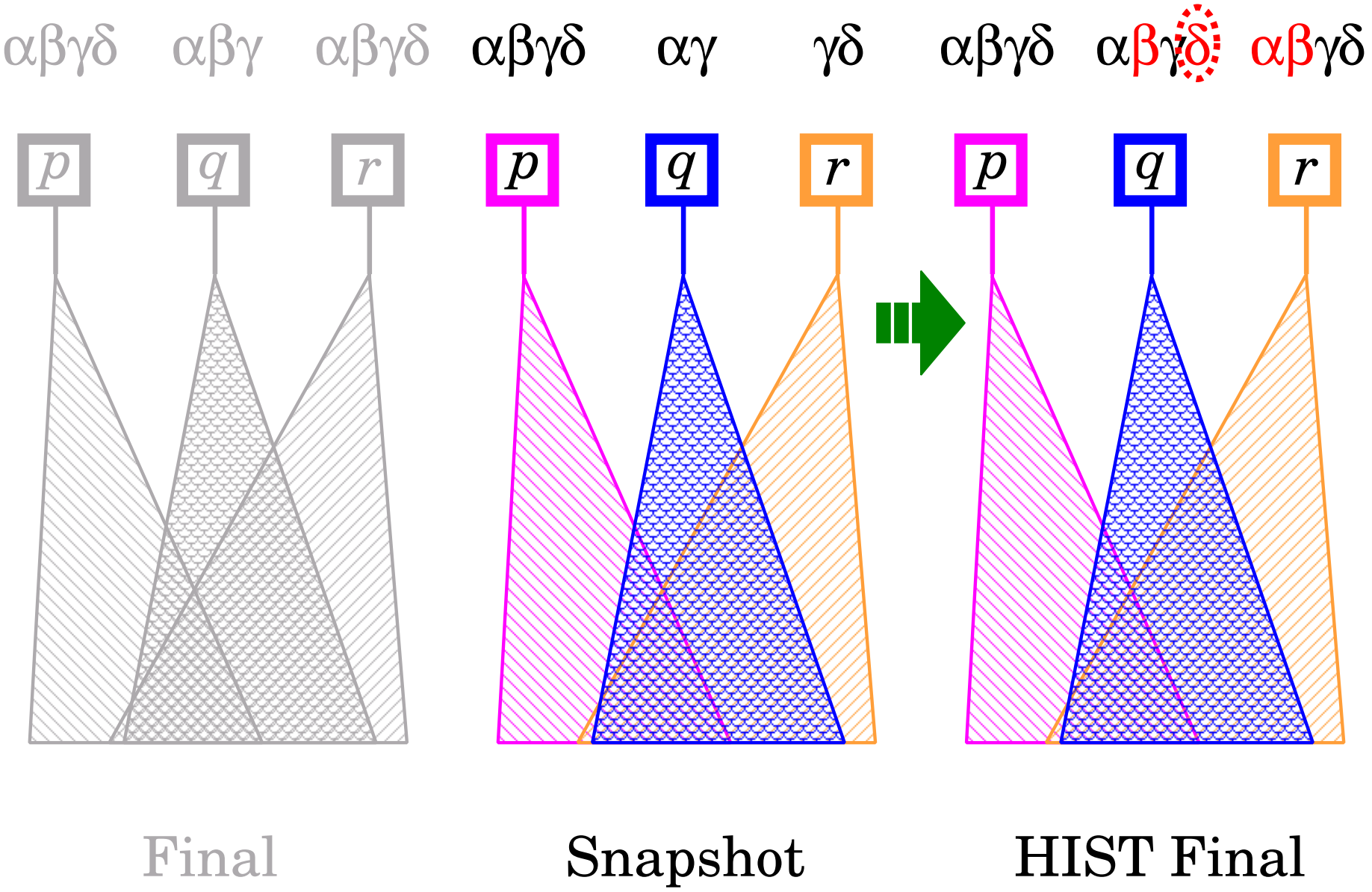
- **Solution** :

  A more informed firing prediction based on firing patterns.

History-based Image Speculation Scenario

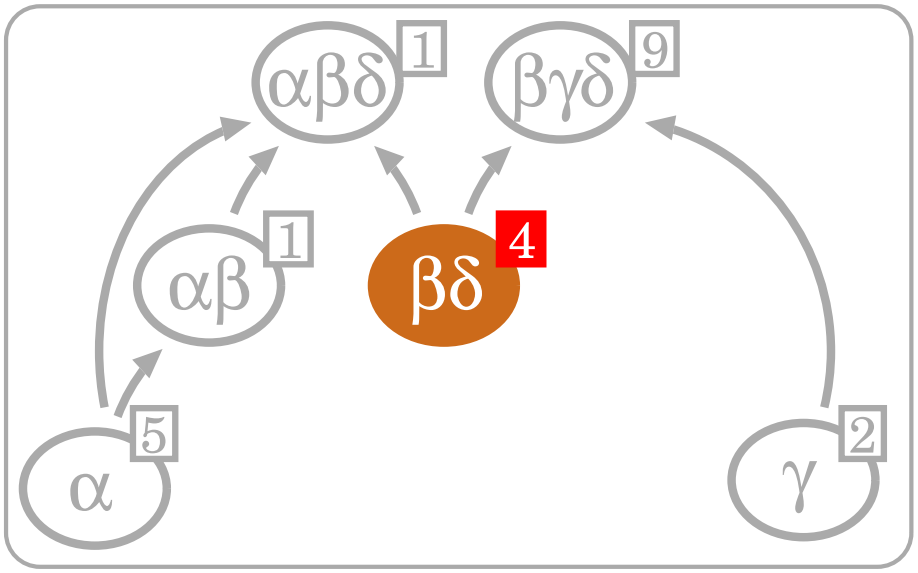


History-based Image Speculation Scenario



Firing Pattern Graph

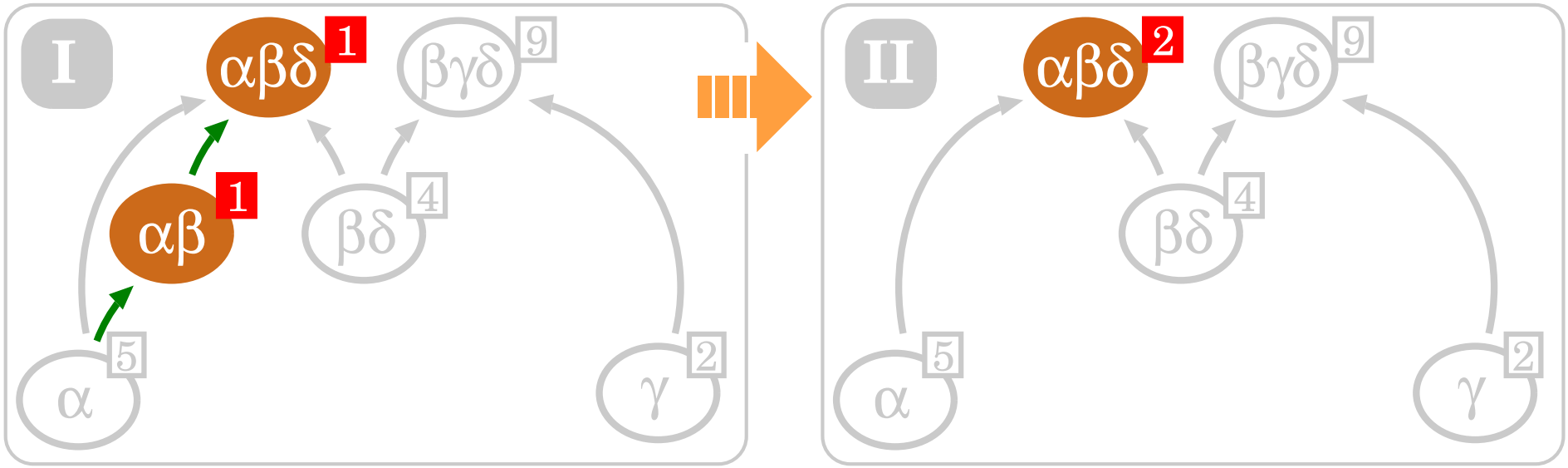
- For each MDD level, firing patterns are stored in a pattern graph: directed acyclic graph $G_k = (V_k, E_k)$ for $K \geq k \geq 1$.
- Each pattern graph node has an associated reference counter.



(snapshot) **Events β and δ have been fired on four MDD nodes at level k.**

Firing Pattern Graph

- For each MDD level, firing patterns are stored in a pattern graph: directed acyclic graph $G_k = (V_k, E_k)$ for $K \geq k \geq 1$.
- Each pattern graph node has an associated reference counter.

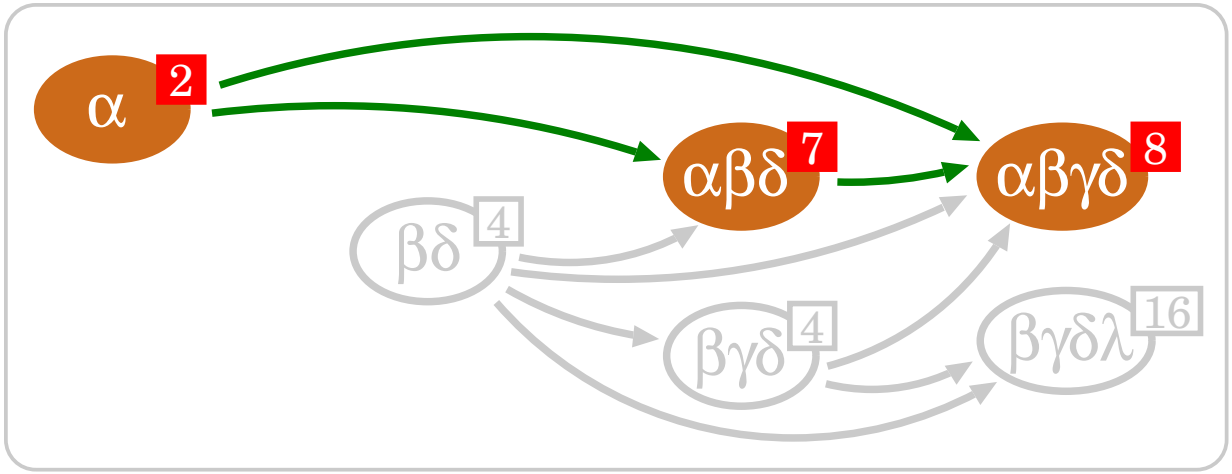


(update 1) **Firing event δ on an MDD node with current firing pattern $\{\alpha, \beta\}$.**

Graph-based Image Speculation

- There might be multiple ways to grow a pattern:

  multiple choices of image speculation.



- To maximize the usefulness of speculation and minimize the overhead, we
  - speculate only during idle time.
  - adjust the aggressiveness of speculation according to

$$SpecHitRate = \frac{\text{the number of speculative results used}}{\text{the number of speculative results computed.}}$$

Pattern Length Based Speculation

Each workstation initializes a variable $MaxDiff$ then

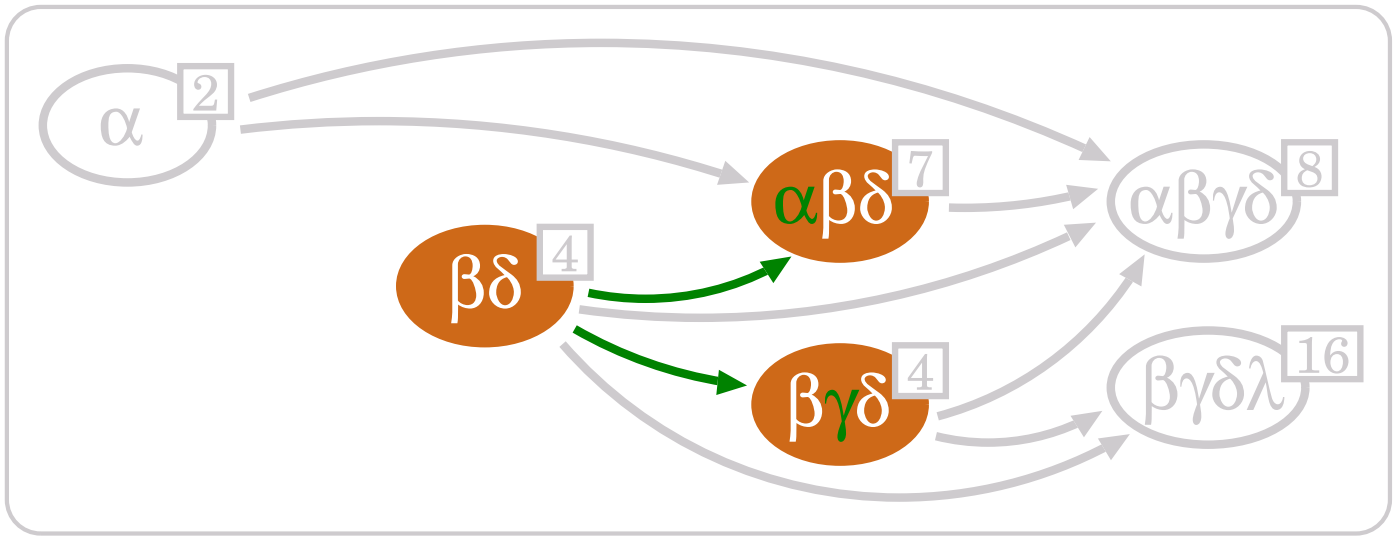
- **increases** $MaxDiff$ **whenever** $SpecHitRate$ **increases**.
- speculatively fires event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying $|\mathcal{E}_u| - |\mathcal{E}_v| \leq MaxDiff$.

Pattern Length Based Speculation

Each workstation initializes a variable $MaxDiff$ then

- **increases** $MaxDiff$ **whenever** $SpecHitRate$ **increases**.
- speculatively fires 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 $|\mathcal{E}_u| - |\mathcal{E}_v| \leq MaxDiff$.

(example) MDD node p referencing pattern graph node encoding pattern $\{\beta, \delta\}$.



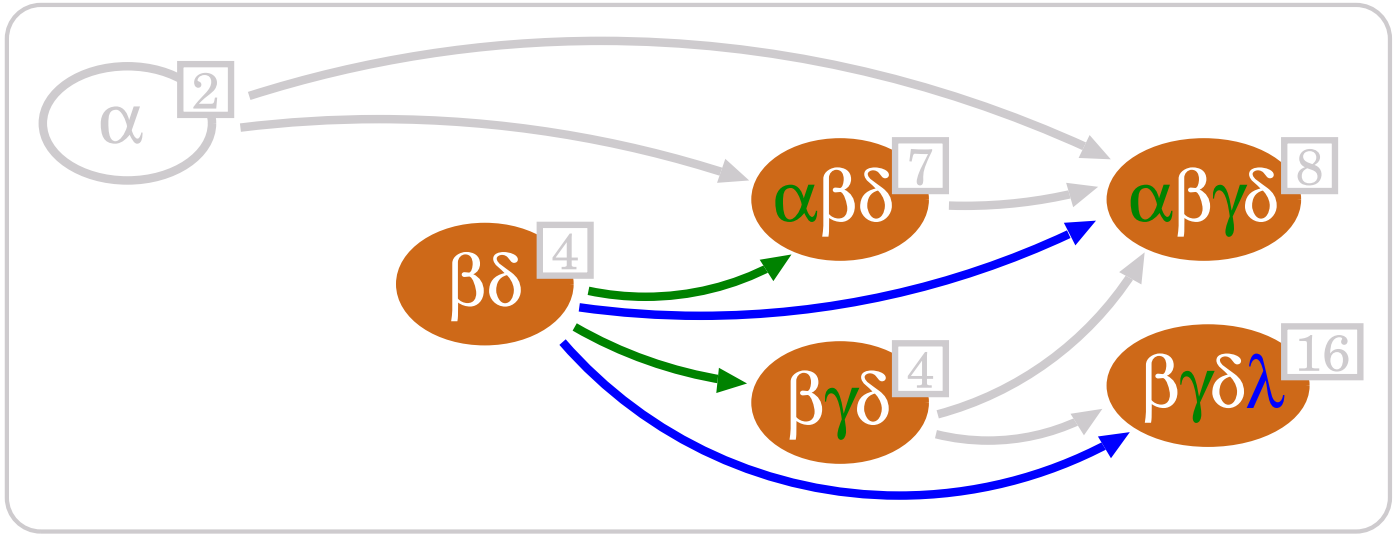
If $MaxDiff = 1$, **we speculatively fire events** α **and** γ **on MDD node** p.

Pattern Length Based Speculation

Each workstation initializes a variable $MaxDiff$ then

- **increases** $MaxDiff$ **whenever** $SpecHitRate$ **increases**.
- speculatively fires 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 $|\mathcal{E}_u| - |\mathcal{E}_v| \leq MaxDiff$.

(example) MDD node p referencing pattern graph node encoding pattern $\{\beta, \delta\}$.

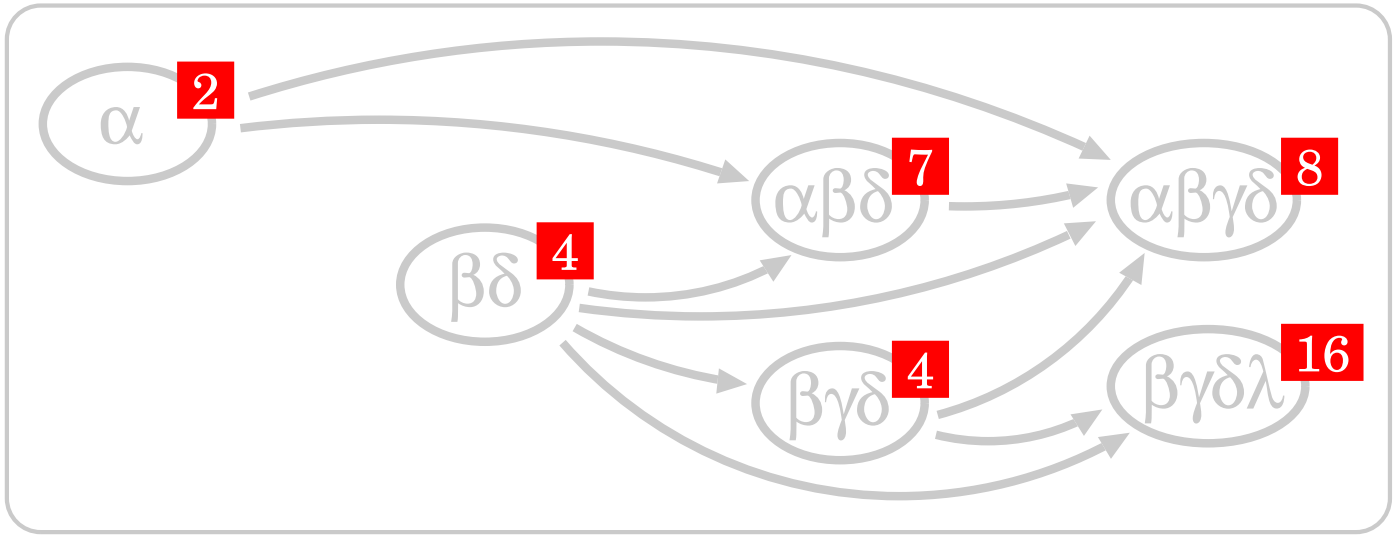


If $MaxDiff = 2$, **we speculatively fire events** α, γ, **and** λ **on MDD node** p.

Pattern Length Based Speculation

Each workstation initializes a variable $MaxDiff$ then

- **increases** $MaxDiff$ **whenever** $SpecHitRate$ **increases**.
- speculatively fires 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 $MinScore$ then

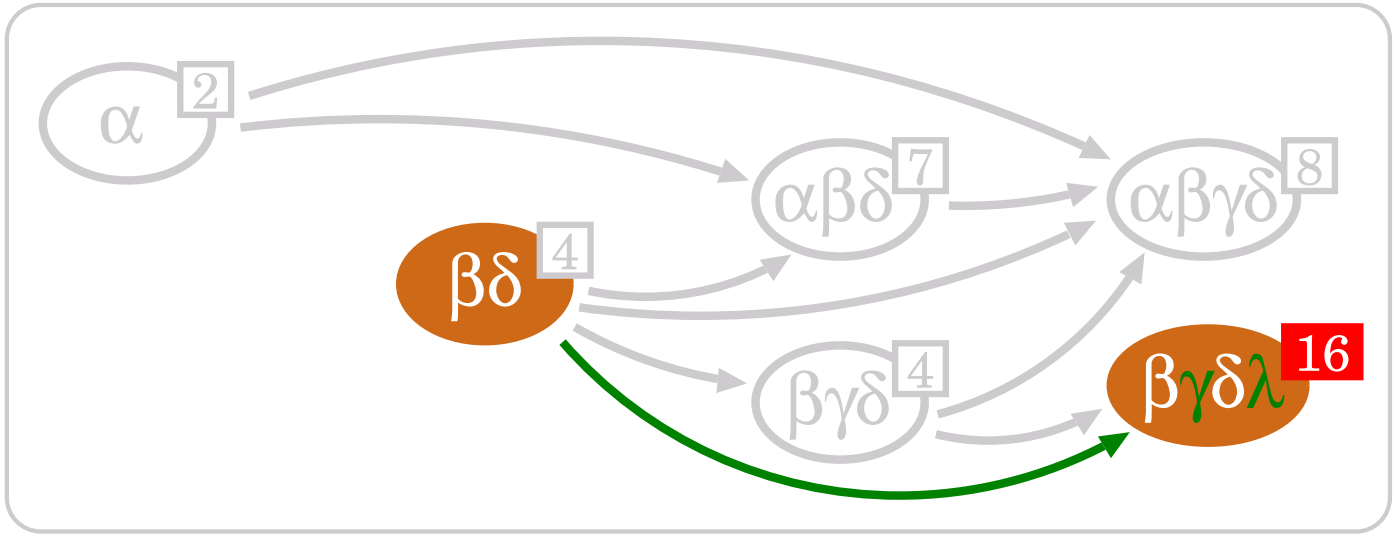
- **decreases** $MinScore$ **whenever** $SpecHitRate$ **increases**.
- speculatively fires event $e \in \{\mathcal{E}_u \setminus \mathcal{E}_v\}$ on MDD node p referencing pattern graph node v for any pattern graph node $u \in v.parent$ satisfying $u.ref/(|\mathcal{E}_u| - |\mathcal{E}_v|) \geq MinScore$.

Weighted Score Based Speculation

Each workstation initializes a variable $MinScore$ then

- **decreases** $MinScore$ **whenever** $SpecHitRate$ **increases**.
- speculatively fires 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 $\{\beta, \delta\}$.



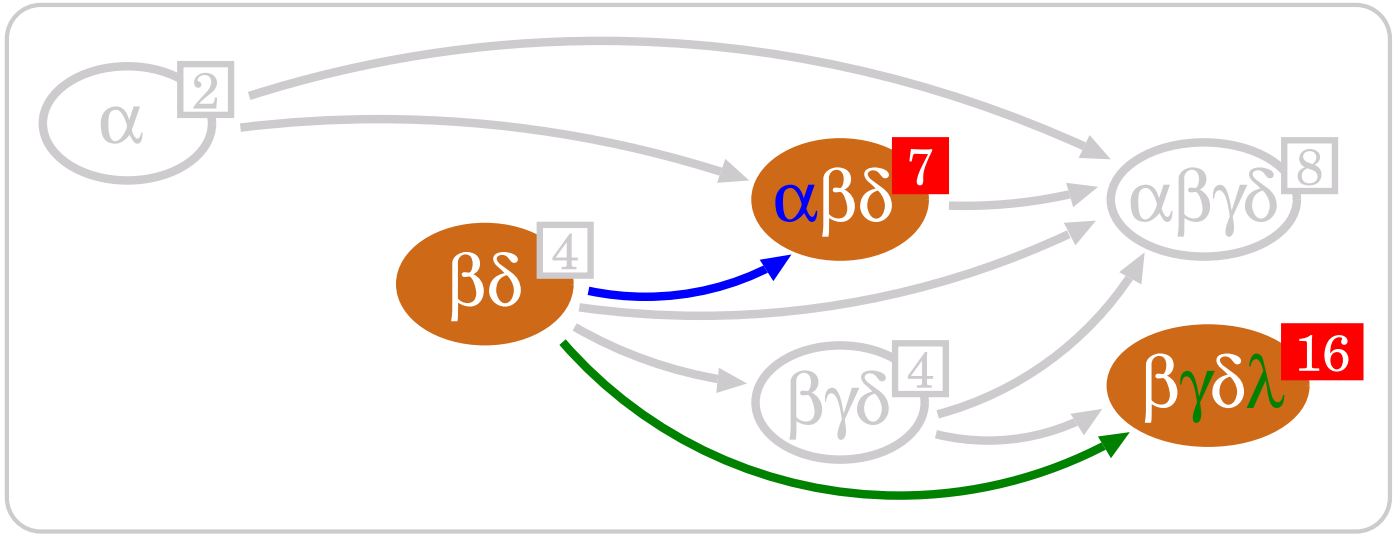
If $MinScore = 8$, **we speculatively fire events** γ **and** λ **on MDD node** p.

Weighted Score Based Speculation

Each workstation initializes a variable $MinScore$ then

- **decreases $MinScore$ whenever $SpecHitRate$ increases.**
- speculatively fires 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 $\{\beta, \delta\}$.



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

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.

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.

Fine-Grained Fullness-Guided Chaining

Chaper 7

Ming-Ying Chung, Gianfranco Ciardo, and Andy Jinqing Yu.

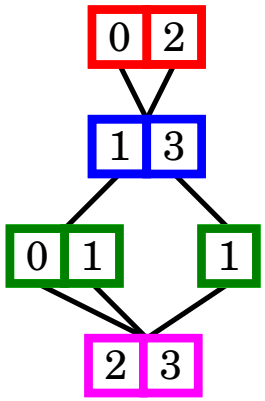
A fine-grained fullness-guided chaining heuristic for symbolic reachability analysis. ATVA 2006.

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

$\mathcal{S}_3 = \{0, 1, 2, 3\}$

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

$\mathcal{S}_1 = \{0, 1, 2, 3\}$



$$\frac{12}{3 \times 4 \times 2 \times 4} = 12.5\%$$

$$\mathcal{X} = \left\{ \begin{array}{cccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 3 & 3 & 1 & 1 & 1 & 1 & 3 & 3 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 & 2 & 3 \end{array} \right\}$$

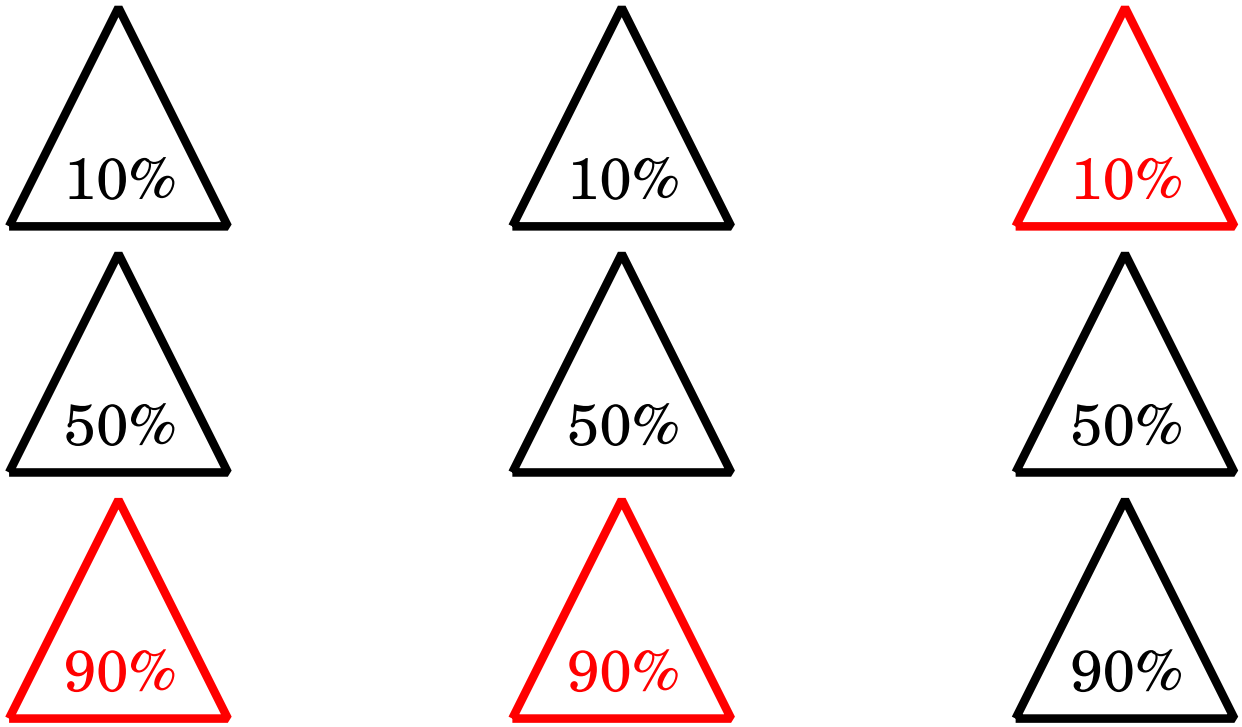
Fine-Grained Fullness-Guided Chaining

Chaper 7

Ming-Ying Chung, Gianfranco Ciardo, and Andy Jinqing Yu.

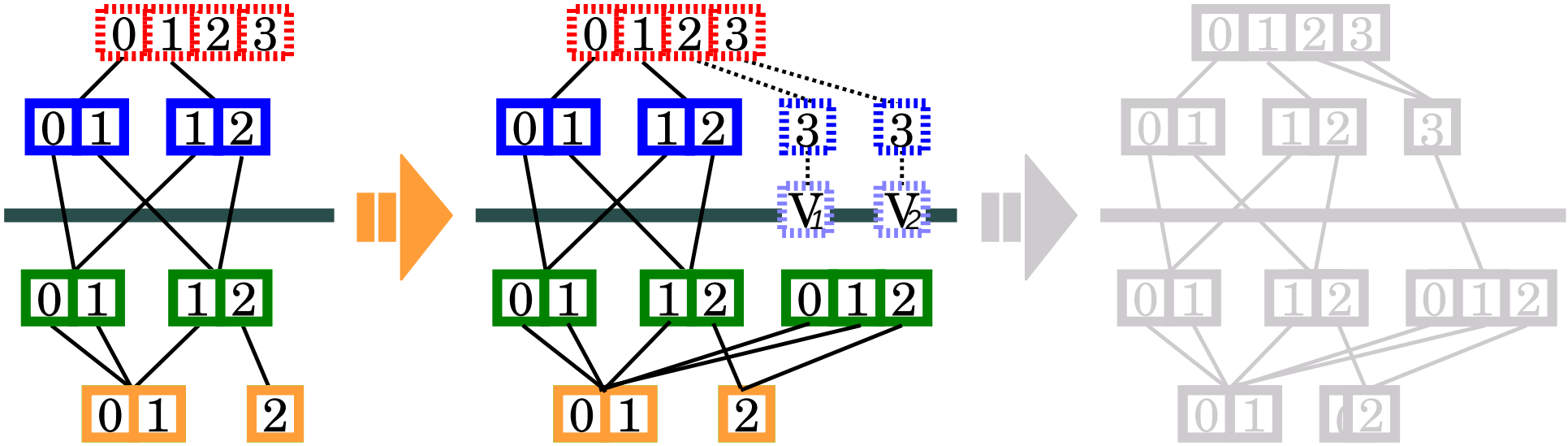
A fine-grained fullness-guided chaining heuristic for symbolic reachability analysis. ATVA 2006.

$$\sigma(p,i,j) = \phi(p[i]) \cdot \phi(r_k^\star[i][j]) \cdot (1-\phi(p[j]))$$



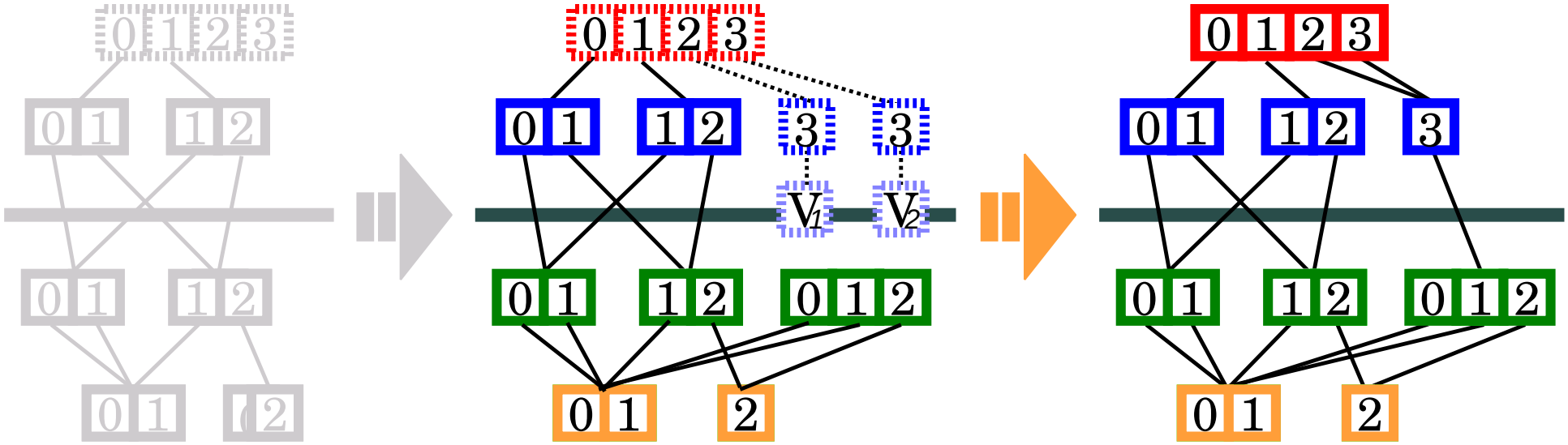
Future Research

True Parallelism - Virtual MDD Nodes



- **MDD nodes are still canonical**.
- Controlling the use of virtual indices.
- Respecting the chaining order.

True Parallelism - Virtual MDD Nodes



- MDD nodes are still canonical.
- **Controlling the use of virtual indices**.
- **Respecting the chaining order**.

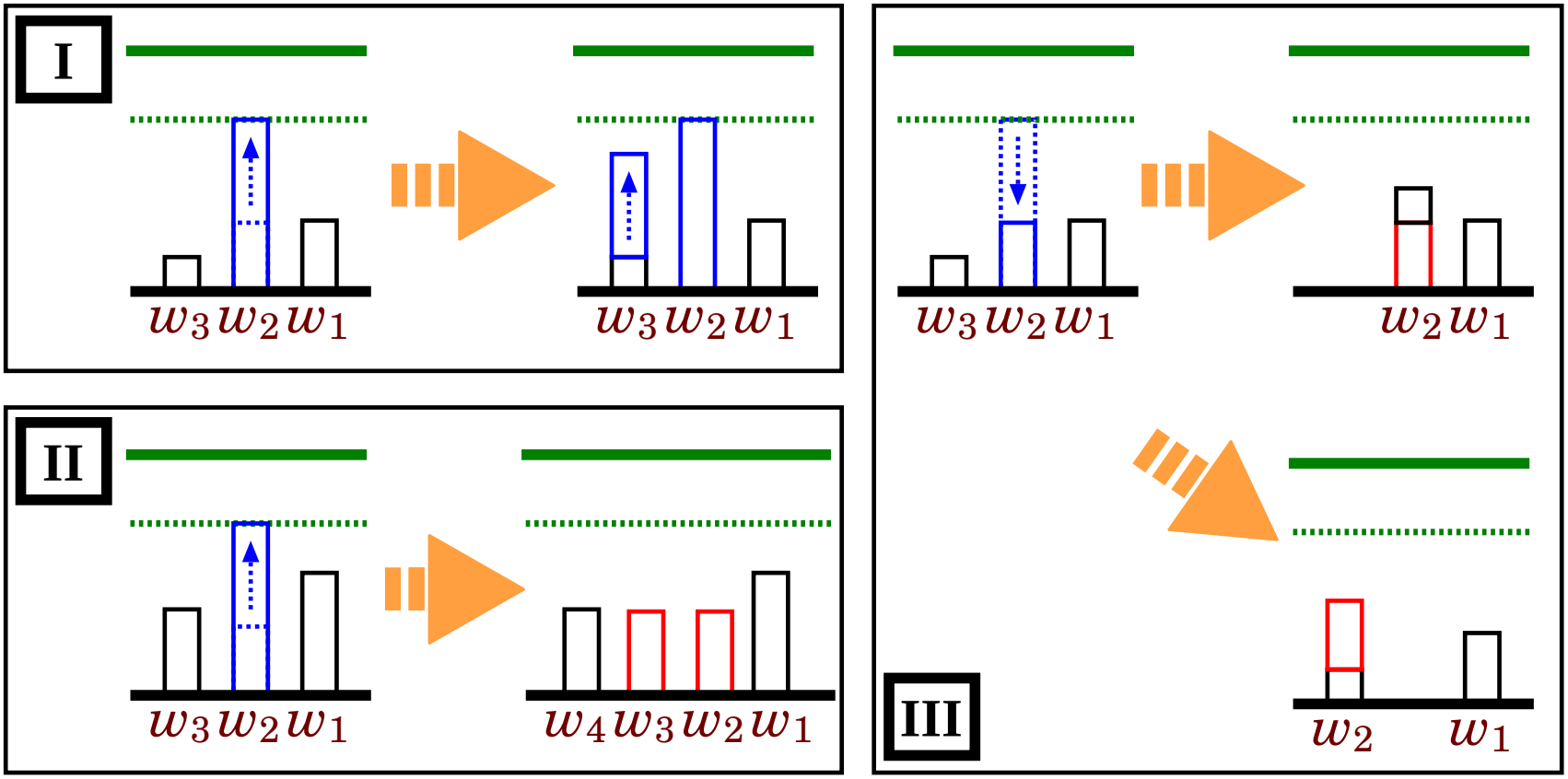
Speculation-Based Parallelism

- Single thread performs reachability analysis.
- The rest of the threads perform speculative image computation.

N	Runtime (sec)					Memory (mb)				
	SEQ	1T	2T	3T	4T	SEQ	1T	2T	3T	4T
Slot										
50	5.93	9.01	5,88	4.27	3.36	32	66	67	67	67
100	44.75	67.95	41.8	28.73	22.98	87	153	172	178	170
FMS										
100	6.72	9.48	7.55	6.54	6.09	20	34	37	37	37
150	22.06	30.6	20.48	18.97	16.87	44	56	69	71	71
RIPS										
12333	16.33	25.19	28.12	25.02	24.92	20	18	33	36	39
13433	50.32	75.99	80.71	76.03	75.55	49	55	69	70	78
Robin										
250	6.44	10.22	10.9	11.48	12	167	337	351	356	361
500	28.2	44.43	46.38	49.26	51.83	379	730	785	801	816

Other Future Research

- Distributed computation using a variable number of workstations.



- Speculation-based application.
- Parallel and thread-safe decision diagram library.

Thank You!