Reap What You Sow: Spare Cells for Post-Silicon Metal Fix

Kai-hui Chang[†]
Igor Markov[†‡]
Valeria Bertacco[†]

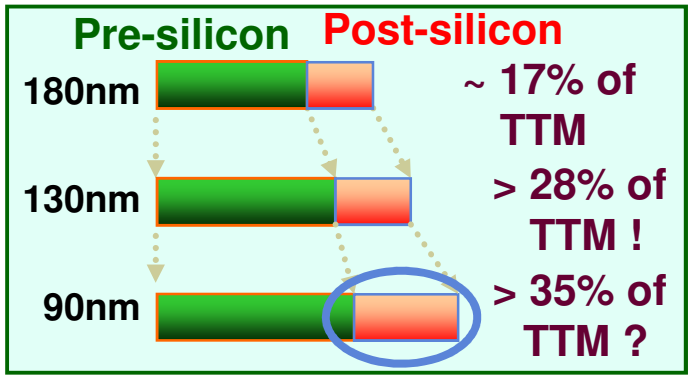
[†]University of Michigan at Ann Arbor

[‡]National Taiwan University

Apr. 15, 2008

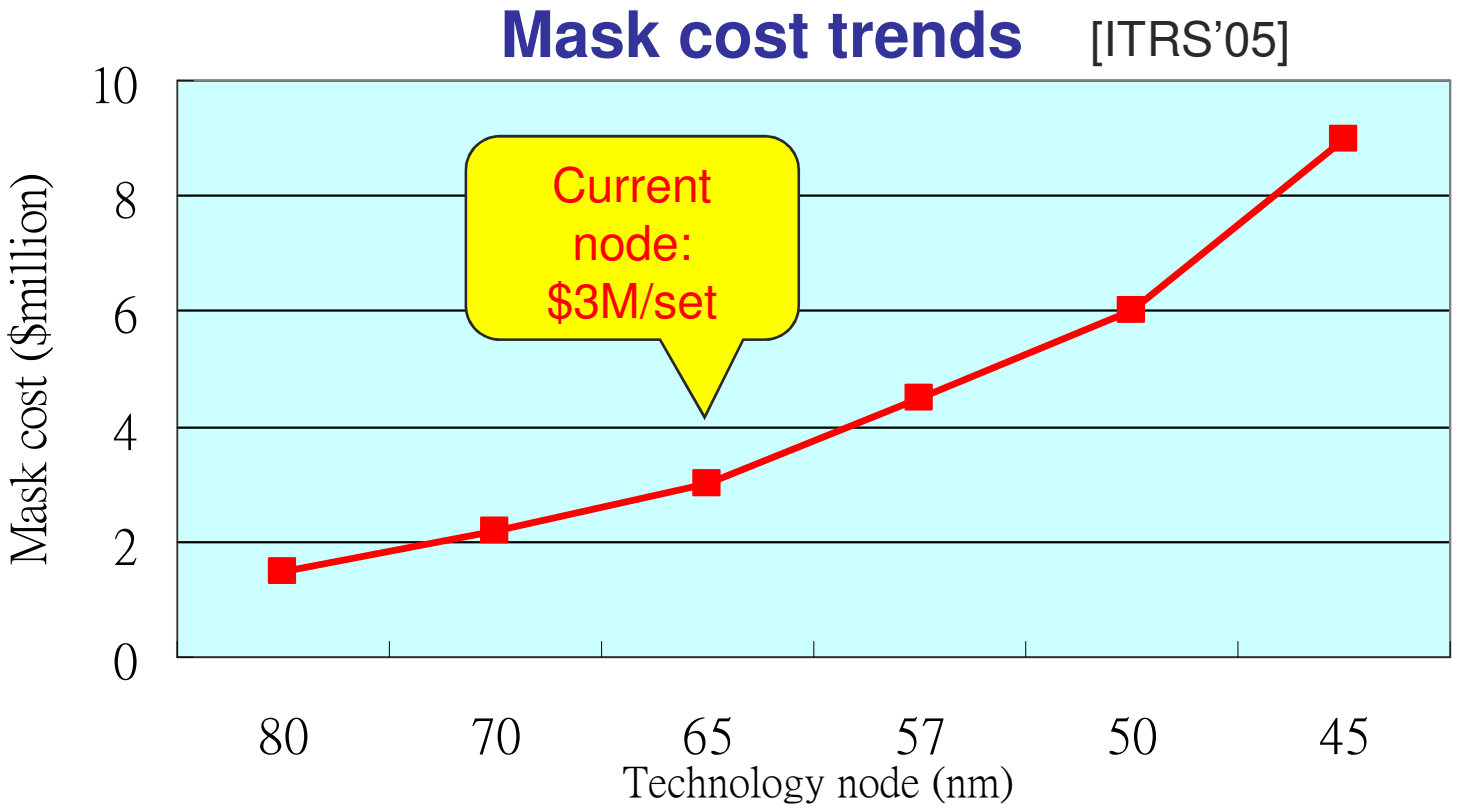
Current Design Challenges

- Due to looming design complexity, more bugs escape pre-silicon verification
  - Post-silicon validation and debugging are responsible for 35% of a chip's time to market
- High-profile bug escapes
  - Pentium – FDIV bug
  - AMD Phenom L3 cache bug
- Decreased time to market shortens verification time → more bugs in silicon
  - Post-silicon fix is growing in importance



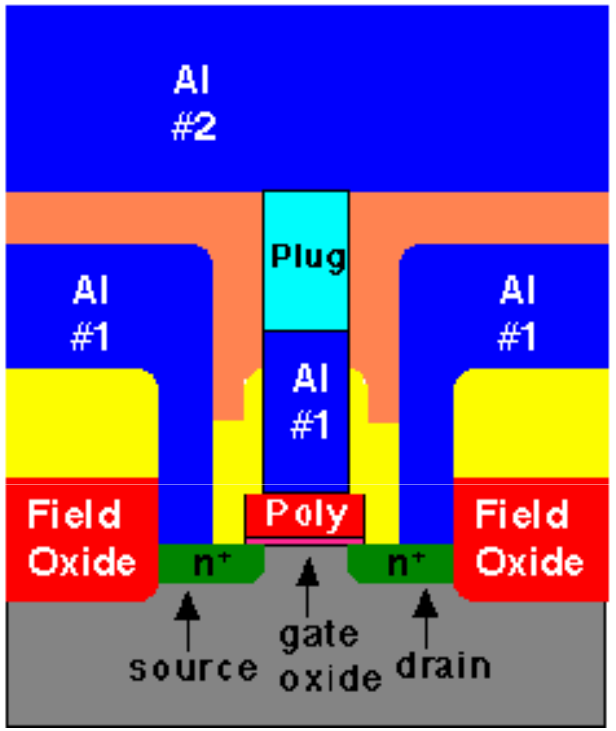
Dramatic Increase in Mask Costs

- Mask cost is increasing dramatically
  - $3M/set at 65nm node

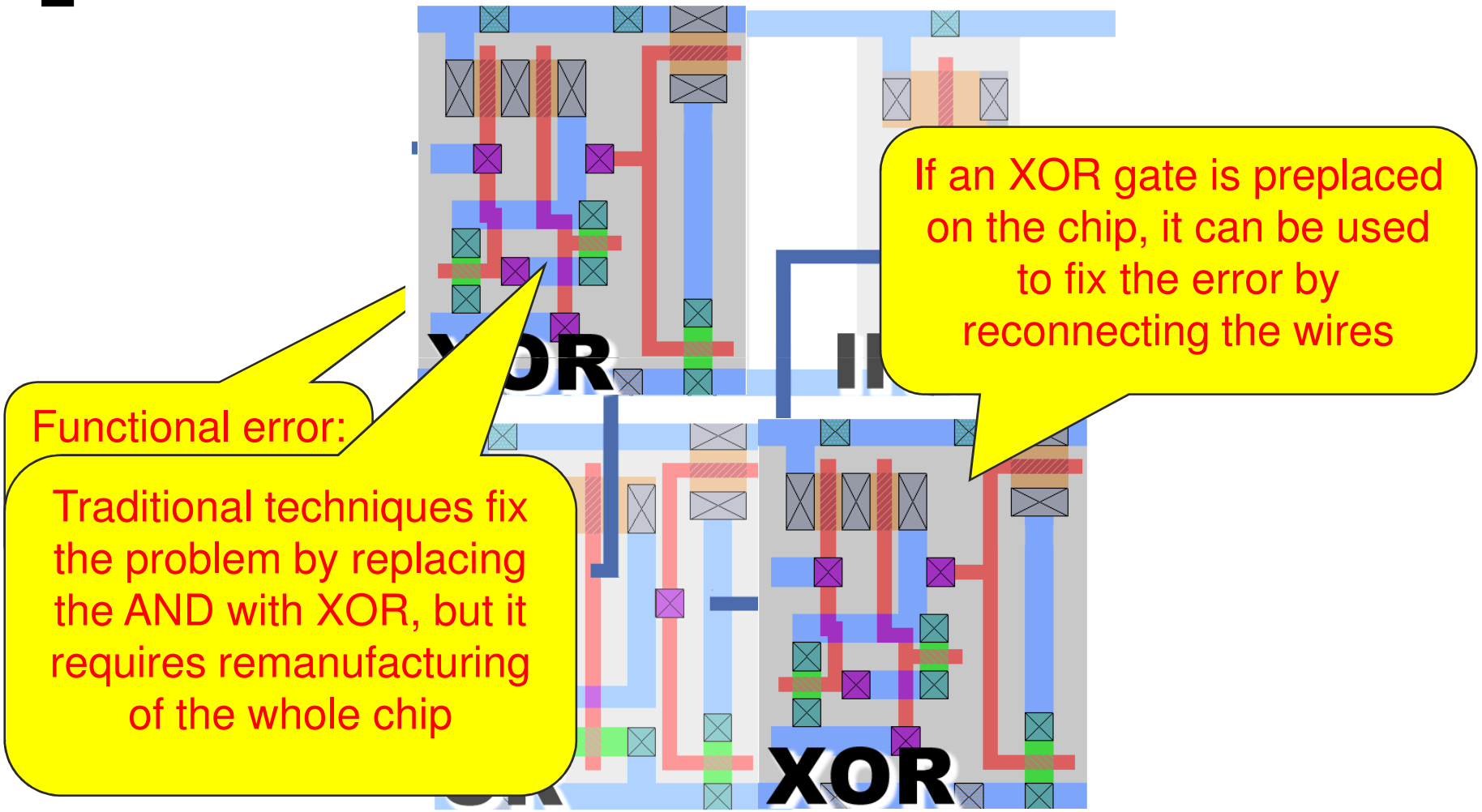


Dramatic Increase in Mask Costs

- Mask cost is increasing dramatically
  - $3M/set at 65nm node
- Transistor masks are most expensive
  ⇒ Reuse can reduce cost
  - **Only metal layers can be changed ⇒ *metal fix***
- Metal fix can be accomplished by
  - Respin of the chip
  - Focused Ion Beam (FIB) modifications of wires
- **No transistor can be changed in metal fix**



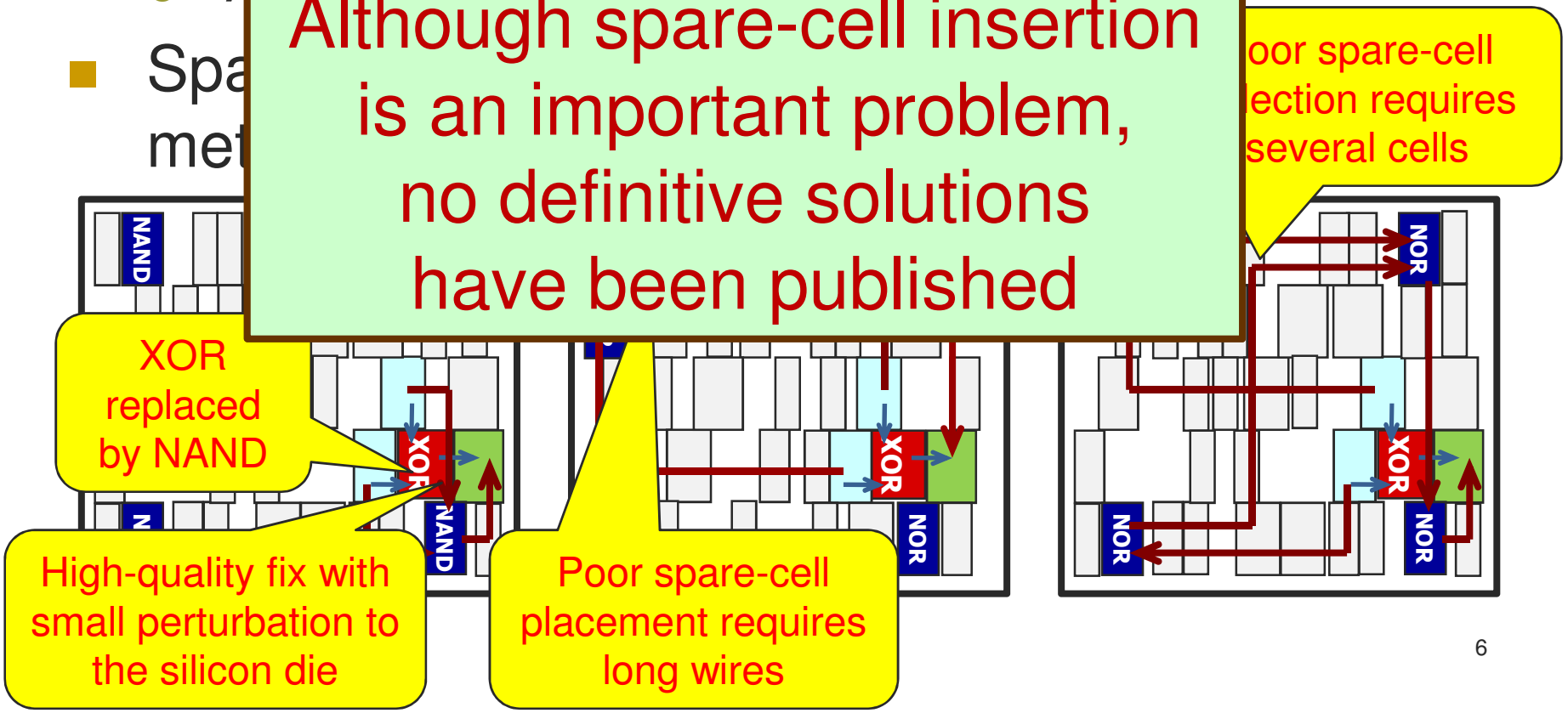
Traditional Fix vs. Metal Fix



Spare-Cell Insertion Problem

- To enable metal fix, *spare cells* need to be preplaced on the silicon die
  - A
- Spa
  met

Although spare-cell insertion is an important problem, no definitive solutions have been published



Why is Spare-Cell Insertion Difficult?

- Predict post-silicon bugs is difficult
  - Given a known bug, determining the best cells for the fix is easy
  - However, post-silicon bugs cannot be known in advance
- Need to considering both logical and physical information
  - Can be challenging because spare cells are disconnected from the netlist
  - Most existing logic synthesis and physical design tools cannot be utilized

Our Contributions

- Connect cell-type selection problem to logic synthesis – SimSynth
  - Measures heterogeneity among signals
  - Addresses cell density problem
- Handle spare-cell placement using physical design methods
- A novel spare-cell insertion methodology
  - Considers both logical and physical aspects
  - Covers both cell selection and placement
- First empirical study for spare-cell insertion

Outline

- Spare-cell selection: SymSynth
  - Based on logic simulation – fast
  - Adaptive to the needs of different design regions
- Spare-cell placement: UniSpare
  - Reduces impact of spare cells on circuit performance
  - Provides better metal fix quality
- Our spare-cell insertion methodology
- Experimental results
- Conclusions

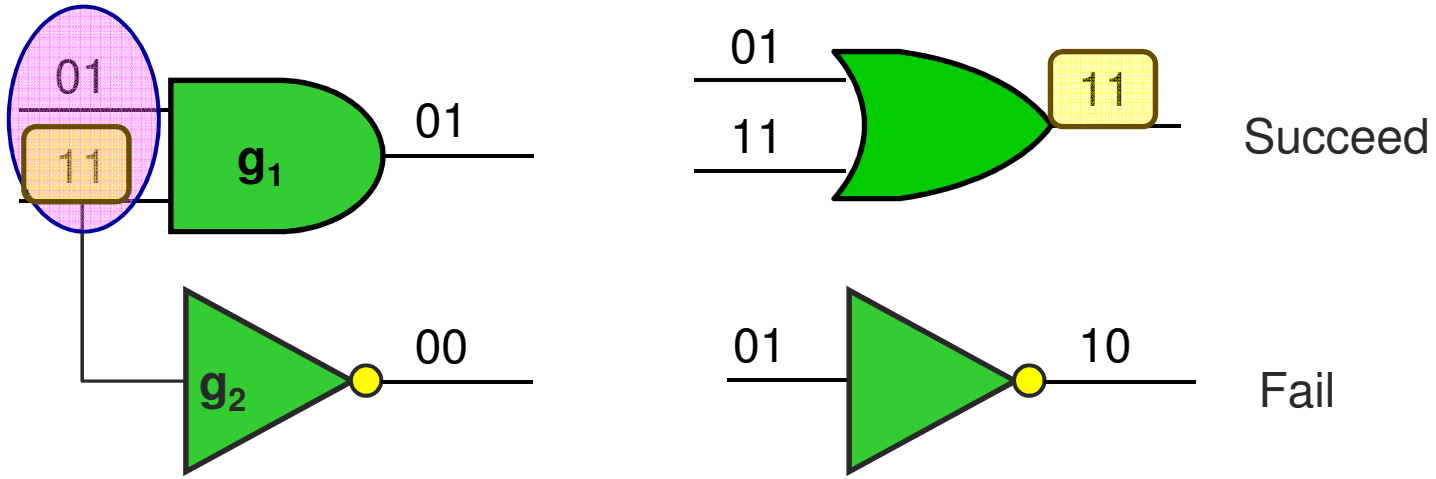
Spare-Cell Selection: SimSynth

- Goal: identify more useful cell types
- Based on the following observations
  - Bugs discovered post-silicon are often subtle bugs
  - To fix the bugs, the functionality of the circuit is only changed slightly

→ Cells that can generate signals *close* to existing ones are more useful

SimSynth Example

- Simulate input patterns to generate *signatures*
  - A bit in the signature is the simulation value of an input vector
  - It is the signal's partial truth-table
- Try each cell type and measure the rate to successfully replicate a signature



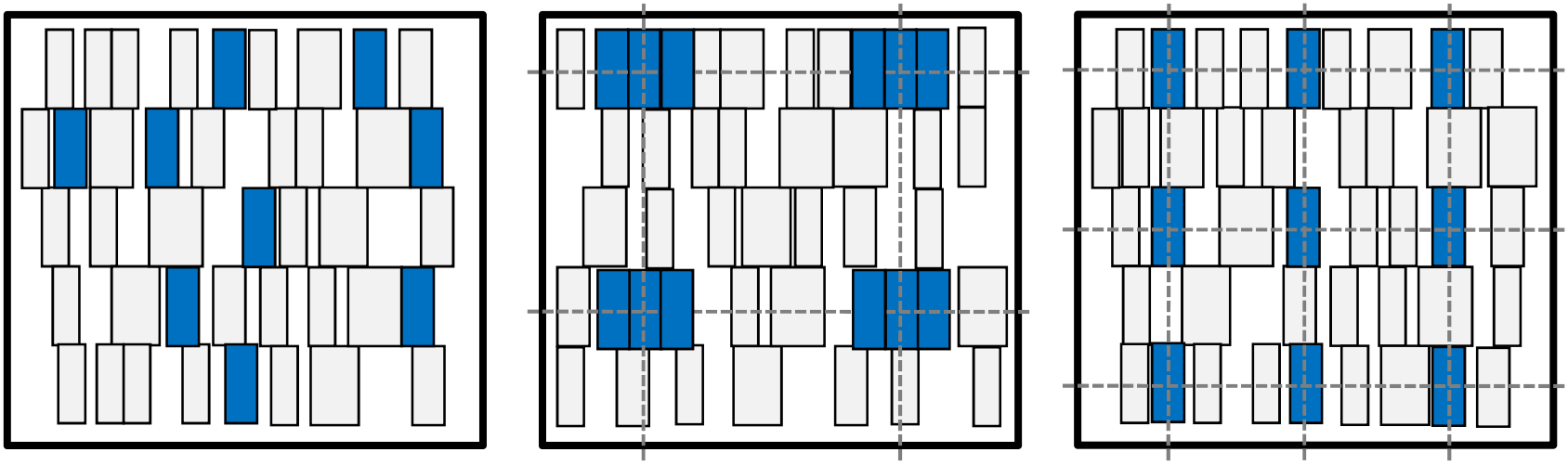
Cell types with higher success rates are more useful

SimSynth Analysis

- Signatures are partial truth tables
  - Allows cells to generate different functions
  - More input patterns → more accurate truth tables
    - Used when smaller function changes are expected
  - Fewer patterns allows more significant changes
- Measures heterogeneity of the circuit
  - Low success rate → signal heterogeneity is high
  - Generating useful signals requires more spare cells
    - Needs higher spare-cell density

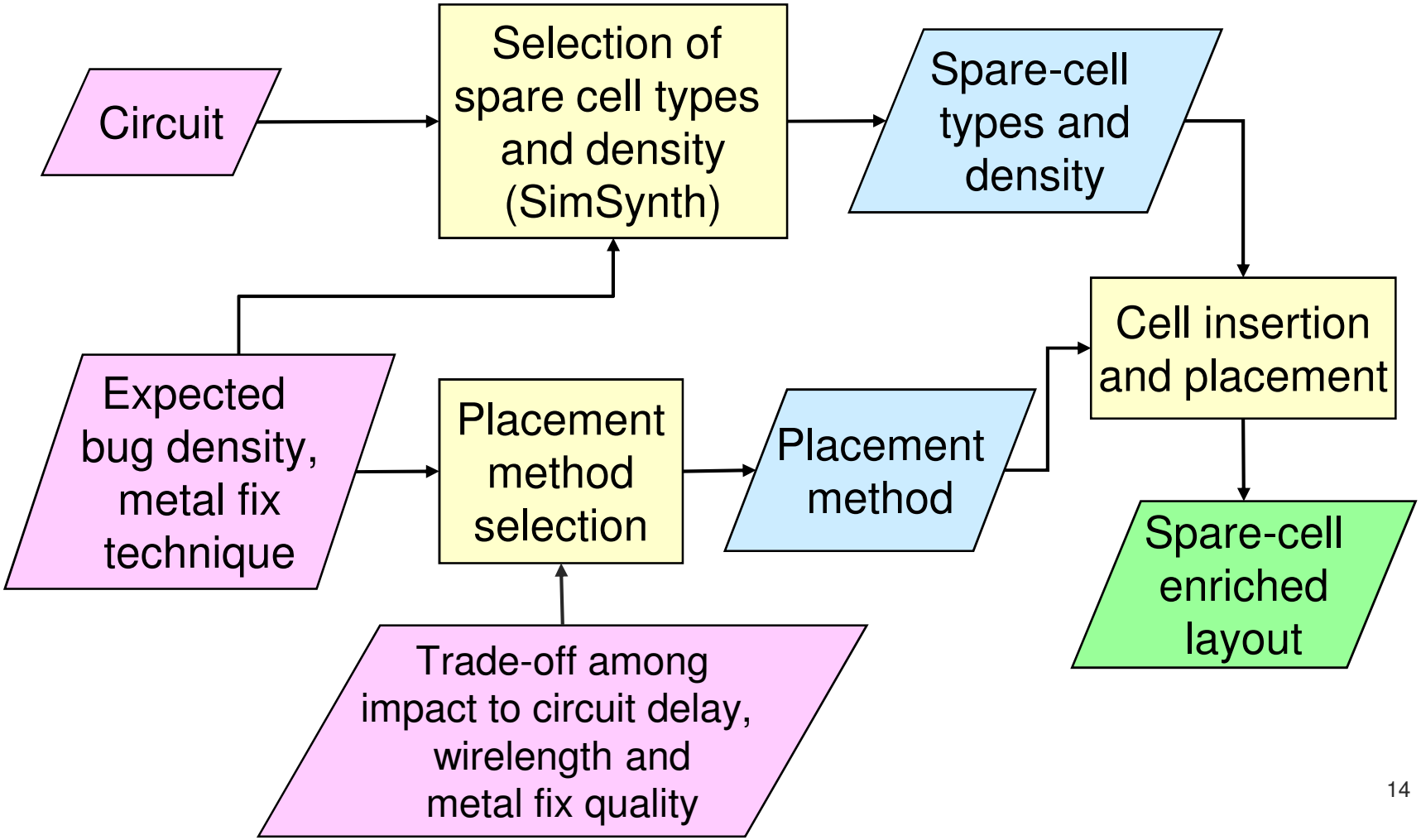
Spare-Cell Placement

- PostSpare
  - Spare cells scattered after placement
- ClusterSpare
  - Cell islands uniformly distributed before placement
- UniSpare (new)
  - Cells uniformly distributed before placement



PostSpare ClusterSpare UniSpare

Our Spare-Cell Insertion Methodology



Previous Work

Author, year	Spare cell type	Placement and routing methods	Drawbacks/limitations
Yee[20], 1997	Most commonly used cell in the design; **one of the earliest works on spare-cell insertion**	Spare cells scattered after placement	Designed for 2 metal layers only
Lee[11], 1997	**NAND/NOR gates with many inputs, BUF, INV, DFF (new spare-cell selection)**	**Placed close to potentially buggy region**	High-input gates may waste space; other cell types may be more useful
Payne[13], 1999	**Ga**		**nique claimed**
Wong[19], 2001	**Co tu**		**claimed**
Schadt[16], 2002	**Pr va**		**y; inputs/outputs evated**
Chaisemartin [5], 2003	**NA**		**eate routing con-**
Bingert[3], 2003	**Ga**		**ccupy too much**
Giles[9], 2003	**New spare-cell selection within cell islands including INV, DFF, MUX, AND, NAND, NOR and BUF**	**Placed according to design hierarchy**	Each module is allowed only one additional I/O; only fixed blocks supported
Or-Bach [12], 2004	**New FPGA-like structure**	N/A	Uses 3 metal layers only; no placement technique claimed
Vergnes[17], 2004	**New structure with functional input bus and an equation input bus**	**Placed with potentially buggy modules by hardwiring inputs of spare cells to signals in those modules**	Occupied routing tracks may create congestion
Brazell[4], 2006	N/A	**Whitespace allocated during Floorplanning; cells inserted after placement**	Spare cells occupy all remaining whitespace — impractical for modern layouts

Table 1: A summary of existing spare-cell insertion techniques described in US patents. Major contributions are marked in boldface.

No publications, only patents!
No empirical evaluations
(bug data are usually confidential)

For details please see the paper

Empirical Evaluation

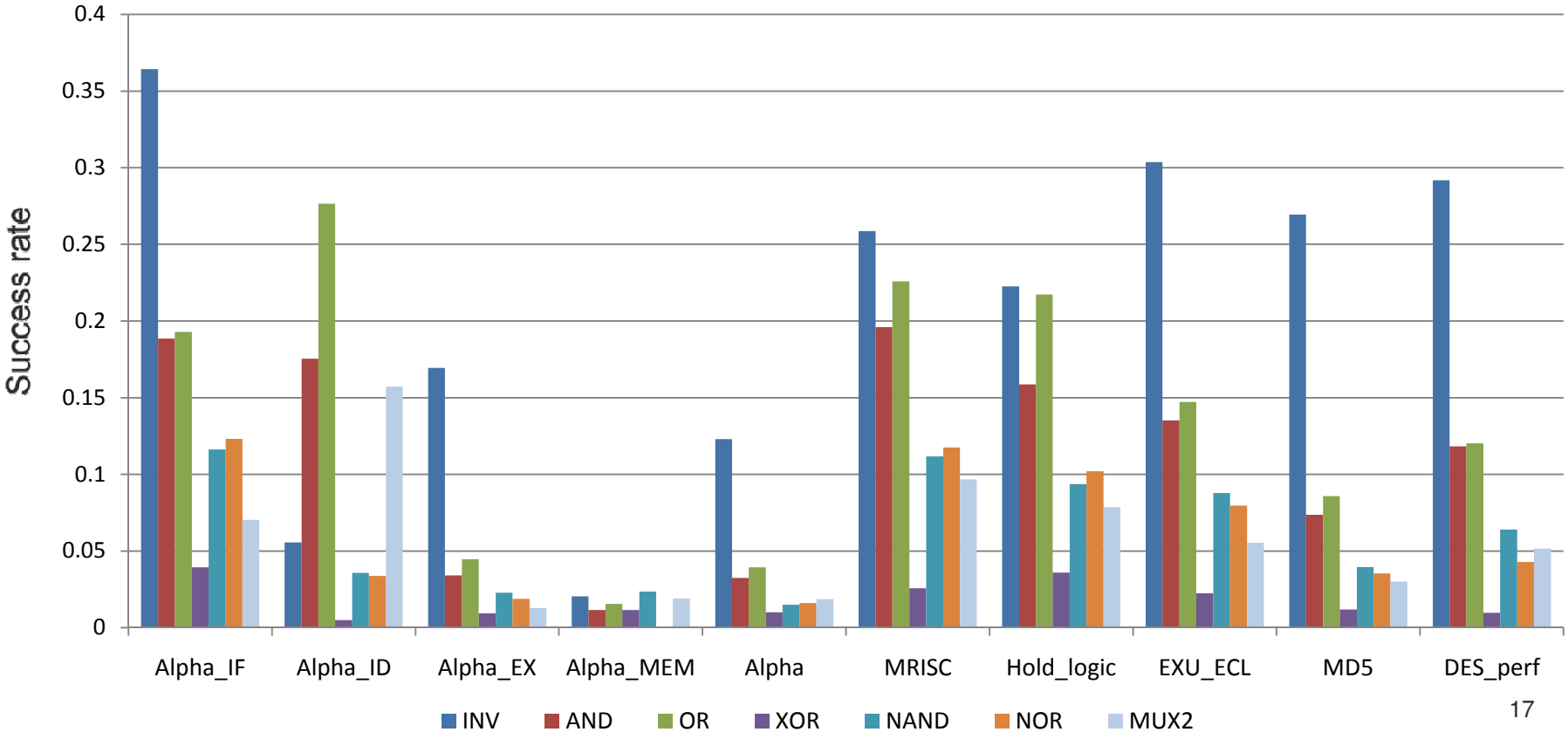
- Benchmarks

Benchmark	Description	Cell count
Alpha_IF	Instruction fetch unit of Alpha	1205
Alpha_ID	Instruction decode unit of Alpha	11806
Alpha_EX	Instruction execution unit of Alpha	20903
Alpha_MEM	Memory stage of Alpha	363
Alpha	Alpha CPU full chip	30212
MRISC	MiniRISC CPU	4359
Hold_logic	Part of picoJava IU control	67
EXE_ECL	Part of OpenSparc EXU control	2083
MD5	MD5 encryption/decryption core	9181
DES_perf	DES encryption/decryption core	100776

(Alpha is from *Bug UnderGround* project in Michigan)

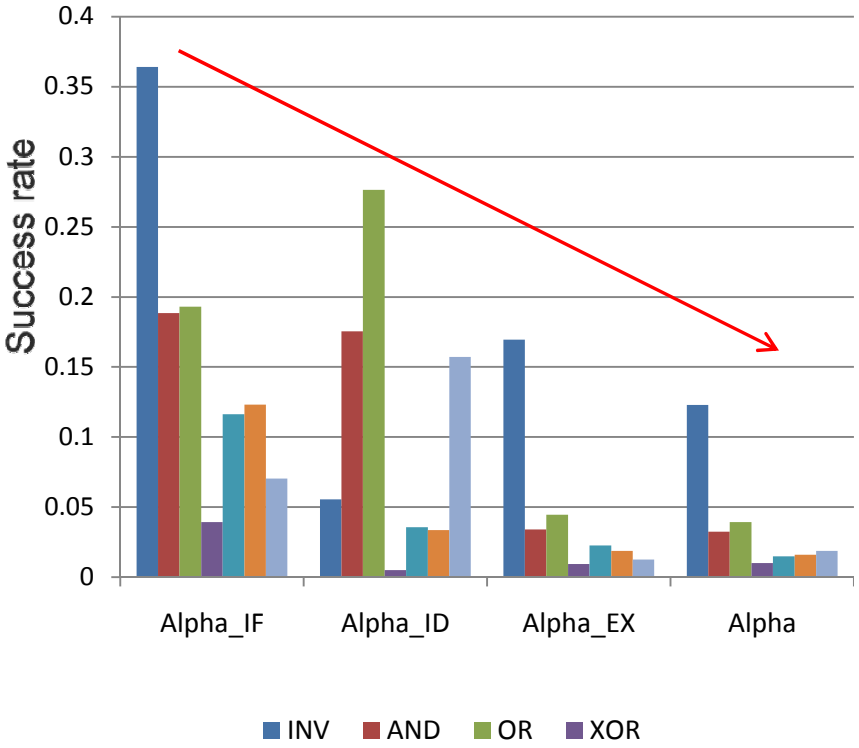
Cell-Type Selection

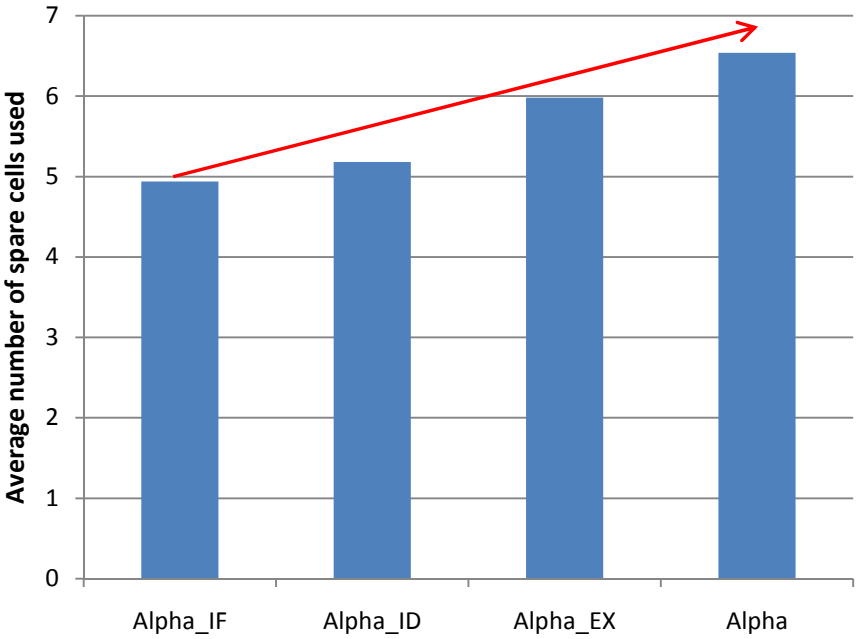
- Different circuit requires different types of cells
- INV, AND, OR, NAND, NOR are more useful



Spare-Cell Density

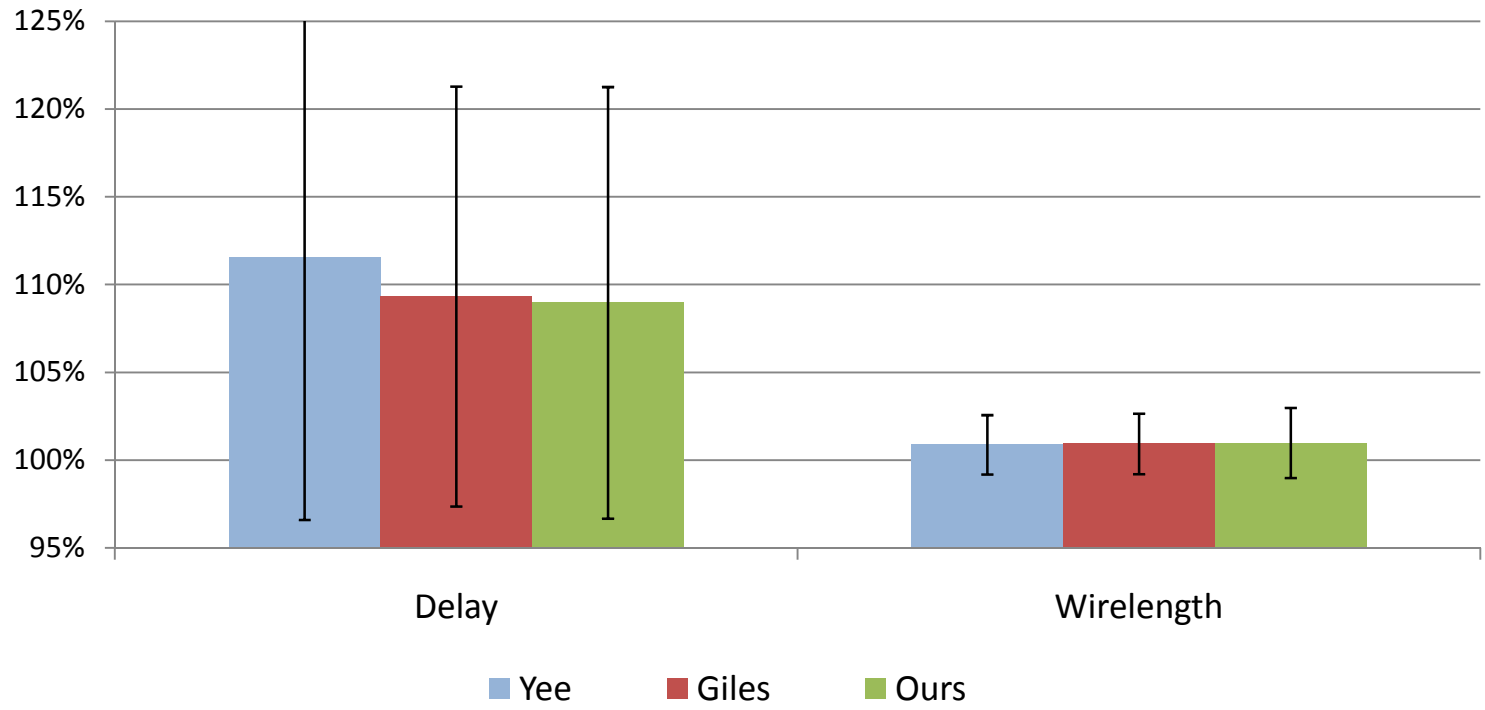
- Resynthesize subcircuits using spare cells
- Measure the number of cells used
- Lower success rate requires more spare cells





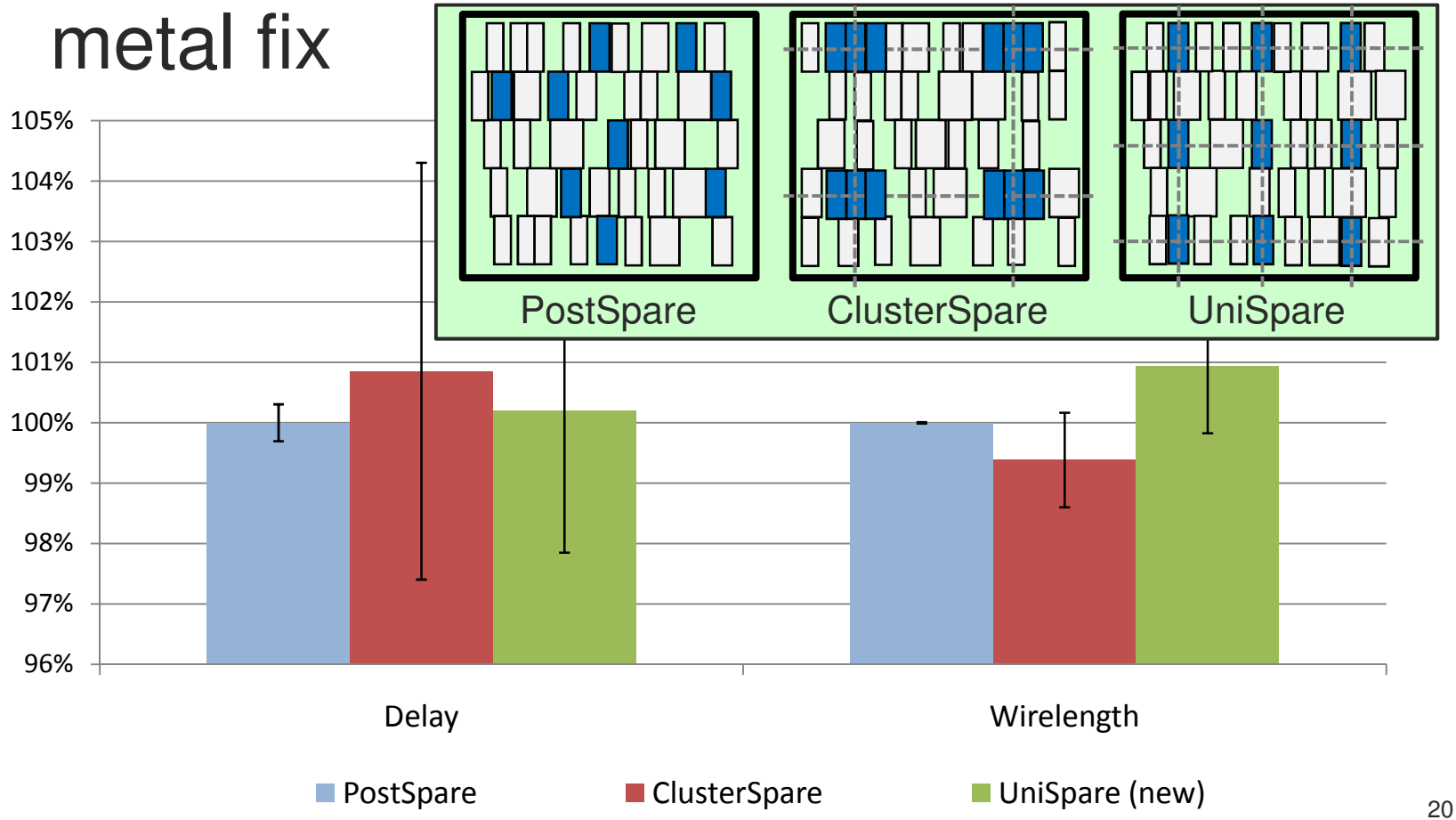
Cell-Type Selection

- Comparison to previous work
  - Ours has 23% and 4% smaller delay increase
  - Wirelength increase is approximately the same



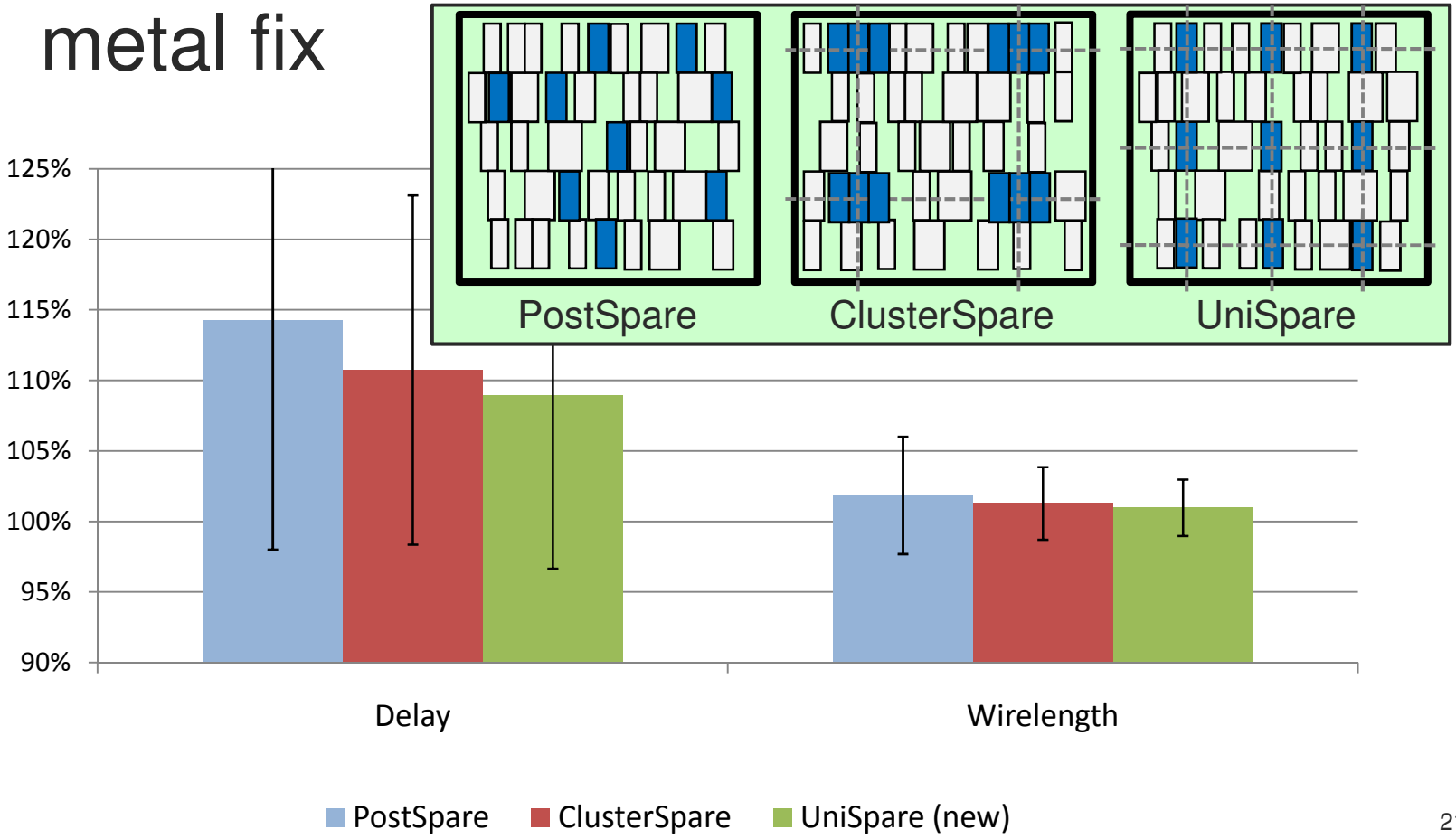
Spare-Cell Placement

- Impact on delay and wirelength before metal fix



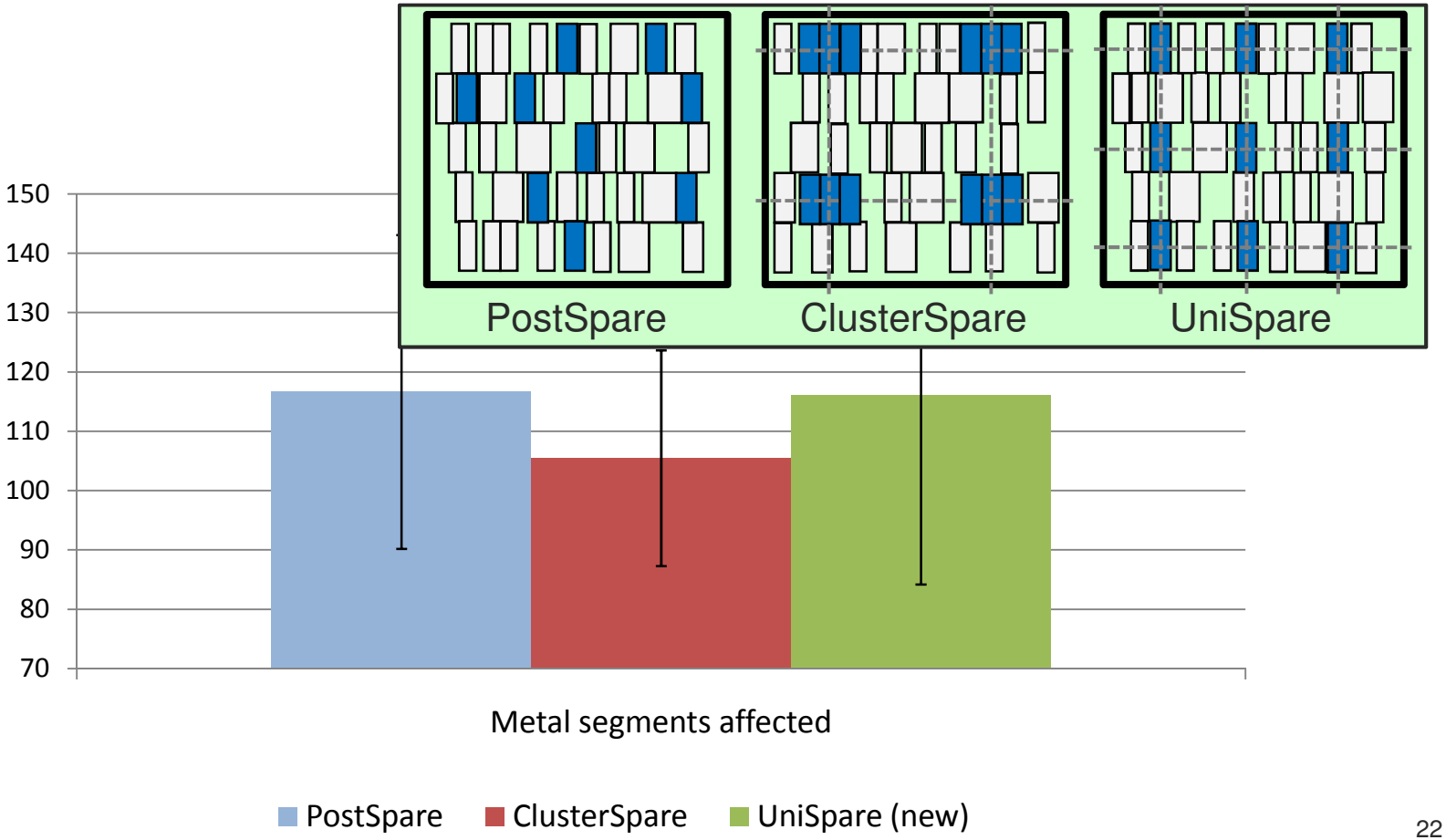
Spare-Cell Placement

- Impact on delay and wirelength after metal fix



Spare-Cell Placement

- Impact on number of metal segments affected

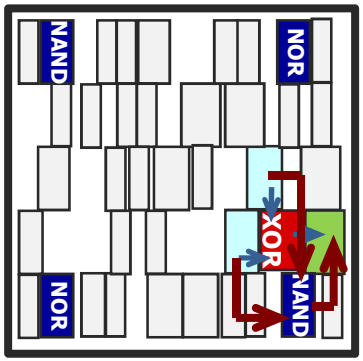


Summary

- Spare cell selection
  - Use SimSynth to determine cell types and density
- Spare cell placement
  - PostSpare
    - Minimal impact on circuit performance, worst metal fix quality
  - ClusterSpare
    - Minimal number of affected metal segments
    - Larger impact on circuit delay
  - UniSpare
    - Minimal delay increase
    - Balance between impact to the circuit and metal-fix quality

Insights and Contributions

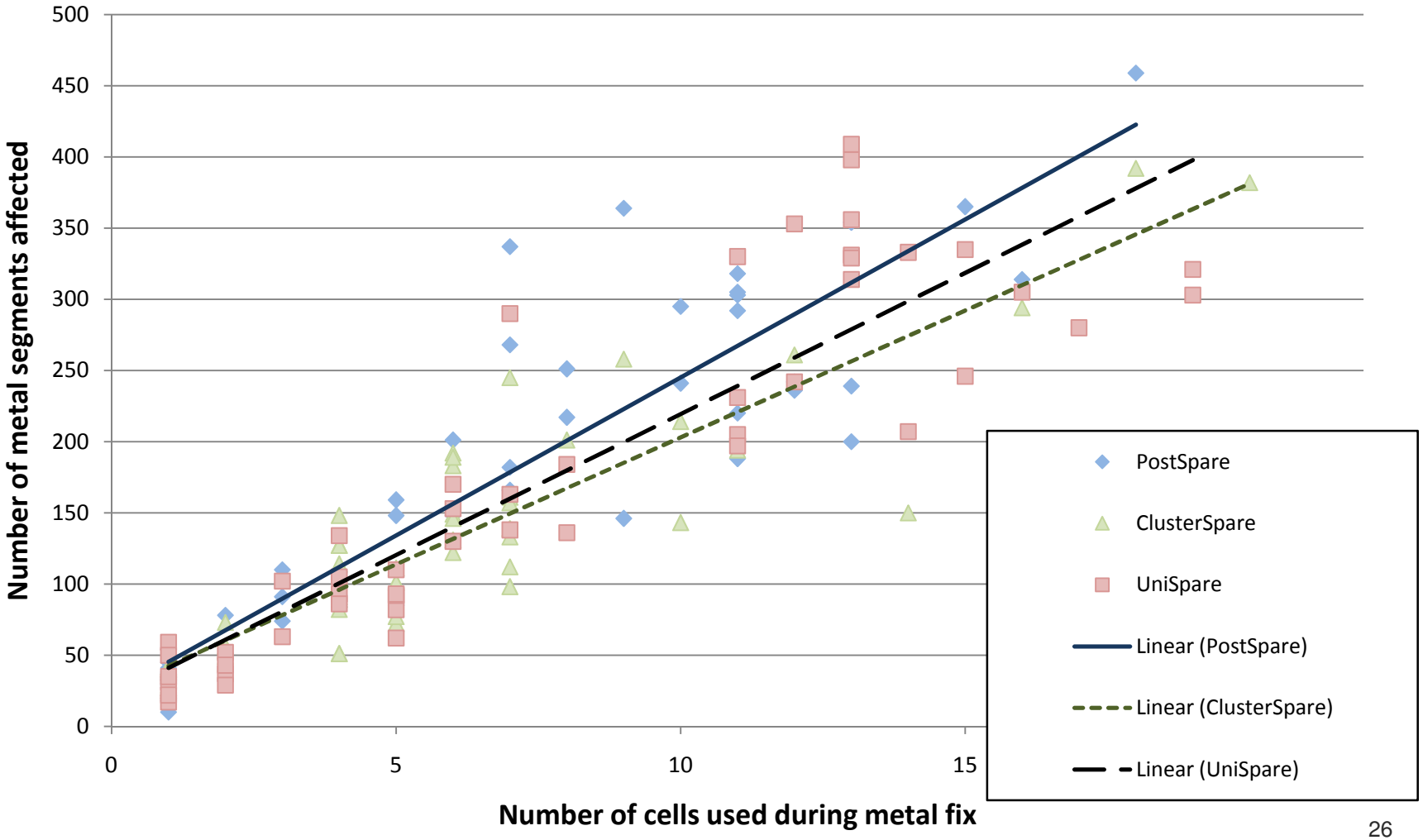
- Cell-type selection: **a logic synthesis problem**
  - A new technique – SimSynth
    - Selects different spare cells for different designs
    - Can also estimate required spare-cell density
- Cell placement: **a physical design problem**
  - Trade-off among delay/wirelength increase, affected metal segments and circuit performance
  - UniSpare provides the best balance between impact to the circuit and metal-fix quality



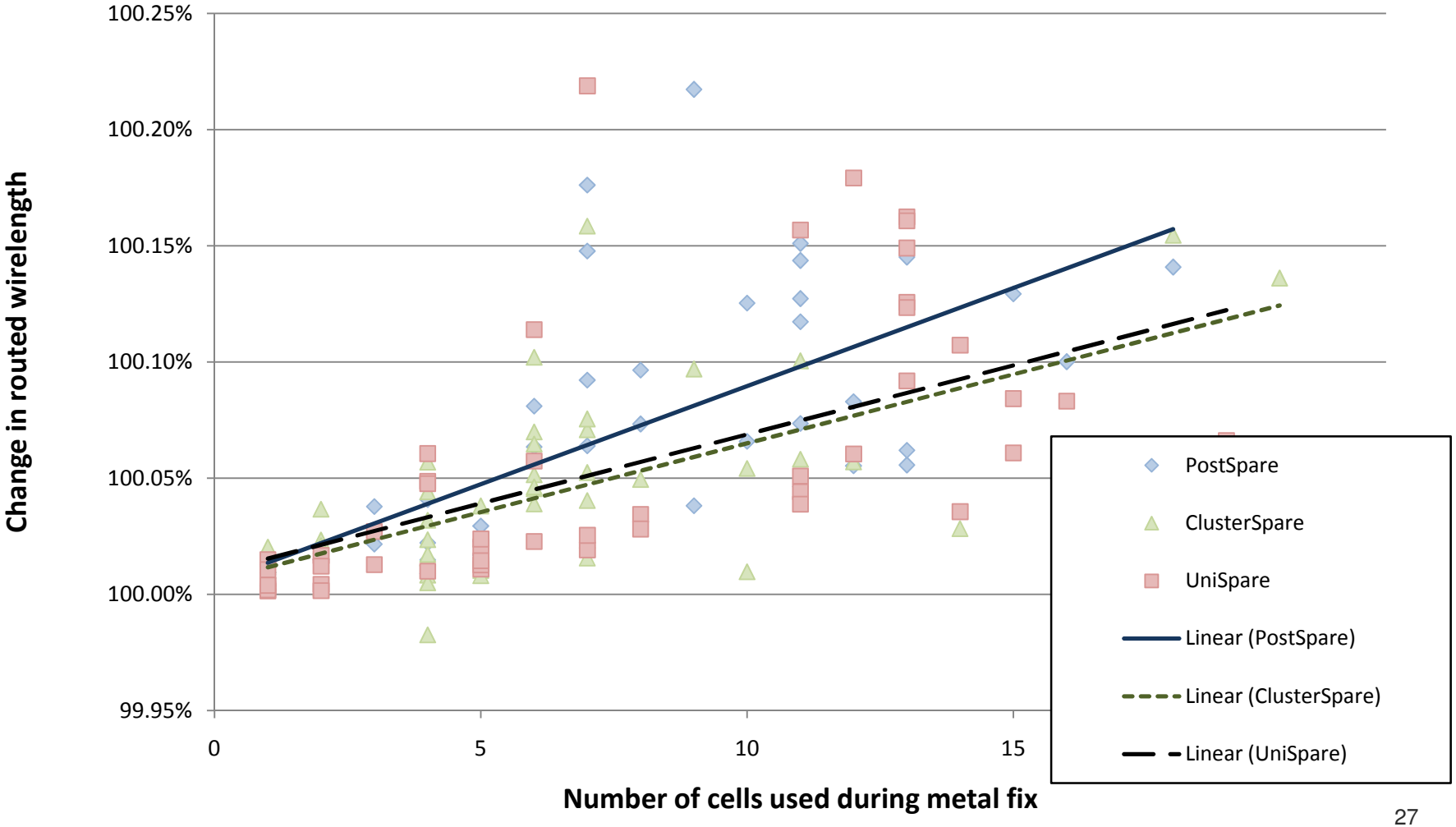
- **A new spare-cell selection & insertion methodology**
  - Considers both logical and physical information
- **First empirical analysis of spare-cell insertion**

Backup Slides

Spare-Cell Placement



Spare-Cell Placement



Previous Work

- Spare-cell insertion method depends on expected bug rate and nature of bugs
  - May vary from design to design
  - Confidential – most work published as patents
- Selection of spare-cell types
  - Most-commonly used cell [Yee'97]
  - Basic gates (NAND, NOR, INV…) [Lee'97,Giles'03]
  - Configurable logic [Payne'99, Wong'01, Schadt'02, Bingert'03, Or-Bach'04]
  - Complex structures [Chaisemartin'03, Vergnes'04]

Previous Work

- Insertion of spare cells
  - Scattered after design placement [Yee'97, Payne'99]
  - Scattered uniformly before design placement [Schadt'02]
  - Floorplaned with the design
    - Scattered uniformly before design placement [Bingert'03]
    - Scattered after design placement [Brazell'06]
  - Placed closer to potentially buggy region [Lee'97, Vergnes'04]
- Drawbacks
  - Lacks analytical and empirical evaluation
  - The same method is applied throughout the design
    - Cannot address different needs from different design regions