An O(nm) Time Algorithm for Optimal Buffer Insertion of m Sink Nets

Zhuo Li and Weiping Shi

{zhuoli, wshi}@ece.tamu.edu

Texas A&M University

College Station, Texas 77845

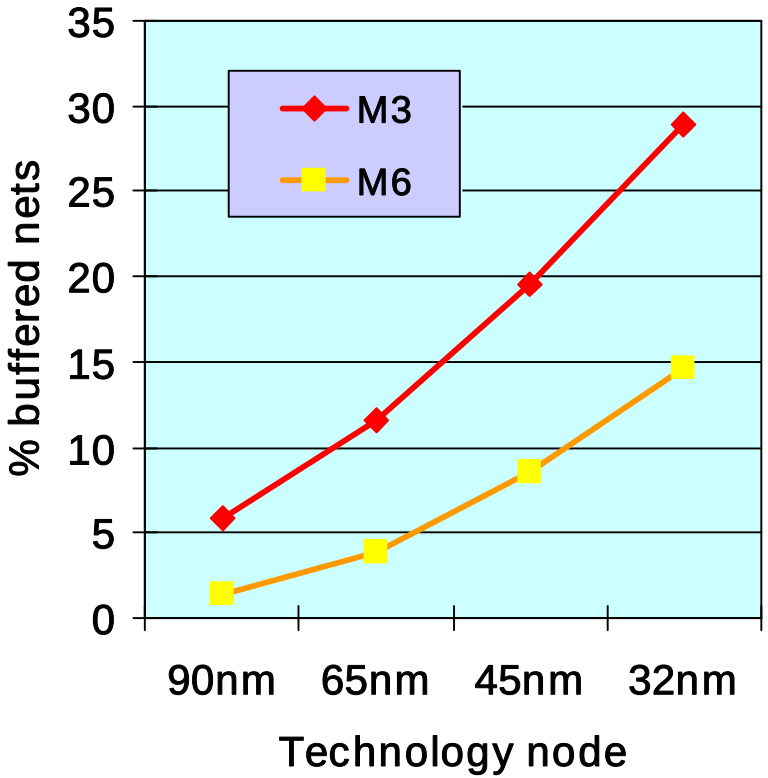
USA

Outline

- Introduction
- O(n) Time Algorithm for 2-pin Nets
- O(mn) Time Algorithm for m-pin Nets
- Experimental Results
- Conclusion

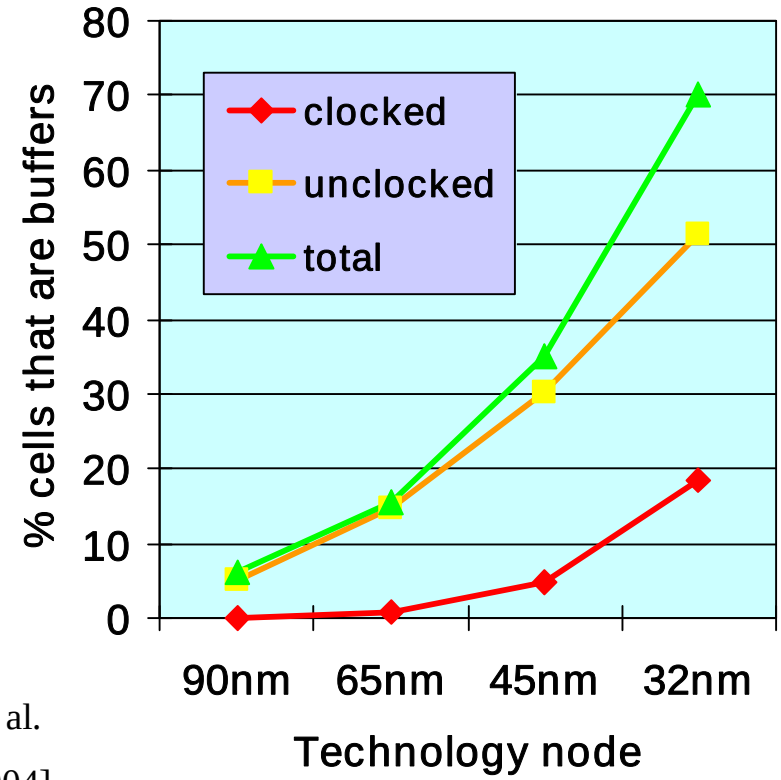
Introduction

- Buffer insertion and sizing is an effective method to reduce interconnect delay.
- With large number of nets and buffer positions, fast algorithms are crucial.



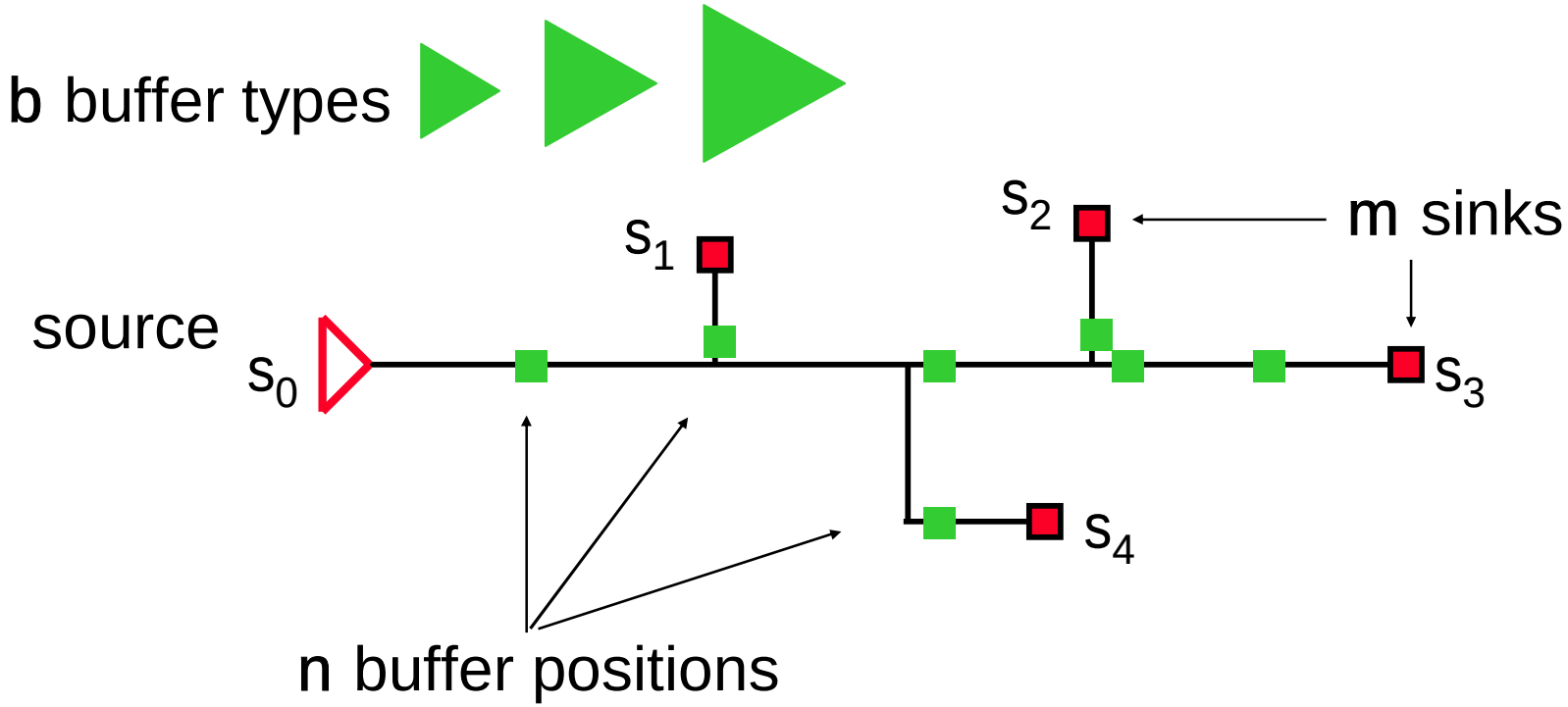
Saxena, et al.

[TCAD 2004]



Problem Formulation

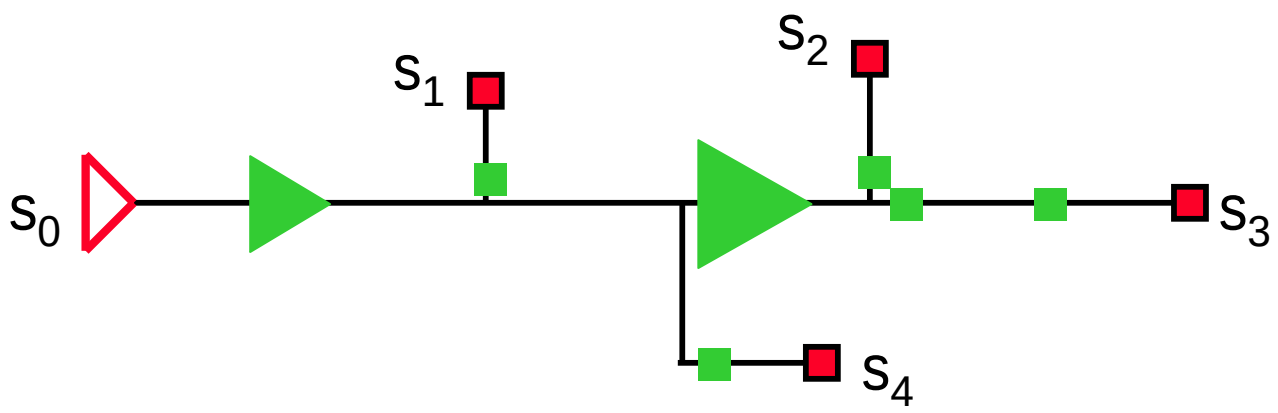
- Given: A routing tree, possible buffer positions, sink capacitances and required arrival times (RAT), wire resistance and capacitance.
- Delay model: Elmore delay for interconnect and linear delay model for buffers.



Maximum Slack Problem

- Find: Where to insert buffers so that the slack at the source $Q(s_0)$ is maximized.

$$Q(s_0) = \min_{i>0} \{ RAT(s_i) - delay(s_0, s_i) \}$$



without buffer, $Q(S_0)$= – 50 ps

with 2 buffers, $Q(S_0)$= 100 ps

Previous Research

- Maximum slack
  - van Ginneken [ISCAS 90]: $O(n^2)$ time for one buffer type, where n is the number of buffer positions.
  - Lillis, Cheng and Lin [TCAS 96]: $O(b^2n^2)$ time, where b is the number of buffer types.
  - Shi and Li [TCAD 05]: $O(b^2n\log n)$ time.
  - Li and Shi [TCAD 06]: $O(bn^2)$ time.
- Minimum cost (area, power, etc.)
  - Lillis, Cheng and Lin [TCAS 96]: pseudo-polynomial time algorithm.
  - Shi, Li and Alpert [ASPDAC 04]: buffer cost minimization is NP-hard if b is a variable.

Sinks m and Buffer Positions n

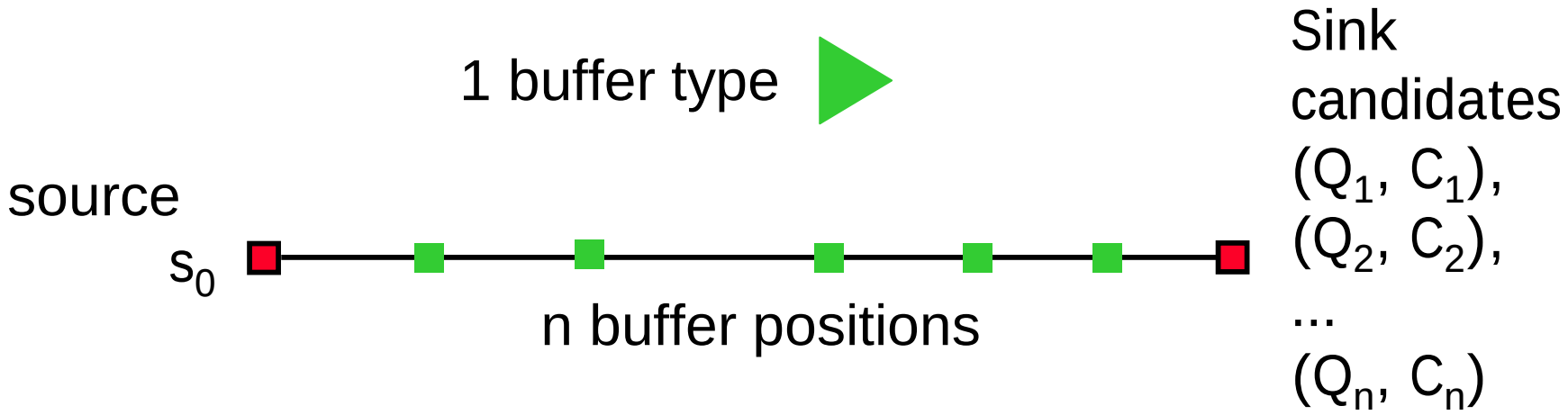
- All previous research assumes m and n are same order.
- In practice, m << n
  - In one IBM chip, 95% of the most time-consuming nets have less than 5 sinks, while n is hundreds to thousands.
  - In another IBM chip, 80% of the most time-consuming nets have less than 50 sinks.
- In this paper, we propose
  - Simple algorithms and data structures.
  - $O(b^2n)$ time for 2-pin nets. First linear time algorithm in terms of n
  - $O(bmn+b^2n)$ time for m-pin nets.

Outline

- Introduction
- O(n) Time Algorithm for 2-pin Nets
- O(mn) Time Algorithm for m-pin Nets
- Experimental Results
- Conclusion

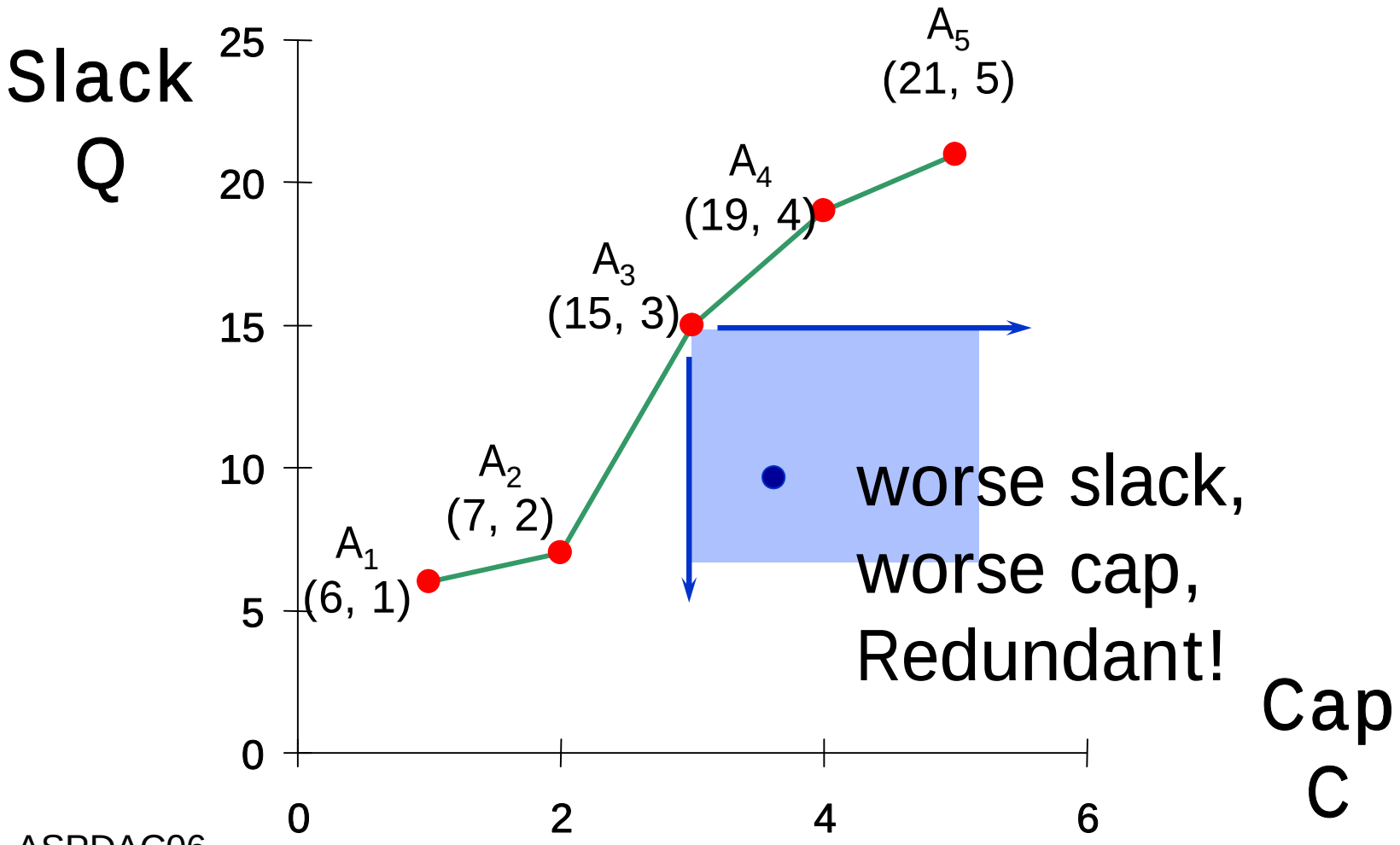
O(n) Algorithm for 2-pin Nets

- Given a 2-pin net with n buffer positions, 1 buffer type, R(w) and C(w) for each wire, and sink non-redundant candidates (Q_1, C_1), …, (Q_n, C_n) in sorted order.
- Compute non-redundant candidates at source



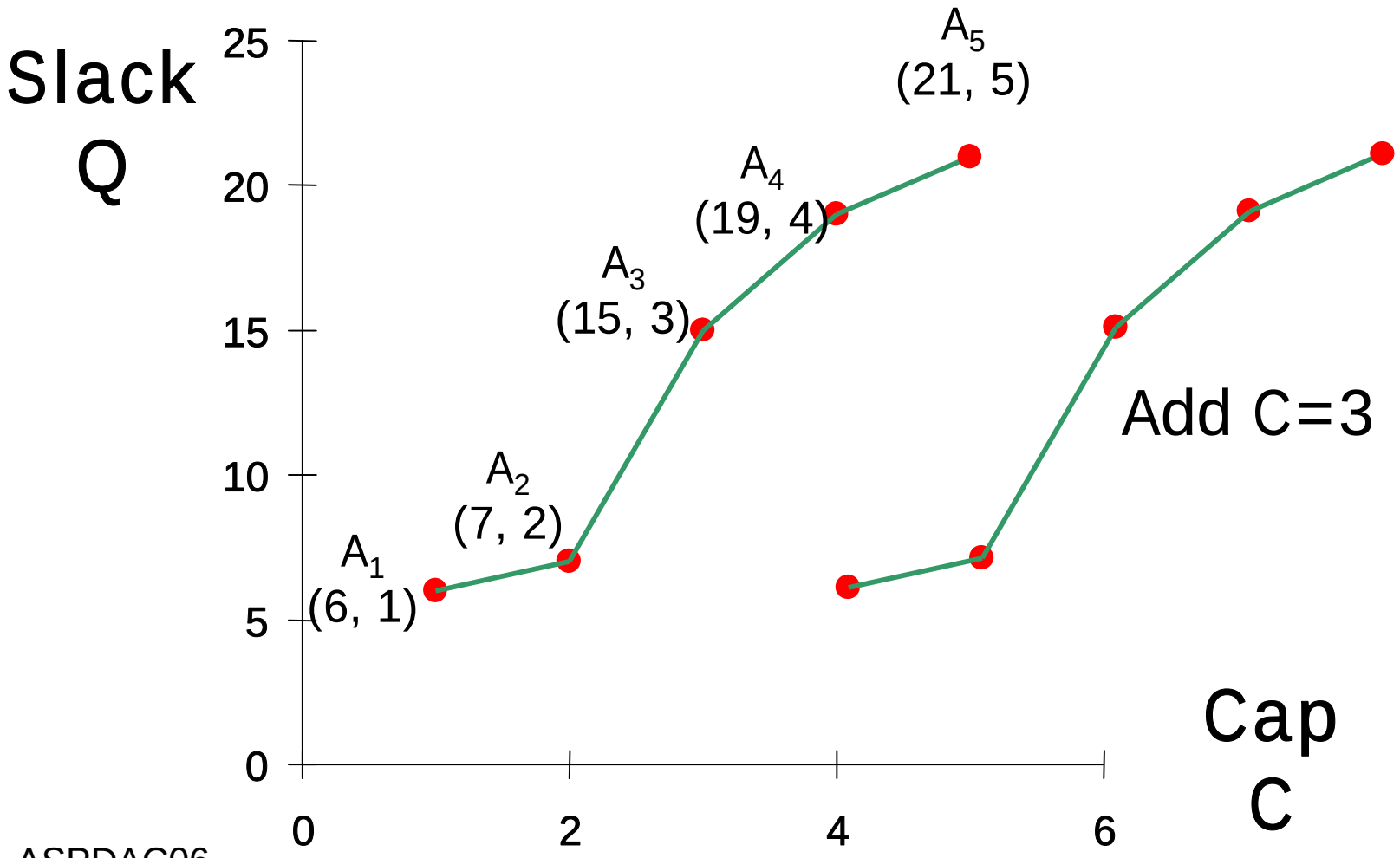
New View of Candidates

- Non-redundant candidates form a monotonically increasing sequence in (Q, C) plane.



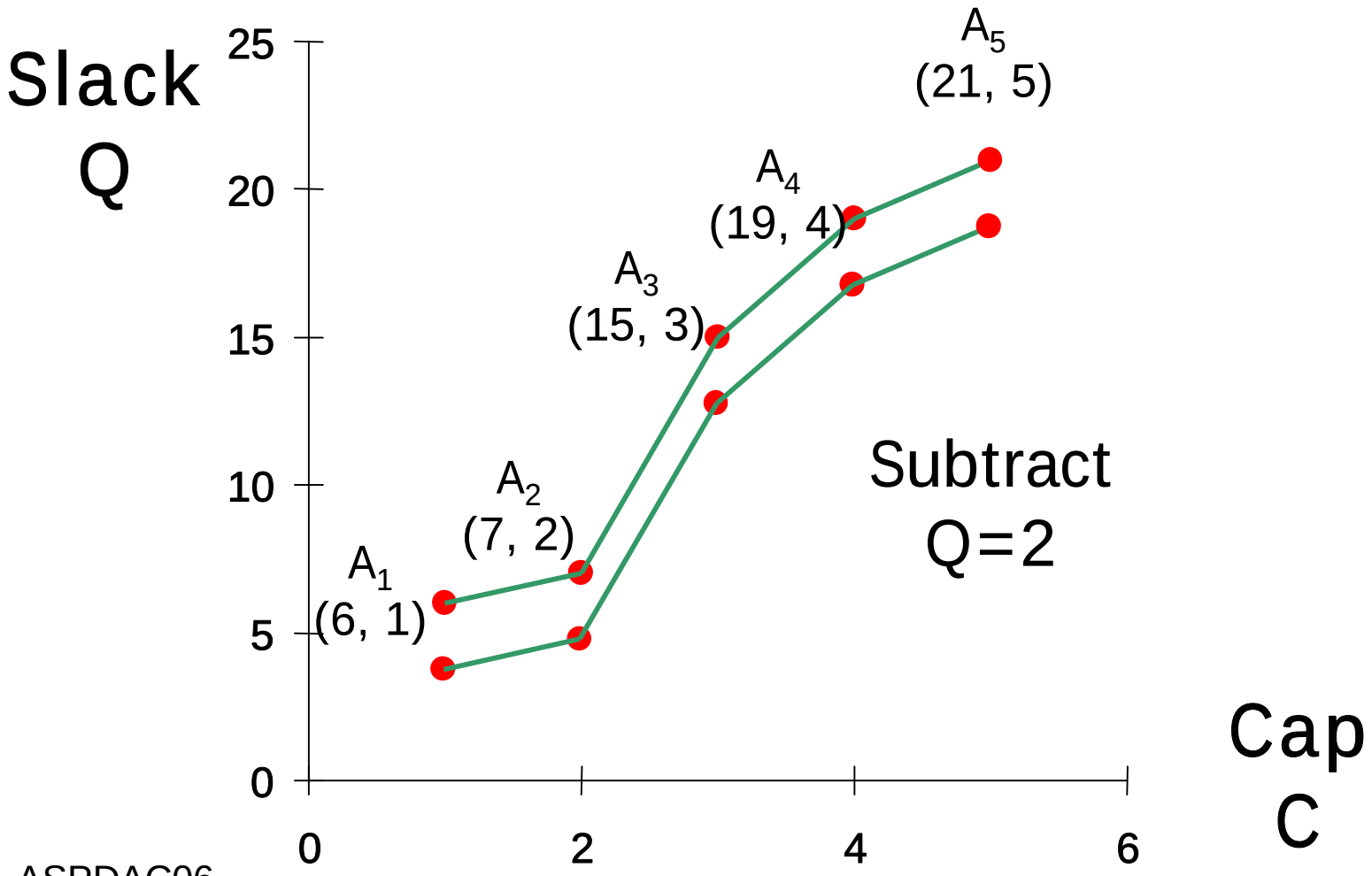
Add C to All Candidates

- When we add capacitance to all candidates, they shift to right in the (Q, C) plane.



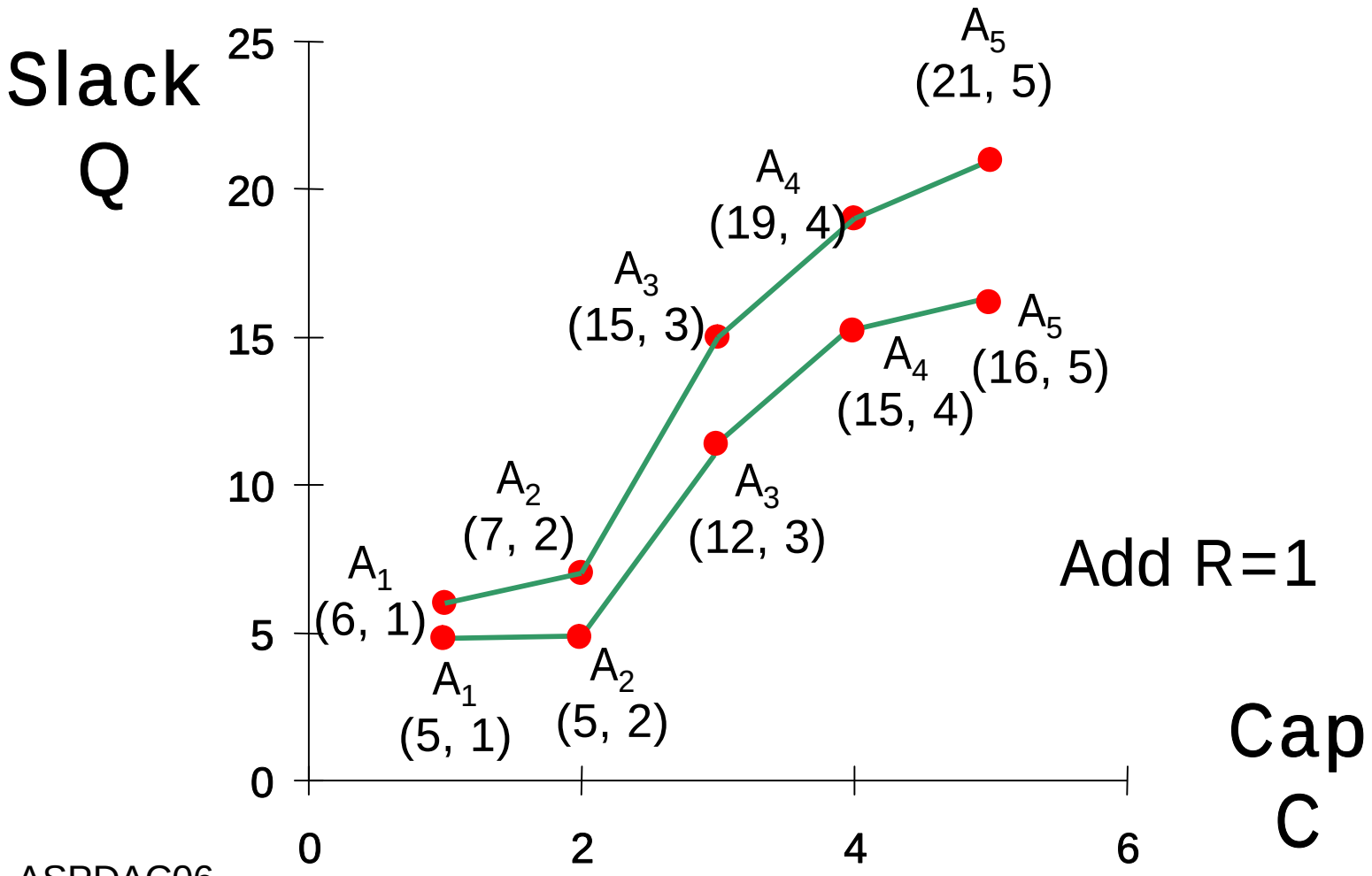
Subtract Q from All Candidates

- When we subtract Q from all candidates, they shift down in the (Q, C) plane.

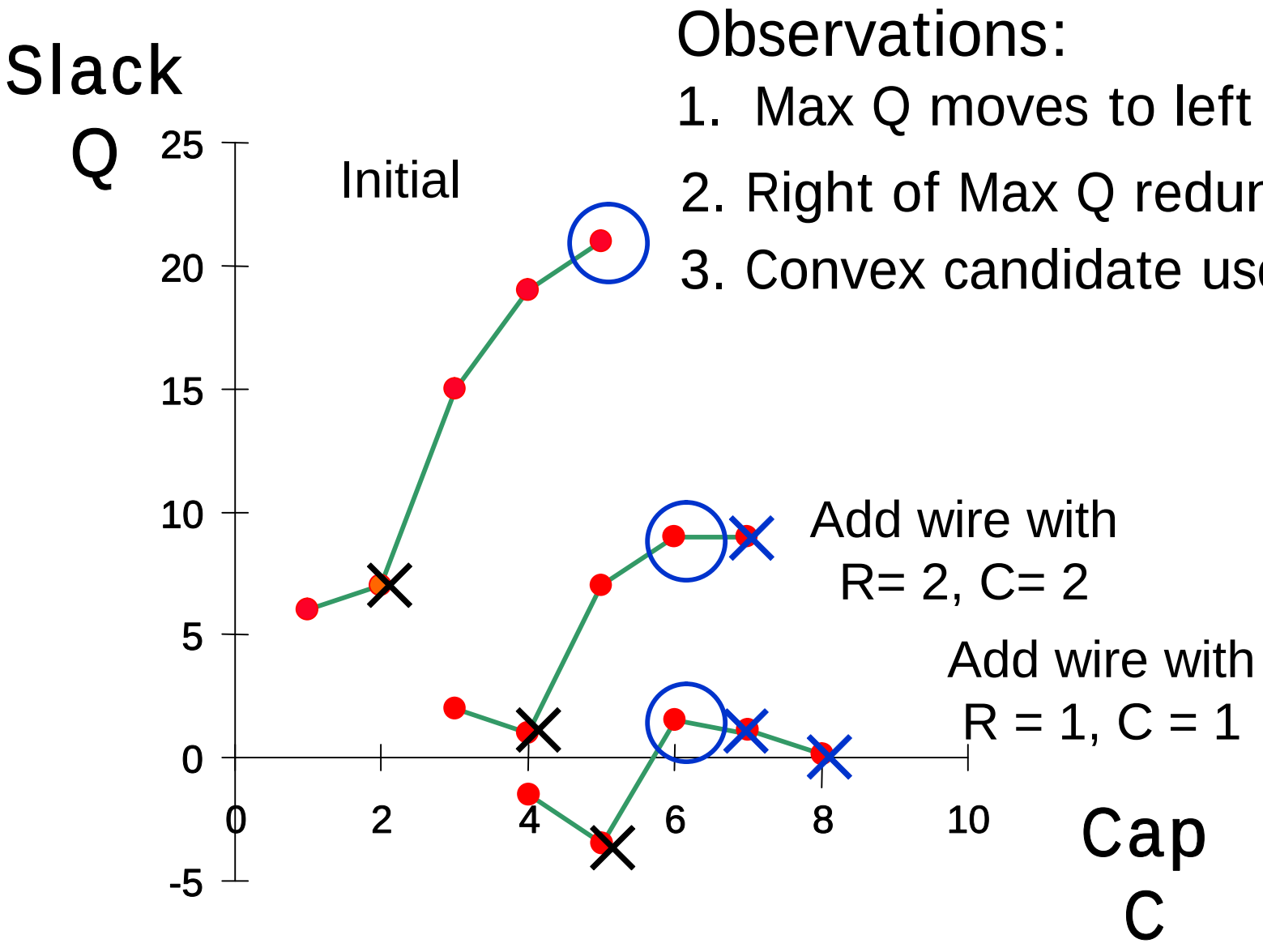


Add R to All Candidates

- When we add resistance to all candidates, they move right and down in the (Q, C) plane.



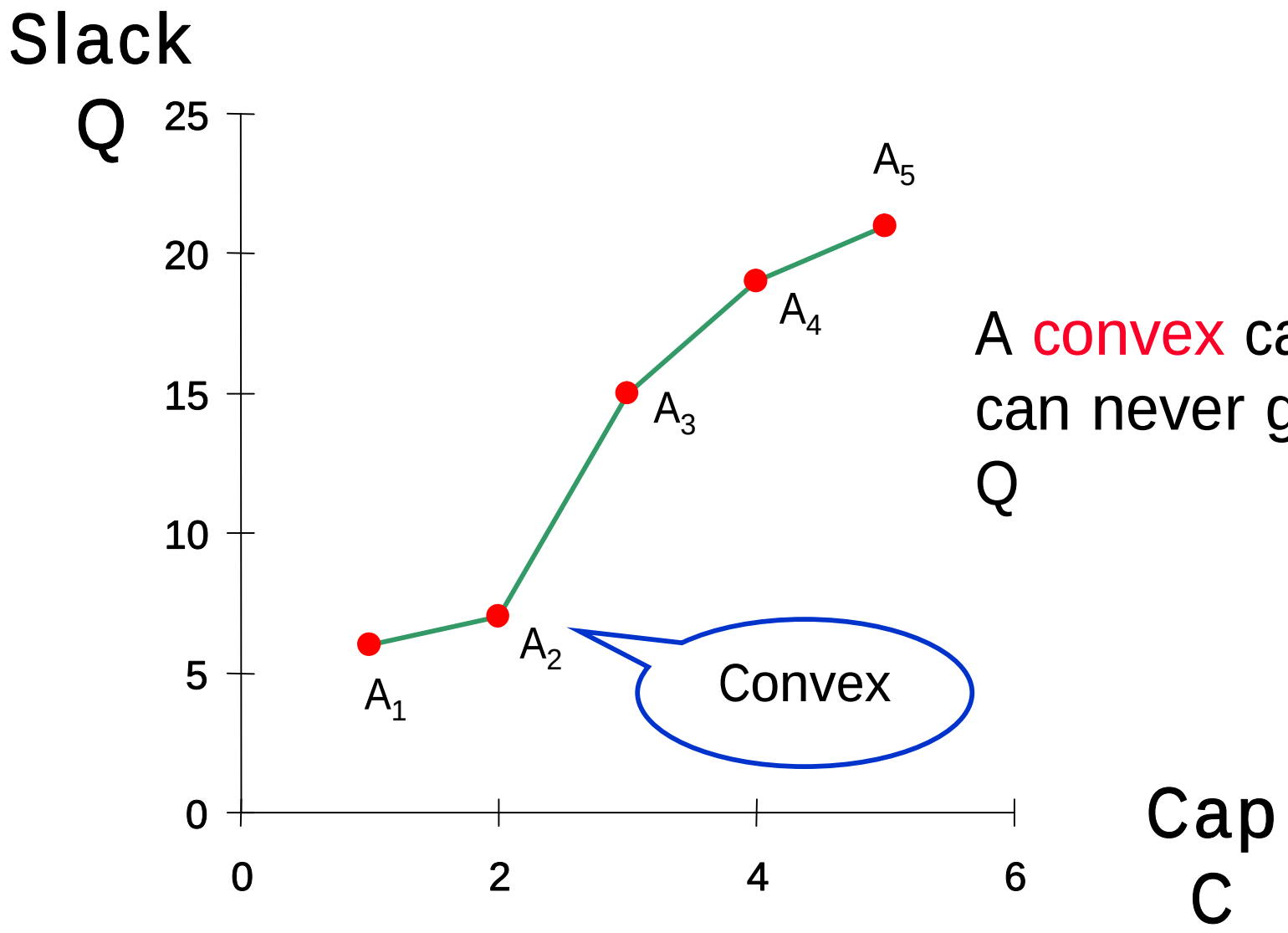
Combined Effect of Add Wires



Observations:

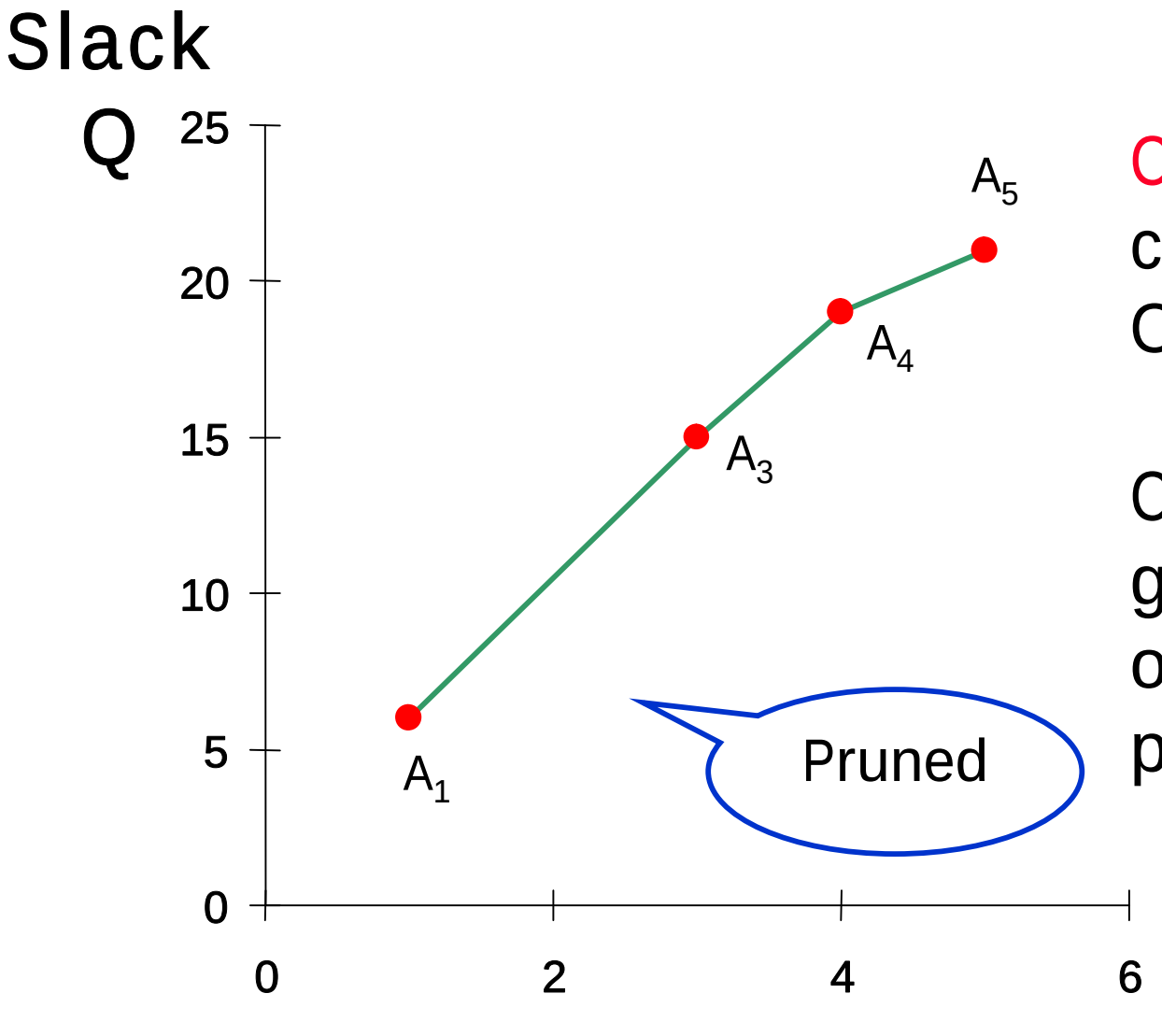
1. Max Q moves to left
2. Right of Max Q redundant
3. Convex candidate useless

Convex Pruning



A convex candidate can never give max Q

Convex Pruning

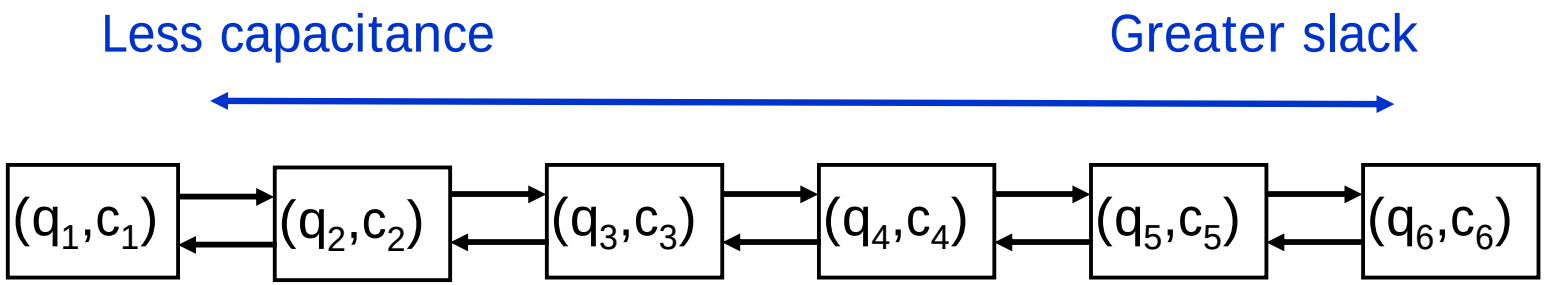


Convex pruning can be done in O(n) time.

Convex pruning guarantees optimality for 2-pin nets

Linked List Data Structure

- Non-redundant and convex pruned
- In increasing Q and increasing C order
- We store slack and capacitance of each candidate implicitly
  - Three global variables Qa, Ca and Ra that are updated as wires and buffers are added
  - Each candidate also has (q, c) pair that are never updated



(Q, C) Implicitly Stored

- Global variables that are updated
  - Qa is accumulated wire delay
  - Ca is accumulated wire capacitance, and
  - Ra is accumulated wire resistance
- Each candidate also has (q_i, c_i) that are not updated
- (Q, C) calculation
  - Slack $Q_i = q_i - Qa - Ra*c_i$
  - Capacitance $C_i = c_i + Ca$
- For example (q, c) = (15, 1), (19, 2), (21, 3)
  - If Qa=2, Ca=1, Ra=0, then
    (Q, C) are (15–2, 1+1), (19–2, 2+1), (21–2, 3+1)
  - If Qa=0, Ca=0, Ra=1, then
    (Q, C) are (15–1*1, 1), (19–1*2, 2), (21–1*3, 3)

Add Wire

- For a wire of R(w) and C(w), we update (Q, C) values of all candidates in O(1) time, by updating only Qa, Ca and Ra:
  - Qa = Qa + R(w)*C(w)/2 + R(w)*Ca
  - Ca = Ca + C(w)
  - Ra = Ra + R(w)
- For example (q, c) = (15, 1), (19, 2), (21, 3) and Qa=1, Ca=1, Ra=2
  - Add a wire R(w)=2, C(w)=1, then
  - Qa = 1 + 2*1/2 + 2*1 = 4
  - Ca = 1 + 1 = 2
  - Ra = 2 + 2 = 4

Add Buffer

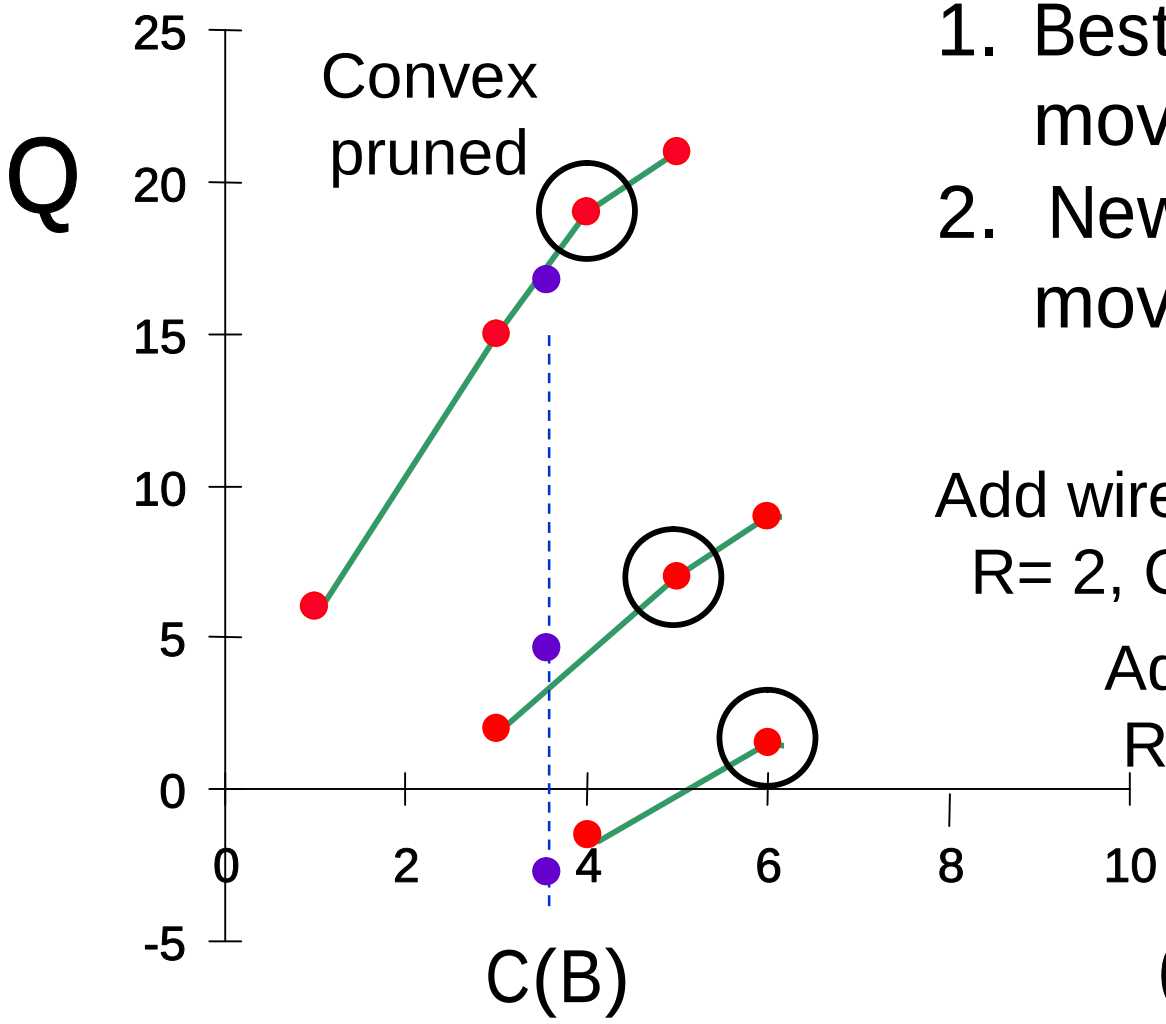
- Assume only one buffer type B for now. Let R(B) be driver resistance, C(B) be input capacitance, and t(B) be intrinsic delay.
- Define the best candidate α as the candidate that maximizes slack among all candidates after B is inserted.
- Define the new candidate β as the candidate formed by α with the buffer B

$$Q(\beta) = Q(\alpha) - R(B)C(\alpha) - t(B),$$

$$C(\beta) = C(B).$$

Buffer delay

(Q, C) Plane View: Add Buffers



Observations:

1. Best candidate moves to left
2. New buffer position moves to left

Add wire with
R= 2, C= 2

Add wire with
R = 1, C = 1

Form and Insert New Candidates

- All n best candidates can be found in O(n) time
  - Best candidate index always moves to left
  - Best candidates can be found by local search
- All n new candidate can be inserted into the data structure in O(n) time
  - Position of new candidate always moves to left
- To insert new candidate (Q, C) into the data structure, we set (q, c) values as

  $q = Q + Qa + Ra*C,$

  $c = C – Ca$
- It is now consistent with the implicit data structure

Algorithm for 2-Pin Nets

- Initia
  - der
  -
  -
- Rep
- For each wire w — O(1) — O(1) per deletion
  - Update Qa, Ca and Ra
  - Prune redundant candidates at right, if any
- For each buffer position — total O(n)
  - Search in decreasing Q order for the best candidate
  - Form a new candidate — O(1)
  - Search in decreasing C order for the position of new candidate — O(1) per del
  - Insert new candidate — O(1)
  - Perform local redundancy pruning and convex pruning

Total time O(n)
Total memory O(n)

2-Pin Nets with b Buffer Types

- Fo
  -
  - ft
- Th
  - One pointer for best candidate, moves to left
  - One pointer for new buffer position, moves to left
- Time complexity
  - At most O(bn) new candidates are inserted: O(bn)
  - At most O(bn) redundant candidates deleted: O(bn)
  - At most O(n) wires are added: O(n)
  - Each of the 2b pointers goes through the entire candidate list: $2b*O(bn) = O(b^2n)$

Total time $O(b^2n)$
Total memory $O(bn)$

Outline

- Introduction
- O(n) Time Algorithm for 2-pin Nets
- O(mn) Time Algorithm for m-pin Nets
- Experimental Results
- Conclusion

Multi-Pin Nets

- Convex pruning is not optimal under merging
  - Non-convex candidates could generate optimal solution with candidates from other merging branch
- Solution: Two lists
  - NR list for non-redundant candidates, for storing candidates
  - CP list for convex pruned candidates, for generating new candidates

Algorithm for m-Pin Nets

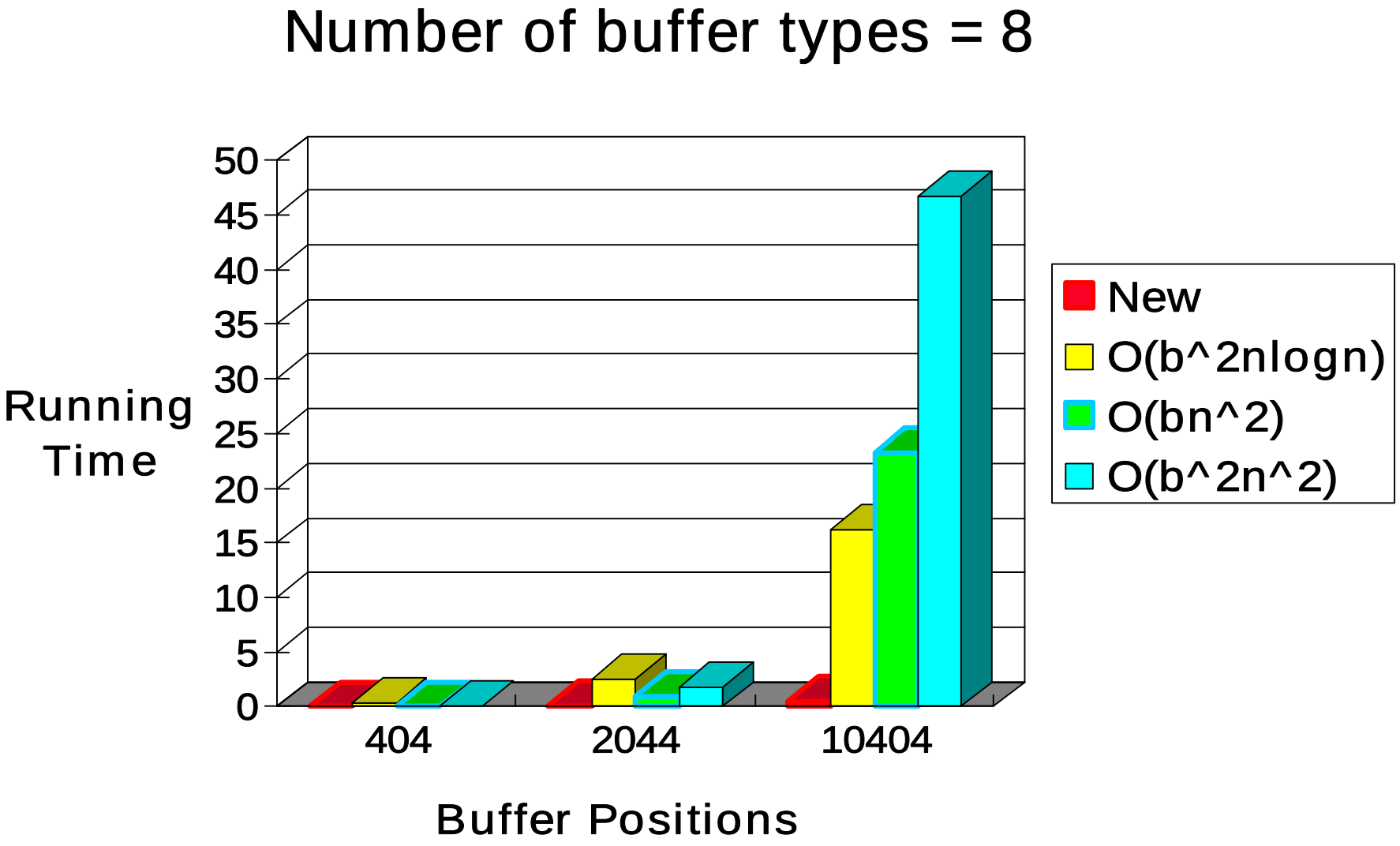
- Initial
- **Total time $O(b^2n+bmn)$**
- **Total memory $O(bn)$**
- For each 2-pin segment
  - Apply 2-pin algorithm on CP list
  - When add wire, add to both CP and NR
  - When insert new candidate, insert to both CP and NR
- For each merging point — O(b^2n)
  - Perform redundancy pruning for NR lists of two branches
  - Perform van Ginneken style merging of two NR lists, and create a new CP list

O(mbn)

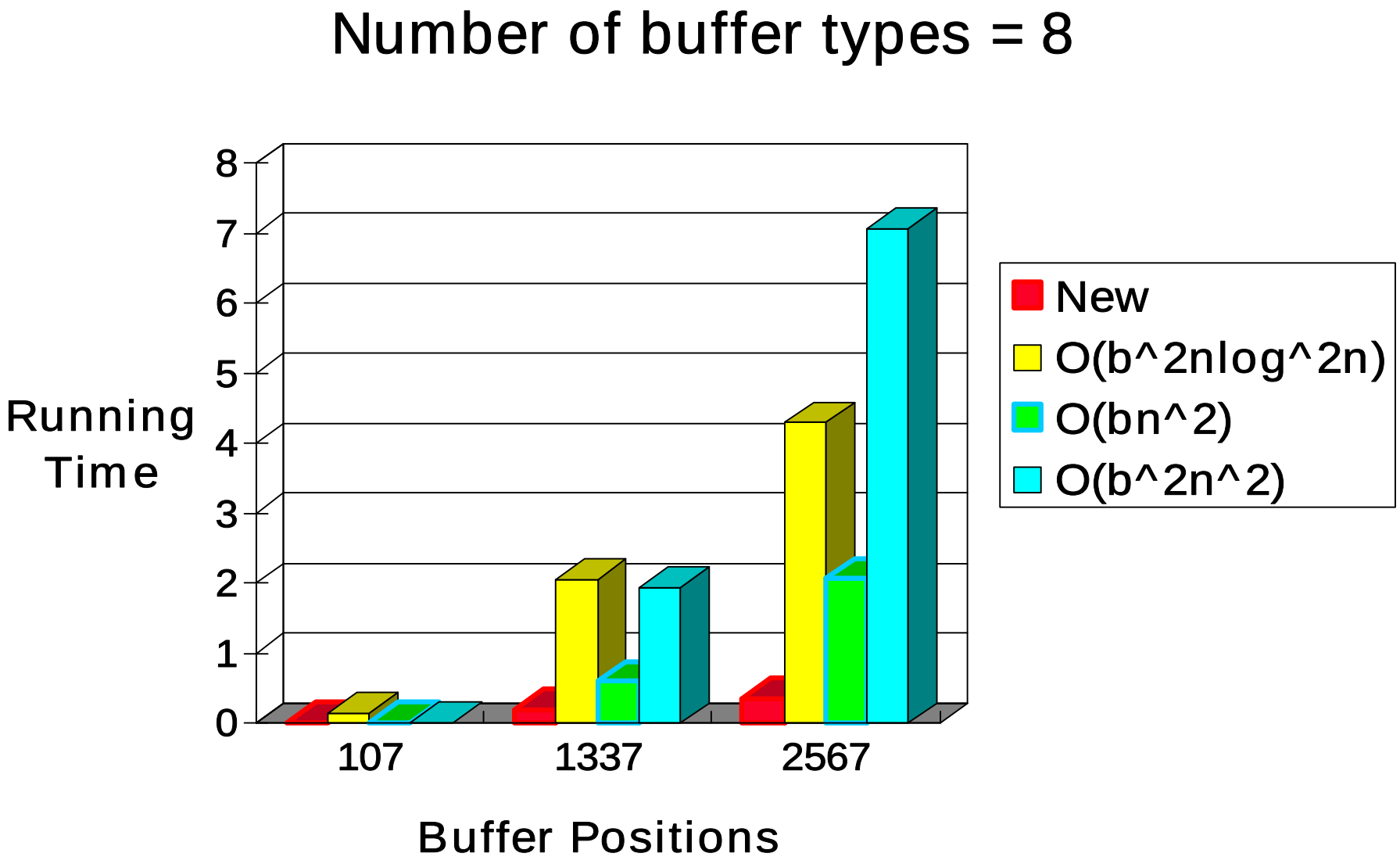
Outline

- Introduction
- O(n) Time Algorithm for 2-pin Nets
- O(mn) Time Algorithm for m-pin Nets
- Experimental Results
- Conclusion

Two-Pin Nets

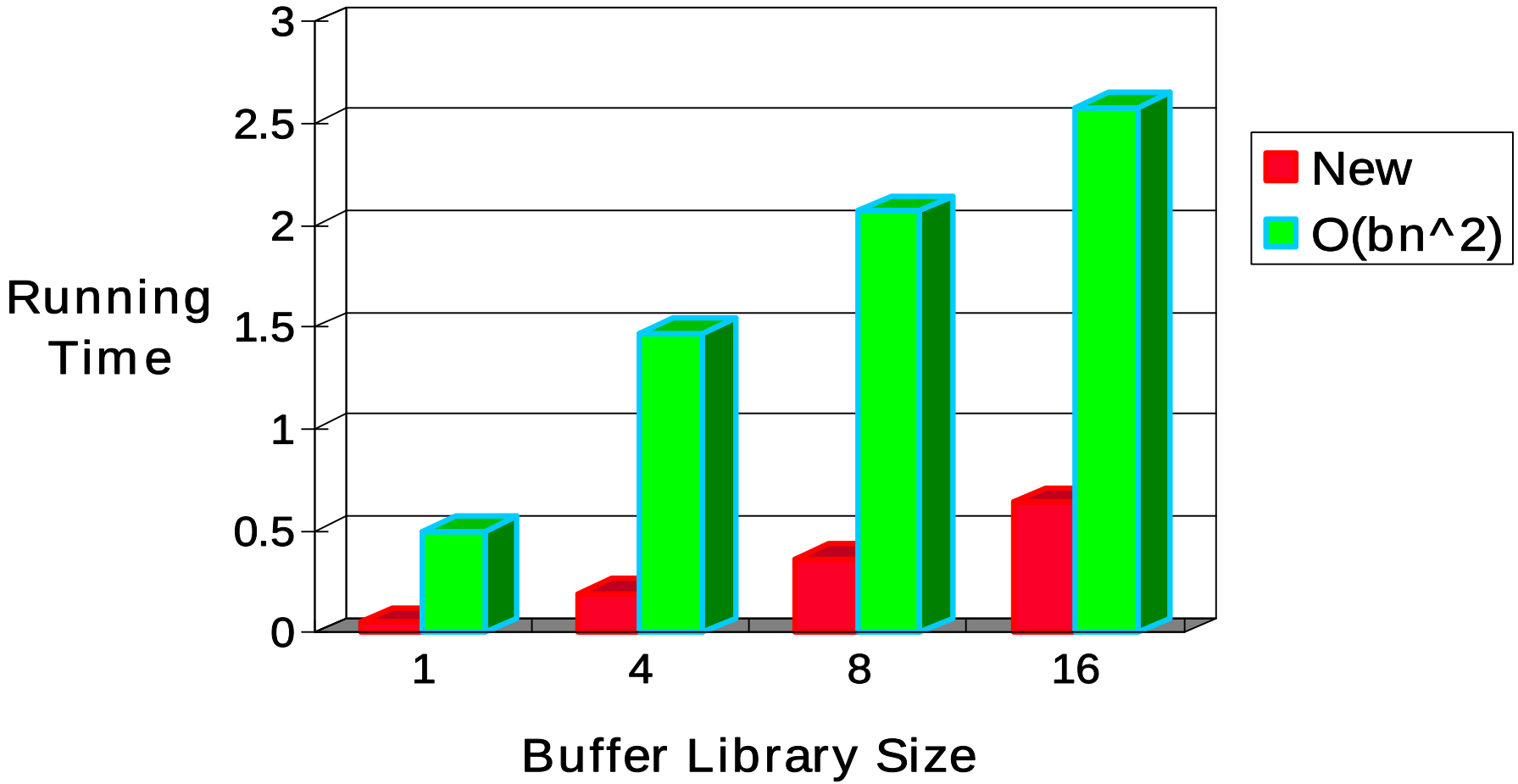

Number of buffer types = 8
Running Time
50
45
40
35
30
25
20
15
10
5
0
404
2044
10404
Buffer Positions
New
O(b^2nlogn)
O(bn^2)
O(b^2n^2)

Multi-Pin Nets (25 sinks)



Multi-Pin Nets (25 sinks)

Number of buffer positions = 2567



Conclusion

- New algorithm for optimal buffer insertion
  - $O(b^2n)$ for 2-pin nets
  - $O(bmn + b^2n)$ for m-pin nets
- Theoretical innovations
  - Simple data structure
  - Fast new candidates generation
  - Fast candidates insertion
- Practical applications
  - Simple and robust
  - Efficient on industrial test cases
- Extension to min cost buffer insertion and more general buffer delay models

m sinks and n buffer positions

- All previous algorithms assume m and n are of the same order.
- For most of nets in industrial applications, m is much less than n.
  - Among 1000 most synthesis-time-consuming nets in one IBM ASIC chip, 95% nets with sinks less than 5, and n is generally tens or hundreds times larger than m.
  - In another chip, among 5000 most time-consuming nets, 80% nets with sinks less than 50.
- O(mn) algorithm:
  - Two-pin nets: $O(b^2n)$ time.
  - Multi-pin nets: $O(b^2n + bmn)$ time.
  - Simple data structures.

The algorithm is linear respect to n.
Add more buffer positions to improve timing.