Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan

Throughput Optimization for Latency-Insensitive System with Minimal Queue Insertion

Juinn-Dar Huang, Yi-Hang Chen, Ya-Chien Ho

Advanced
Design
Automation
Research

Yi-Hang Chen
carlok@adar.ee.nctu.edu.tw

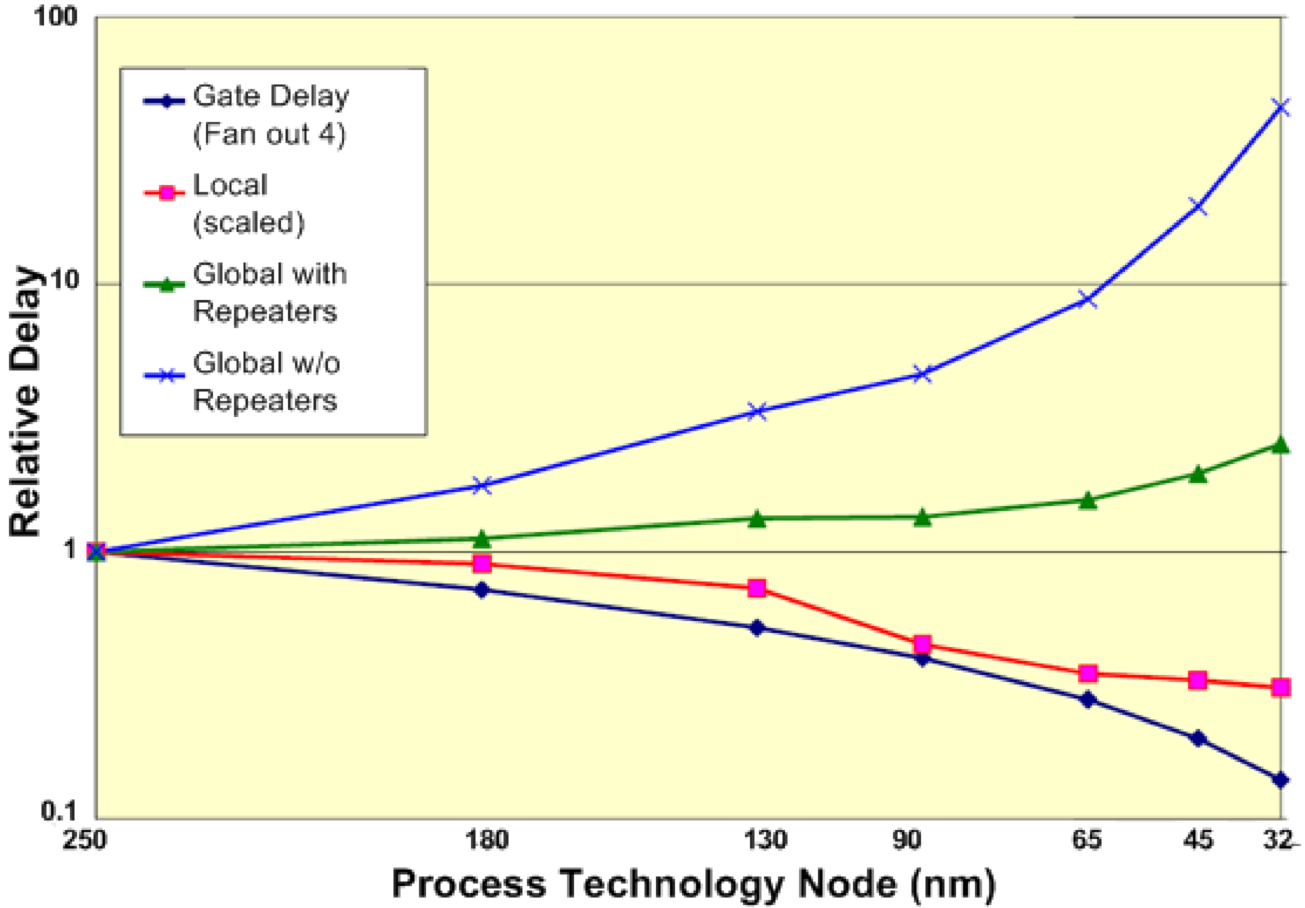
Jan. 28, 2011

Outline

- Introduction
- Preliminaries
  - Latency-Insensitive System (LIS)
  - Marked Graph (MG)
- Proposed Queue Sizing Method
  - Quantitative Graph (QG) and Compacted QG (CQG)
  - Compaction Phase
  - ILP Formulation
  - Recovery Phase
- Experimental Results
- Conclusions

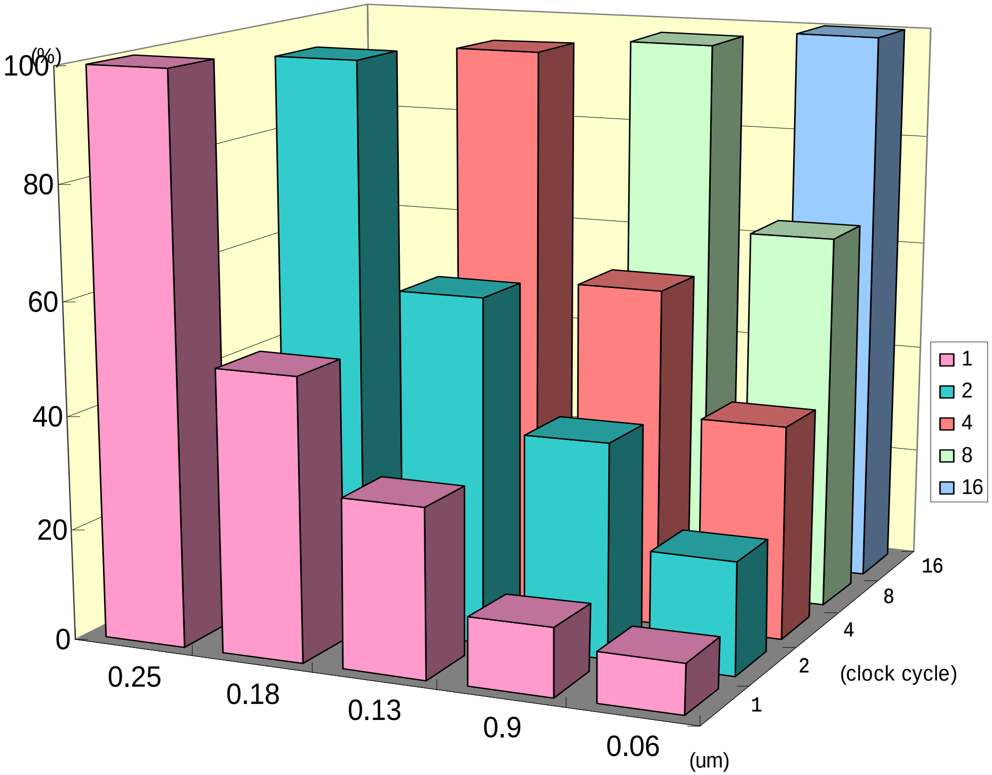
Introduction (1/3)

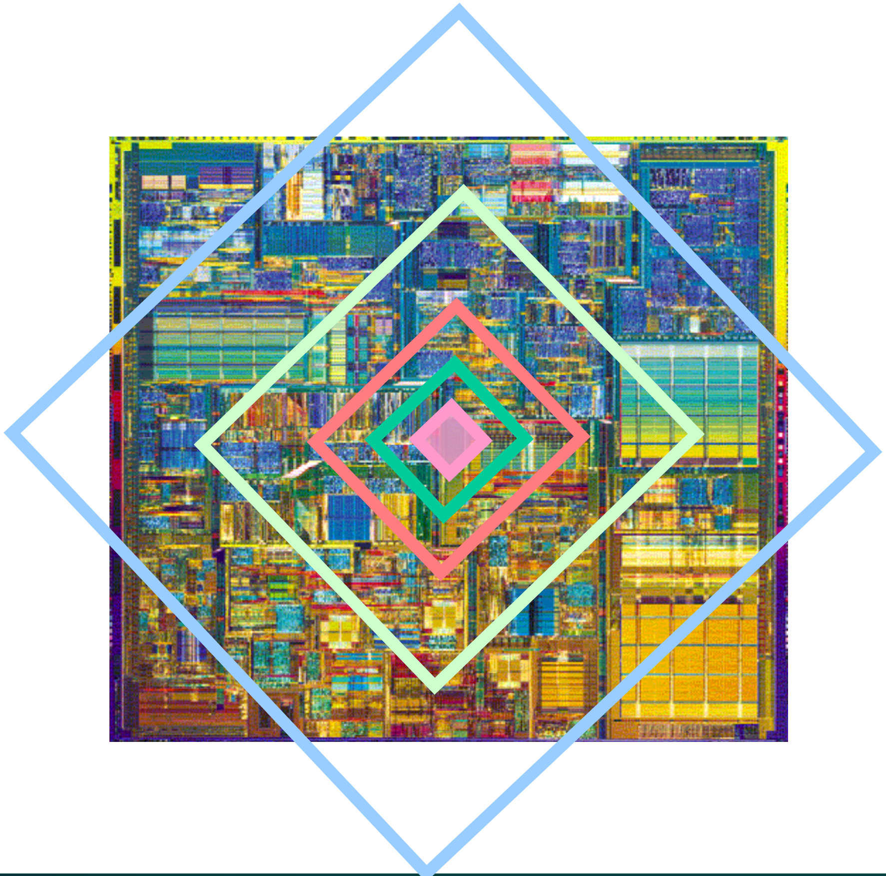
- **As the manufacturing process keeps scaling down…**
  - Global interconnect delay becomes the largest fraction of a clock cycle time



Introduction (2/3)

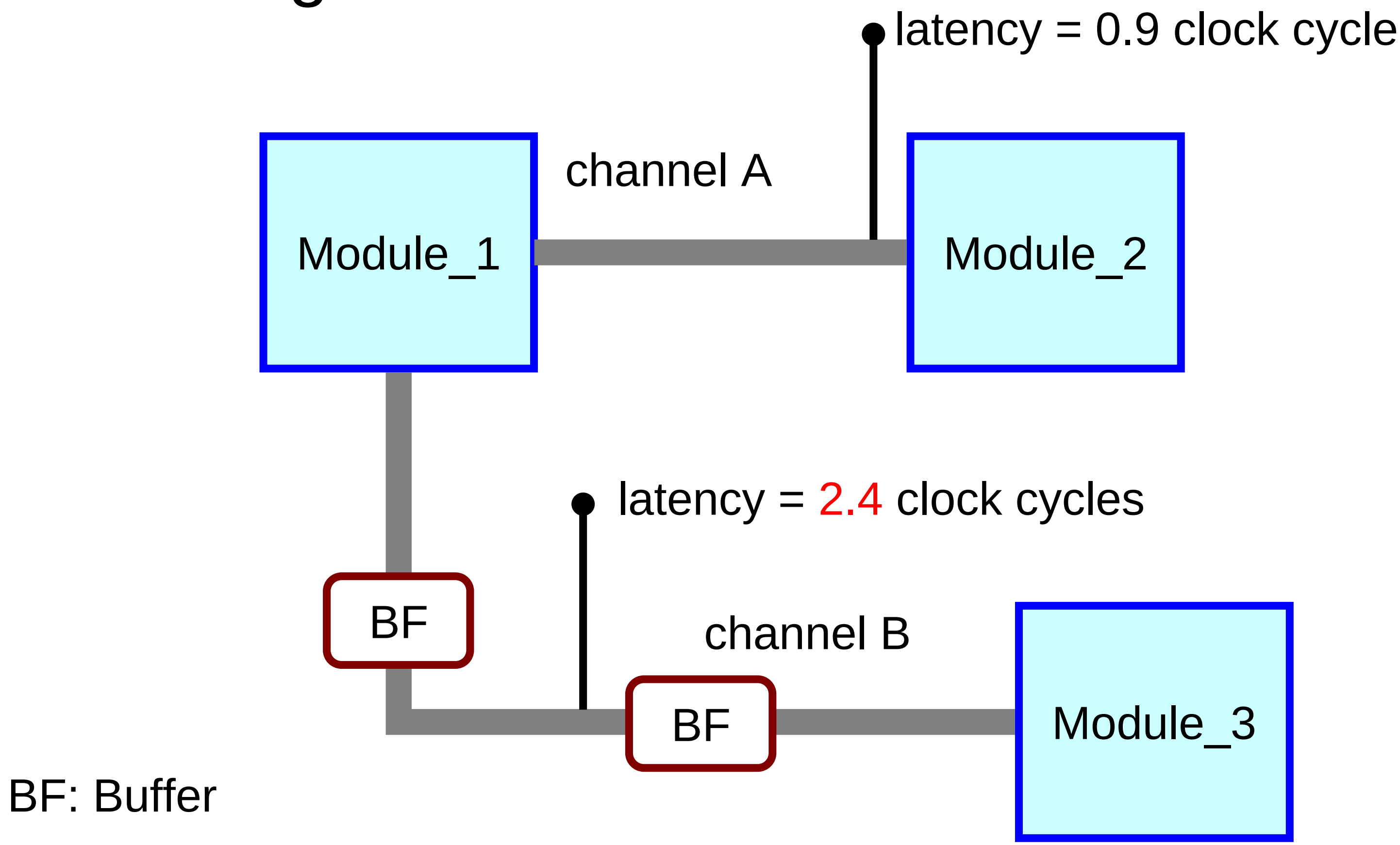
- DSM dilemma:
  - For a 0.06 micron process, a signal can reach only 10% of the die's length in a clock cycle
  - Design paradigm shifts from "computation-" to "communication-" bound design





Introduction (3/3)

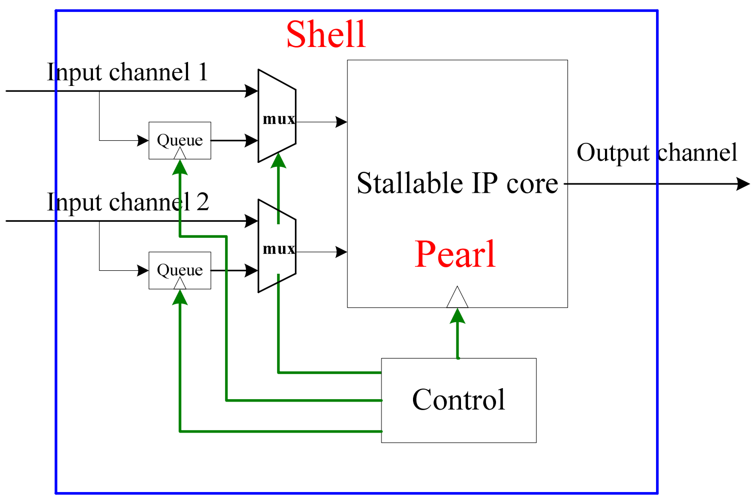
- Relax timing constraint



  – Performance may be degraded due to multi-cycle communication

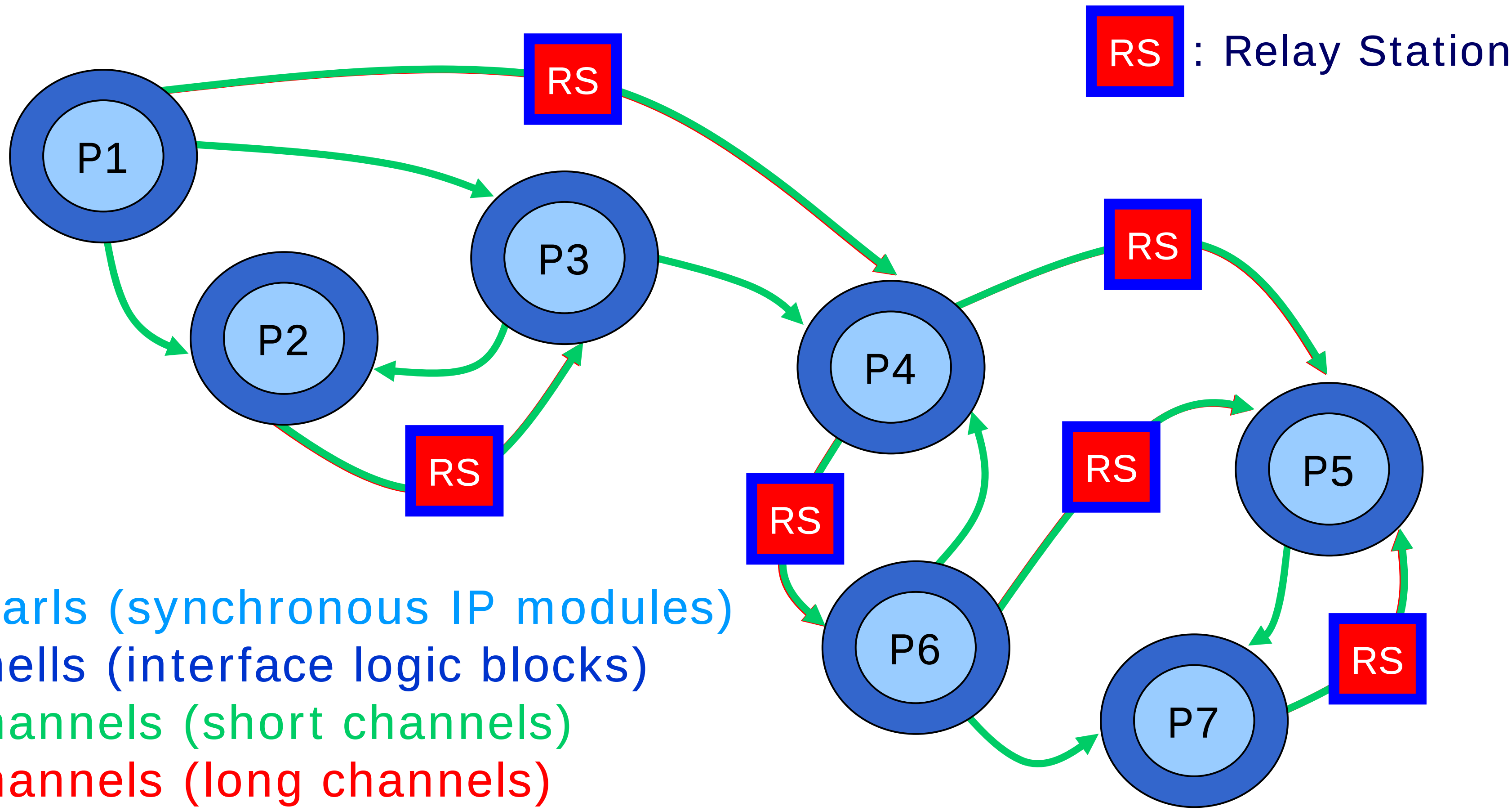
LIS (1/3)

- Latency-insensitive system (LIS) is a design methodology to deal with arbitrary variation in channel latency
- Interface logic blocks (shells) encapsulate the pre-designed IP modules (pearls)
- Insert relay stations (buffers) to pipeline long interconnects



LIS (2/3)

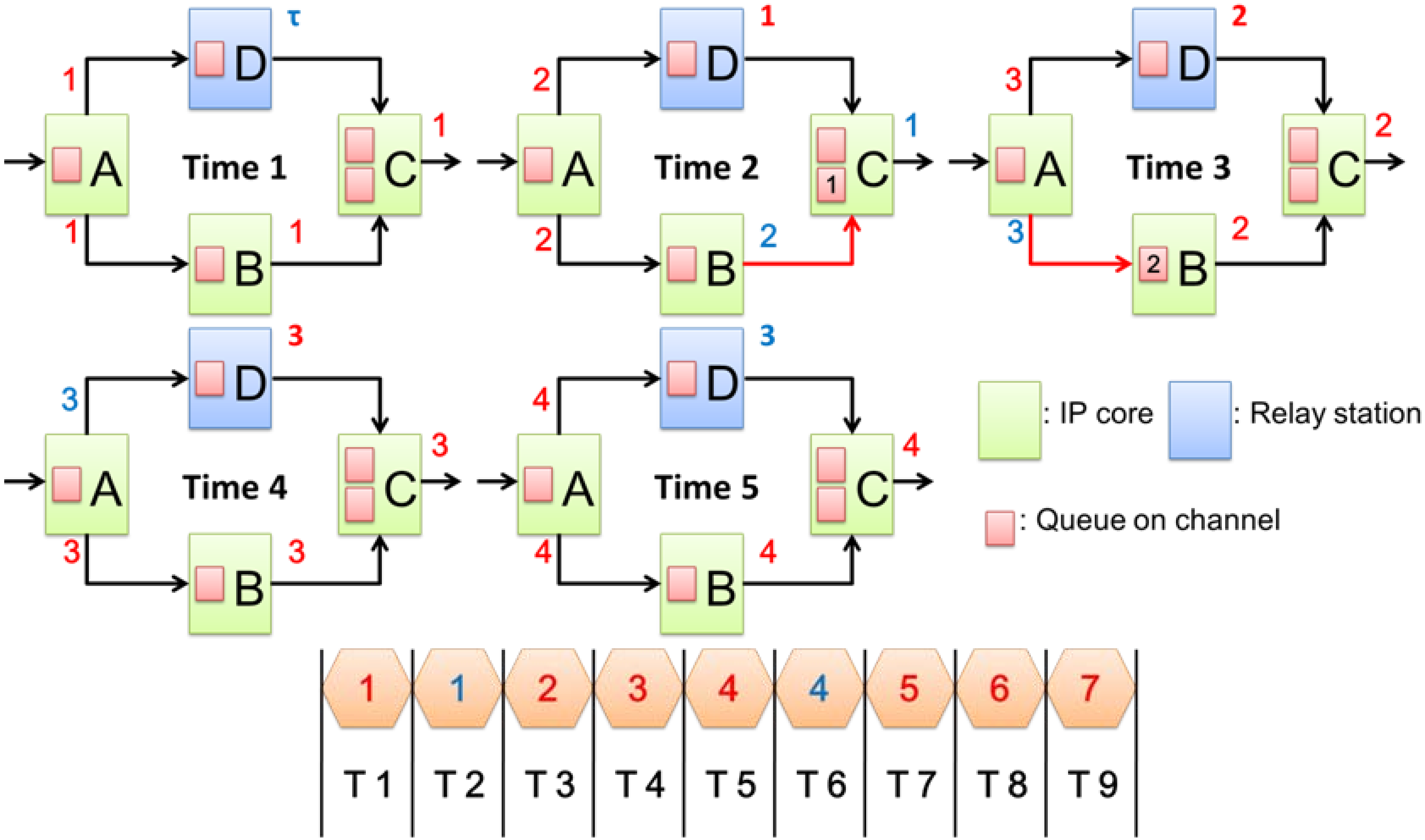
An LIS Example



Pearls (synchronous IP modules)
Shells (interface logic blocks)
Channels (short channels)
Channels (long channels)

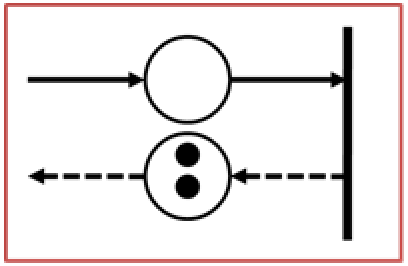
LIS (3/3)

- Functional behavior is identical as the original system
  - however, latency and throughput may **NOT** be

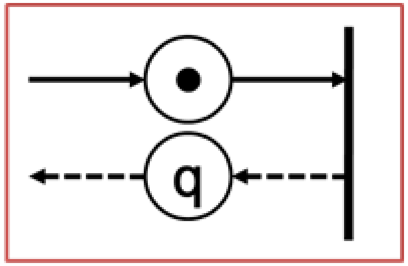


Marked Graph (1/2)

- Marked graph (MG) is a conventional representation for modeling concurrent operations within a system
  - For RS
    - solid edge : no token
      - RS produces a void event initially
    - dashed edge : two tokens
      - every RS contains a two-entry queue
  - For shell
    - solid edge : one token
      - shell produces a valid event initially
    - dashed edge : actual queue size



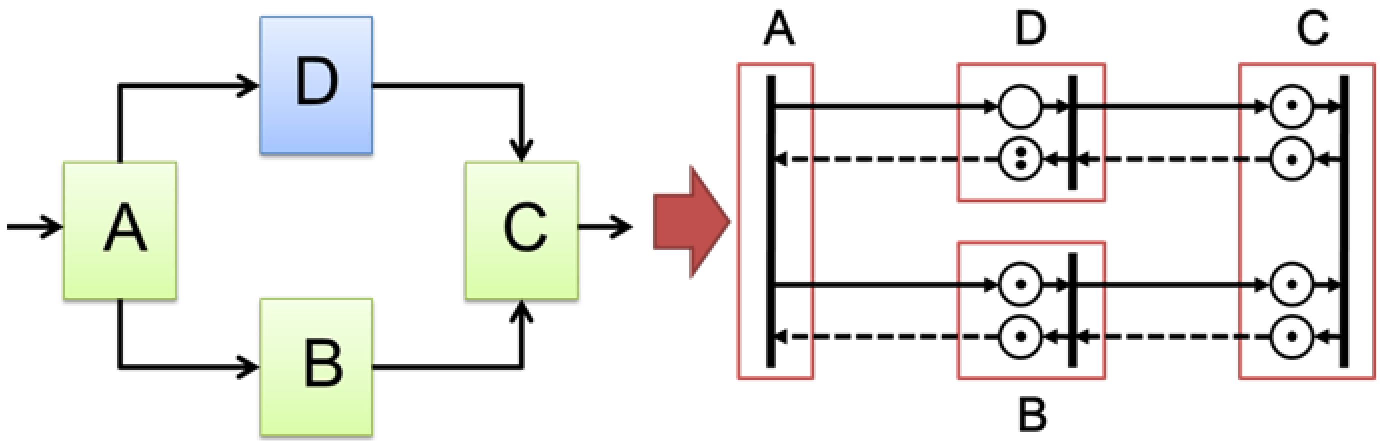
Relay station



Shell

Marked Graph (2/2)

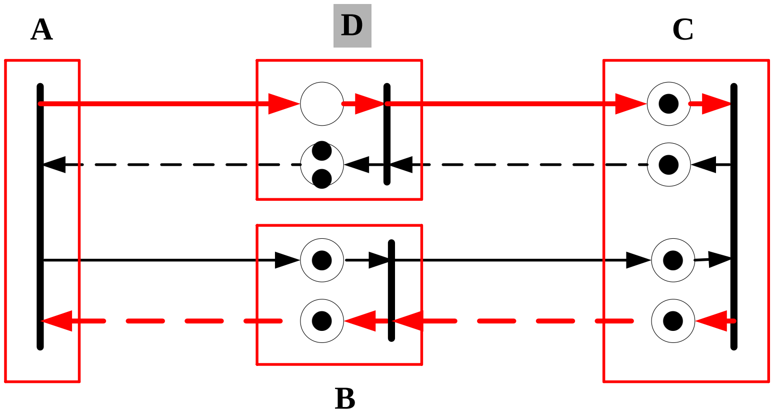
- Transform an LIS into its MG representation
  - the queue size of all channels in shells is set to one



- The maximal sustainable throughput (MST) of an LIS is bound to the lowest token-to-place ratio (TPR) of all cycles in its corresponding MG

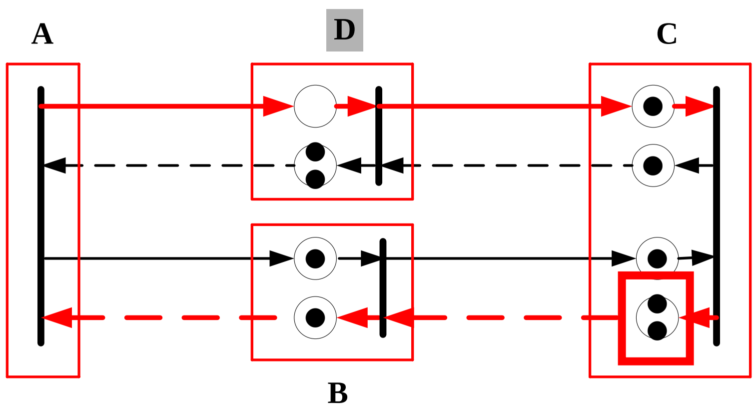
Marked Graph Example

- System throughput : find the cycle with the lowest ratio of tokens to places



Critical cycle : A-D-C-B-A

System throughput : 3/4



Add one queue at channel B-C

System throughput : 1 (4/4)

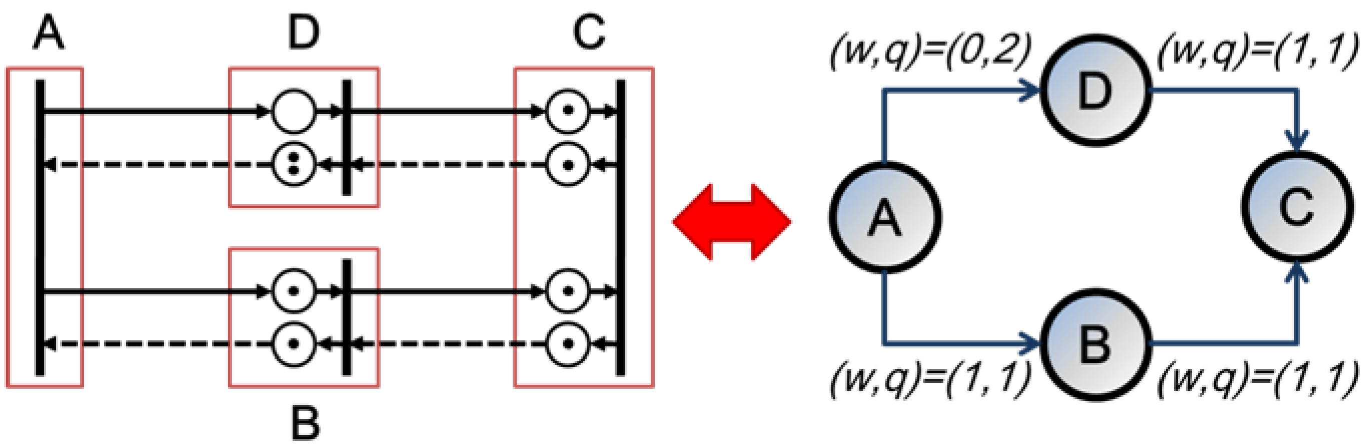
Optimize throughput by queue sizing in marked graph

Outline

- Introduction
- Preliminaries
  - Latency-Insensitive System (LIS)
  - Marked Graph (MG)
- Proposed Queue Sizing Method
  - Quantitative Graph (QG) and Compacted QG (CQG)
  - Compaction Phase
  - ILP Formulation
  - Recovery Phase
- Experimental Results
- Conclusions

Quantitative Graph (QG)

- A *quantitative graph* (QG) with respect to a given MG is a quadruple (V, E, w, q)
  - V is the set of vertices
  - E is the set of edges
  - w : E→ Z^+ specifies the number of valid tokens
  - q : E→ Z^+ indicates the queue size



→ : regular direction (F)

- - -> : reverse direction (R)

Compacted Quantitative Graph (CQG)

- A *compacted quantitative graph* (CQG) H is defined as a sextuple (V, E, w, q, b, c)
  - (V, E, w, q) is identical to that of QG
  - $c: E \rightarrow Z^{+}$ assigns an extra *compaction factor* to record the compaction level
  - $b: E \rightarrow Z^{+}$ specifies an extra *burden factor* to register the load level due to compaction

TPR : token-to-place ratio

$$TPR(C) = \frac{\sum_{e \in F} w(e) + \sum_{e \in R} q(e)}{\sum_{e \in C} c(e)}$$

TPD : token-place difference

$$TPD_C(e) = \begin{cases} q(e) - c(e), \text{ for } e \in R \text{ in cycle } C \\ w(e) - c(e), \text{ for } e \in F \text{ in cycle } C \end{cases}$$

F (regular direction) : ⟶

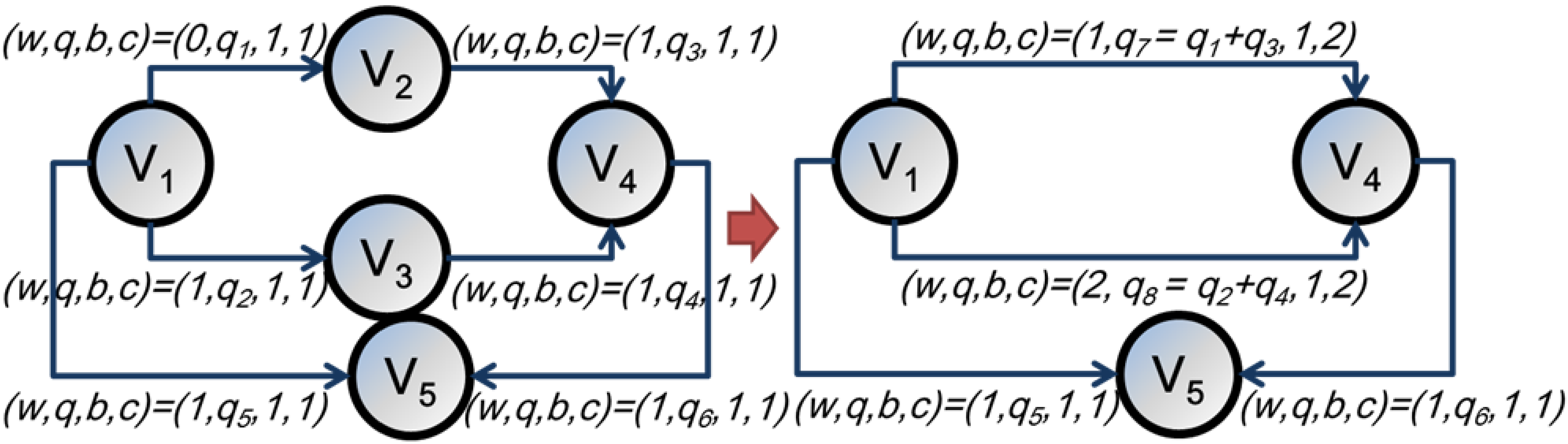
R (reverse direction) : - - - ->

Compaction Phase

- The size of QG becomes extremely large as the corresponding system gets complicated
- We propose a compaction phase to further decrease graph size
  - Path Condensation
  - Edge Unification

Compaction Phase - Path Condensation

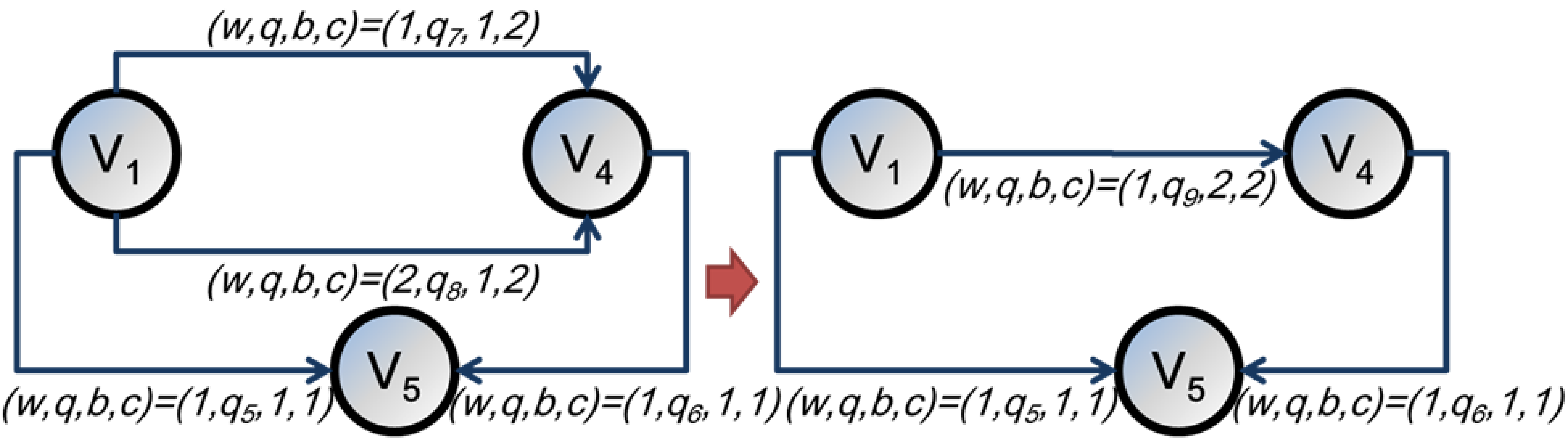
- We call a simple path $p_{u,v}$<u,v_1,…v_n,v> condensable if it satisfies the following two conditions
  - The length of $p_{u,v} \geq 3$, or $n \geq 1$
  - For each intermediate vertex $\{v_1, v_2, \ldots, v_n\}$, input degree and output degree must both be equal to **1**



Compaction Phase - Edge Unification

- For any two vertices v_i and v_j, if there exist multiple edges from v_i to v_j
  - $E_m(v_i, v_j)$ is the set containing all parallel edges from v_i to v_j
  - An edge $e_d \in E_m(v_i, v_j)$ is called a *dominating edge*, if
    - $c(e_d) - w(e_d) \geq c(e_k) - w(e_k)$ for every edge $e_k \in E_m(v_i, v_j)$

Lowest token-place difference ➔ most critical constraint



ILP Formulation

- After a series of path condensation and edge unification operations, a CQG *H* with minimal vertices and edges can be derived
- On top of CQG, using ILP to get optimal solutions

$$\textbf{Minimize: } \sum_{e \in E} q(e)$$

subject to:

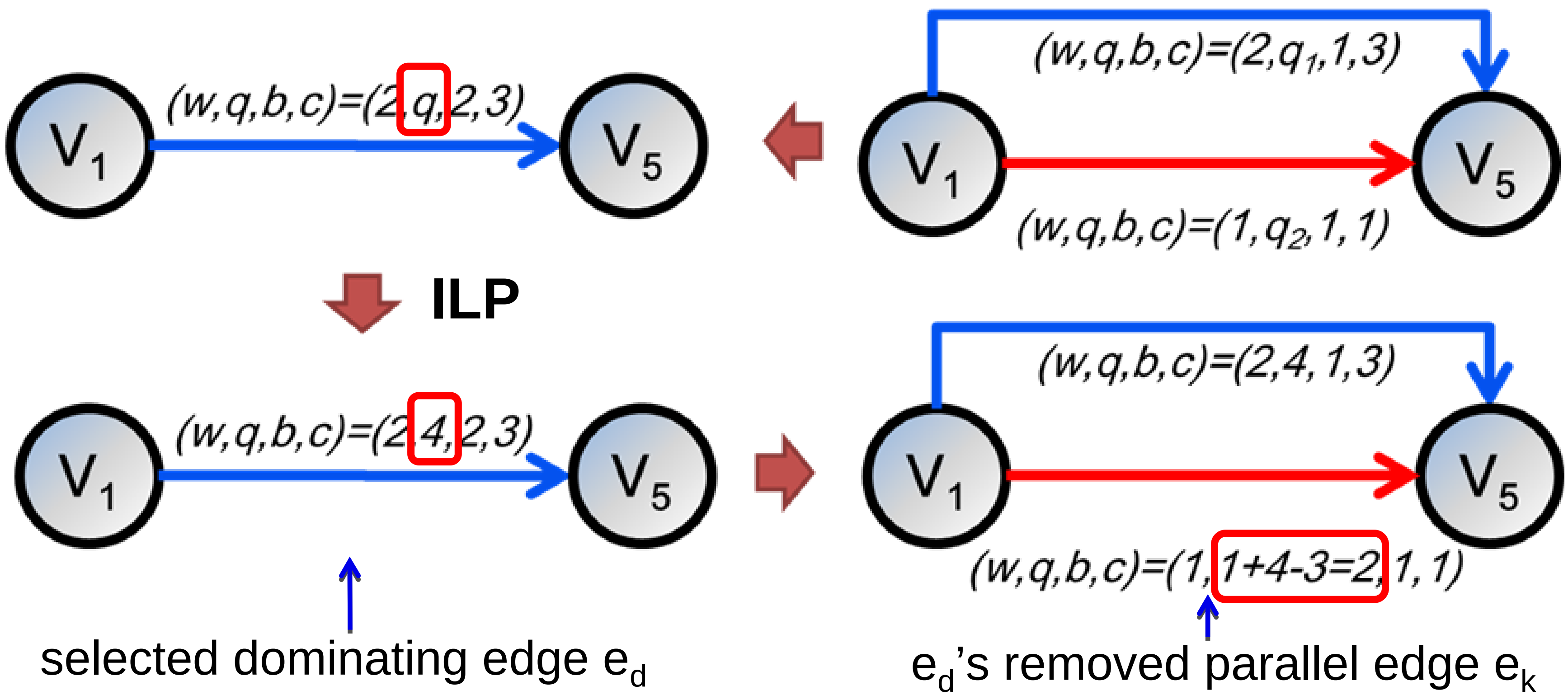
$$\sum_{e \in C} TPD_C(e) \geq 0 \quad \text{for every cycle } C \text{ in } H$$

$$w(e) + q(e) - 2 \times c(e) \geq 0, \quad \text{for every edge } e \text{ in } H$$

- This approach can still handle reasonably large systems within an acceptable runtime

Recovery Phase - Edge Split

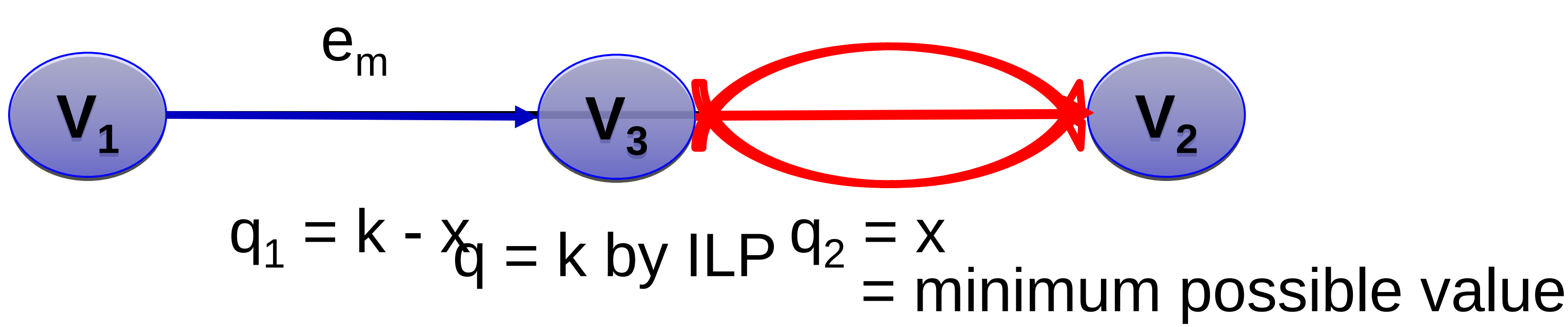
- Edge Split
  - Rebuild multi-edges form edge unification
  - To ensure $TPD_C(e) \geq 0$ for any newly generated cycle
    - $q_k - c_k \geq q_d - c_d$ (or $q_k \geq c_k + q_d - c_d$)



Recovery Phase - Path Expansion (1/2)

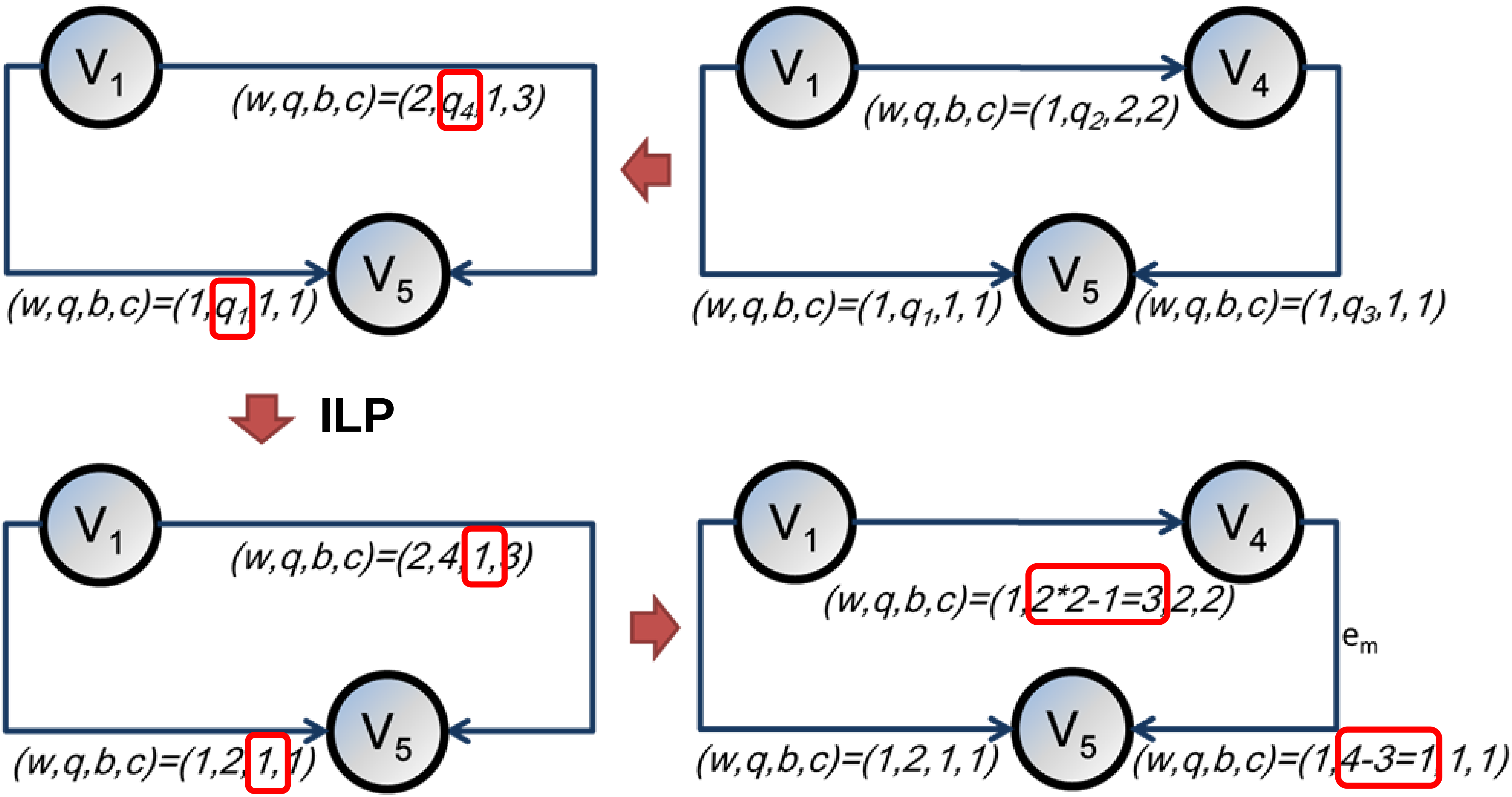
- The way for distributing $q(e_p)$ to those edges along p is not unique
  - Need to guarantee minimal queue insertion
  - Let $e_m \in E(p)$ be the edge with lowest burden factor along a condensable path p

$$q(e) = \begin{cases} 2 \times c(e) - w(e), & \text{for } e \neq e_m \\ q(e_p) - \sum_{e \in E(p),\, e \neq e_m} q(e), & \text{for } e = e_m \end{cases}$$



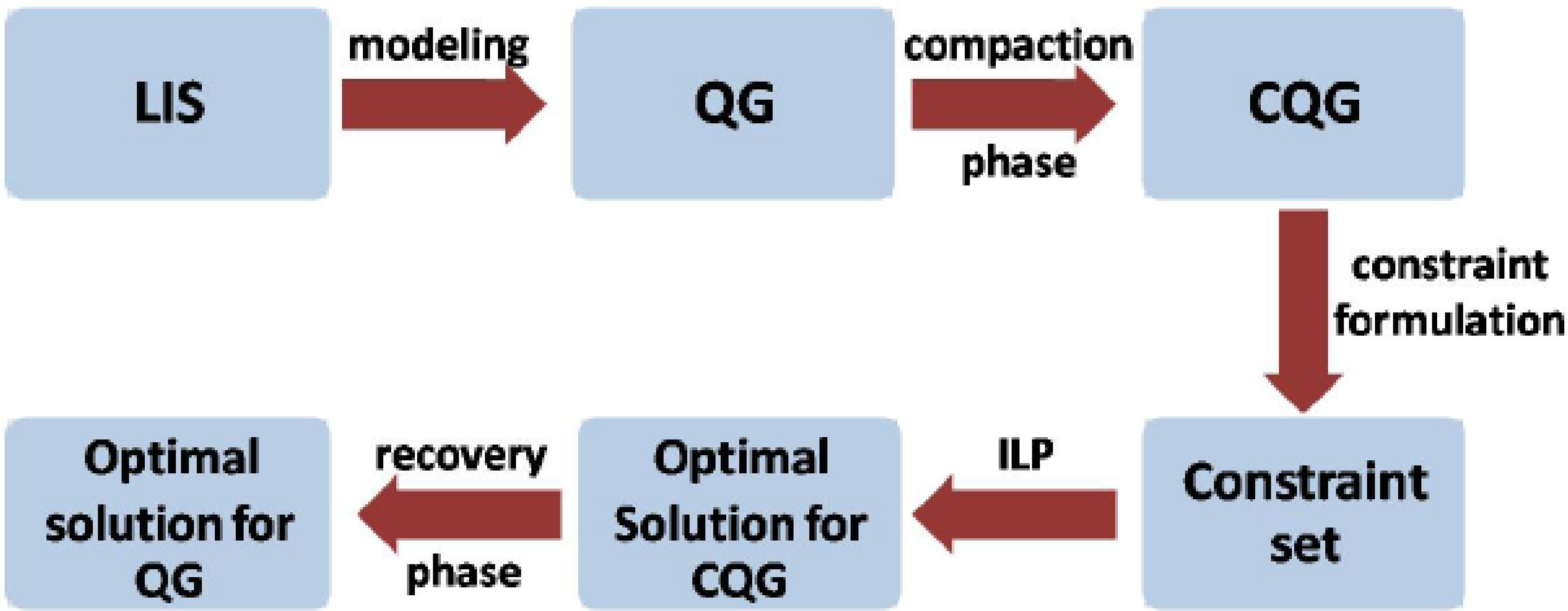
Recovery Phase - Path Expansion (2/2)

- Example



Overall Flow

- The overall flow of our proposed method for minimal queue insertion



Outline

- Introduction
- Preliminaries
  - Latency-Insensitive System (LIS)
  - Marked Graph (MG)
- Proposed Queue Sizing Method
  - Quantitative Graph (QG) and Compacted QG (CQG)
  - Compaction Phase
  - ILP Formulation
  - Recovery Phase
- Experimental Results
- Conclusions

Experimental Results (1/3)

- The proposed technique can successfully reduce the number of vertices and edges

Case Name	Original QG		Minimal CQG	
	(V, E)	#Cycles	(V, E)	#Cycles
Testcase1	(11,15)	55	(8,11)	12
Testcase2	(17,21)	51	(13,17)	14
Testcase3	(45,61)	30540	(20,35)	10123
Testcase4	(58,76)	48590	(39,45)	10497
Testcase5	(104,121)	42435	(56,73)	19754
Testcase6	(126,172)	> 1Million	(77,98)	132415
Testcase7	(175,201)	> 1Million	(66,84)	15423
Testcase8	(297,318)	> 1Million	(116,142)	23862

Experimental Results (2/3)

- Latency of every edge is also randomly assigned with an integer within the interval [0, *L-1*]

L	L=3					
Case Name	Proposed Method		Collins' Method [12]		ILP directly to QG	
	#Queues	Run-time	#Queues	Run-time	#Queues	Run-time
Testcase1	20	0	20	0	20	1
Testcase2	9	0	9	0	9	0
Testcase3	51	5	80	4	51	14
Testcase4	43	14	46	13	43	44
Testcase5	29	40	78	27	29	340
Testcase6	77	867	90	542	*	*
Testcase7	84	32	90	23	*	*
Testcase8	114	73	141	47	*	*
Ratio	0.77	1.57	1	1	-	-

Experimental Results (3/3)

- The improvement can slightly increase as fabrication process keeps scaling

L	L=16					
Case Name	Proposed Method		Collins' Method [12]		ILP directly to QG	
	#Queues	Run-time	#Queues	Run-time	#Queues	Run-time
Testcase1	68	1	68	0	68	1
Testcase2	76	0	77	0	76	0
Testcase3	290	9	437	6	290	19
Testcase4	291	31	351	19	291	52
Testcase5	256	77	386	48	256	459
Testcase6	519	1438	793	913	*	*
Testcase7	673	69	753	40	*	*
Testcase8	641	131	1035	83	*	*
Ratio	0.72	1.58	1	1	-	-

Conclusions

- In this work, we proposed
  - A new representation for LIS: quantitative graph (QG)
  - Two compaction techniques: QG ➔ CQG
  - The optimal solution on CQG can be achieved via ILP
  - Two recovery techniques: CQG ➔ QG
- The experimental results show that
  - The number of cycles can be reduced significantly
  - We can handle moderately large systems in acceptable runtime even using ILP
  - Up to 28% reduction in queue size as compared to the prior art

Thank You!