# Rescue: A Microarchitecture for Testability and Defect Tolerance

Ethan Schuchman and T. N. Vijaykumar

*School of Electrical and Computer Engineering*

*Purdue University*

## Defects Under Scaling

Hard defects not transient faults

Why defects increasing problem?

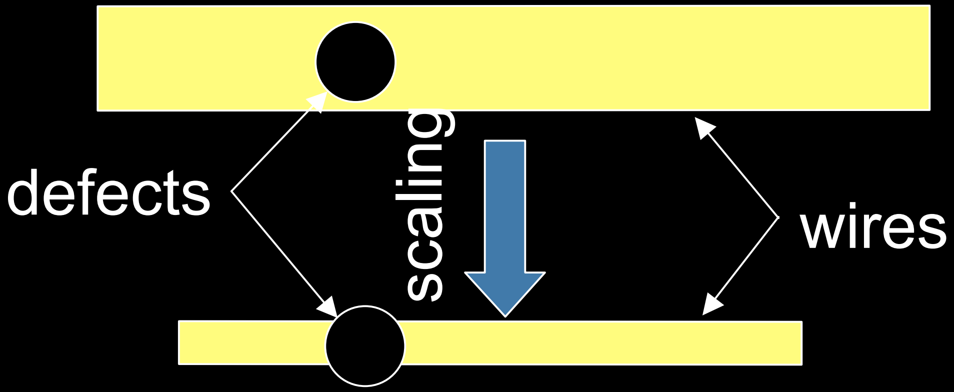
Scaling inherently increases defect susceptibility

Constant chip area

=> defects/chip ($\lambda$) increases => yield decreases as $e^{-\lambda}$

Solved by reducing impurity particles/wafer in fab

But scaling impurities may not be viable beyond 65nm [ITRS]



Defects only get worse

Worth exploring architectural solutions

## Defect Tolerance for Yield: Today

Defective components turned off at test time

Computation redirected to “like components” at run time

Prevalent in memory system:

- spare row, columns, subarrays

Beginning to appear at Core level in CMPs

- Called CPU sparing
- Disable defective core, use the rest

But increasing defect rate => lose many cores

We show finer granularity is feasible, results in better throughput

## Fine-grain Defect Tolerance

At microarchitecture level

- Rescues cores deemed defective by CPU sparing

Needs defect isolation to microarchitecture granularity

- Currently difficult and needs prohibitively more testing time

Once defective components known, disabling easy

- Architecture already accounts for busy resources

Defect avoidance easy

Defect isolation hard using realistic testing (time and hardware)

## Realistic Fine-grain Defect Isolation

Rules of the game:

- Should be based on standard testing methodology
  - Else hard to integrate into current testing process
- Isolation to microarchitectural blocks
  - Else can’t support fine-grained defect tolerance
- Isolation time close to (conventional) detection time
  - Else not economical
- Extra logic kept minimal
  - Else degrades yield

## Contributions

First paper to integrate testing and architecture

Change architecture, *not* testing, for fast isolation

Identify *Intra-Cycle Logic Independence* (ICI)

- Sufficient condition for fast isolation

Propose ICI-compliant microarchitecture called Rescue

Show Rescue suffers only small IPC reduction

- – compared to non defect-tolerant CPU

Evaluate average throughput adjusting for yield

Show Rescue improves over CPU sparing alone

## Overview

# Detecting Defects with Scan

Most prevalent testing technique today

Latches replaced with scan equivalents

Connected into shift register

State shifted in and out through scan chain

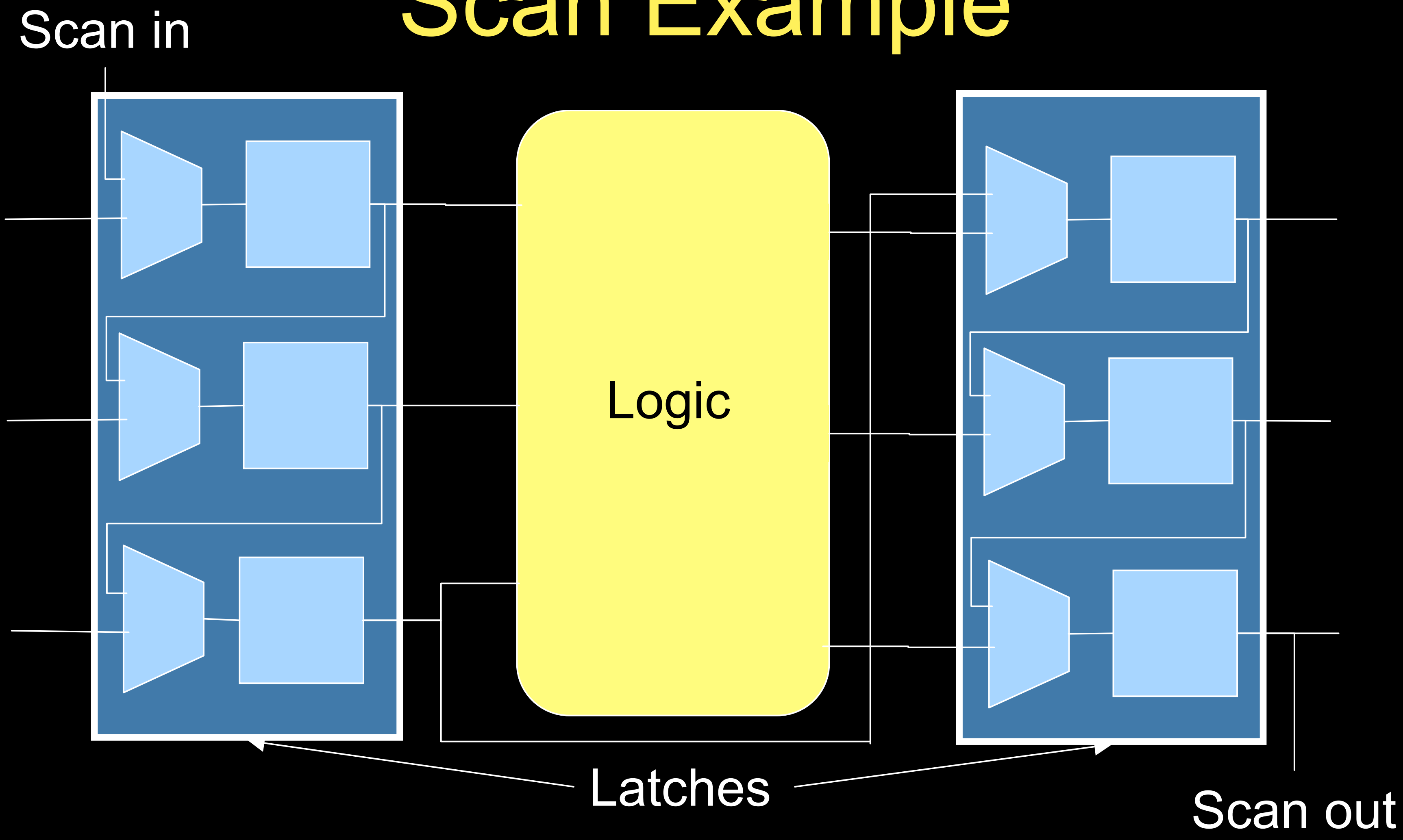
Requires only 3 pins: SI, SO, SE

Automated Test Pattern Generation (ATPG)

- Creates test input and output vectors
- Each vector tests many defects in parallel

Efficient in time and extra hardware

## Scan Example

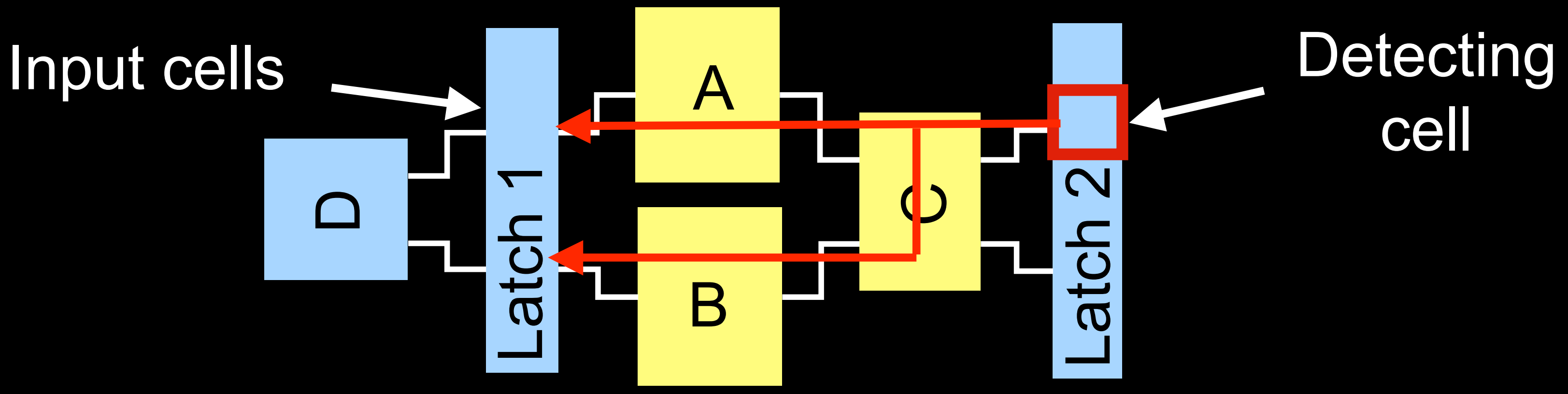


Scan in: 100101    Scan out: 011001    Expected: 010001

# Defect Isolation with Scan

Isolate defects to path between input and output scan cells

- Each scan cell maps to a set of these paths
  - Each path can pass through one or more components
  - Defect must be along these paths



Defect may be in block A, B, or C but not D

Not good enough to isolate to only A or only B

In a pipeline scan isolates defects to paths WITHIN 1 CYCLE

# Overview

Introduction

Background: Scan chains

Intra-Cycle Logic Independence

Rescue

Methodology and Results

Conclusions

## Intra-Cycle Logic Independence (ICI)

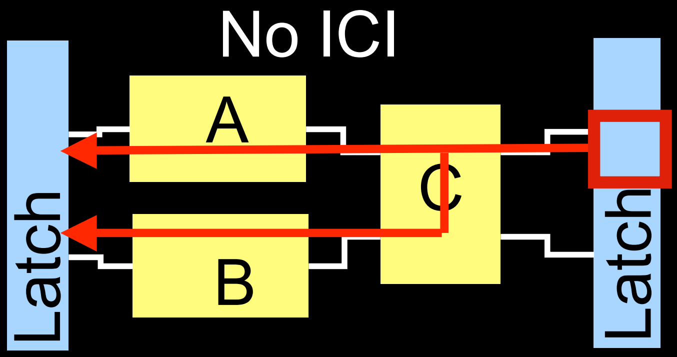
To isolate defect to components:

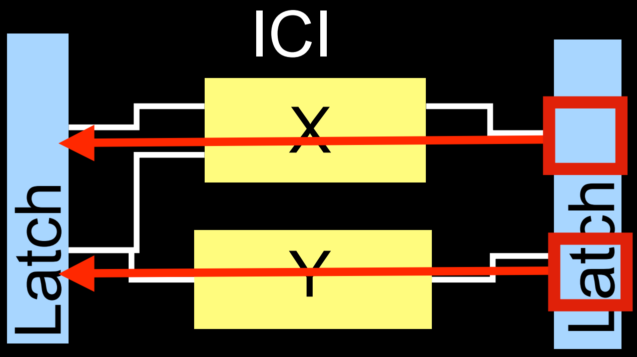
Each scan bit maps to one path with only one component

=> Each component independent of others for one cycle

We call this condition ICI

ICI is sufficient condition for fast isolation to one component



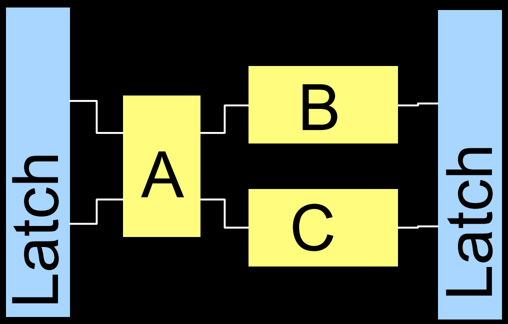


All superscalar stages do not have ICI

We propose transformations to enforce ICI

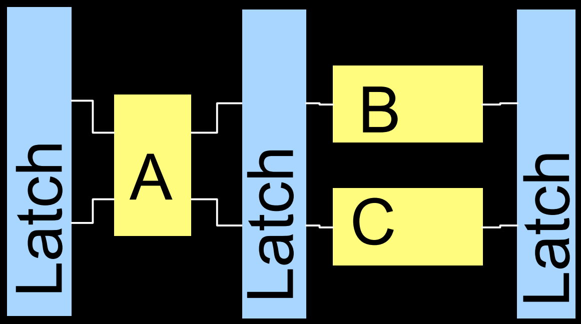
# ICI Transformations

No ICI



B & C depend on A
within 1 cycle

1) Cycle Splitting

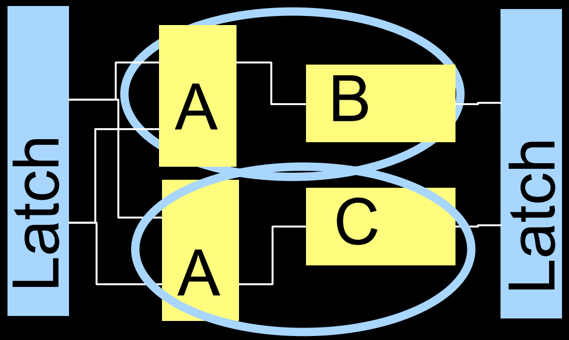


No dependence
within 1 cycle

- Pipeline deeper

+ Little area
overhead

2) Logic Privatization



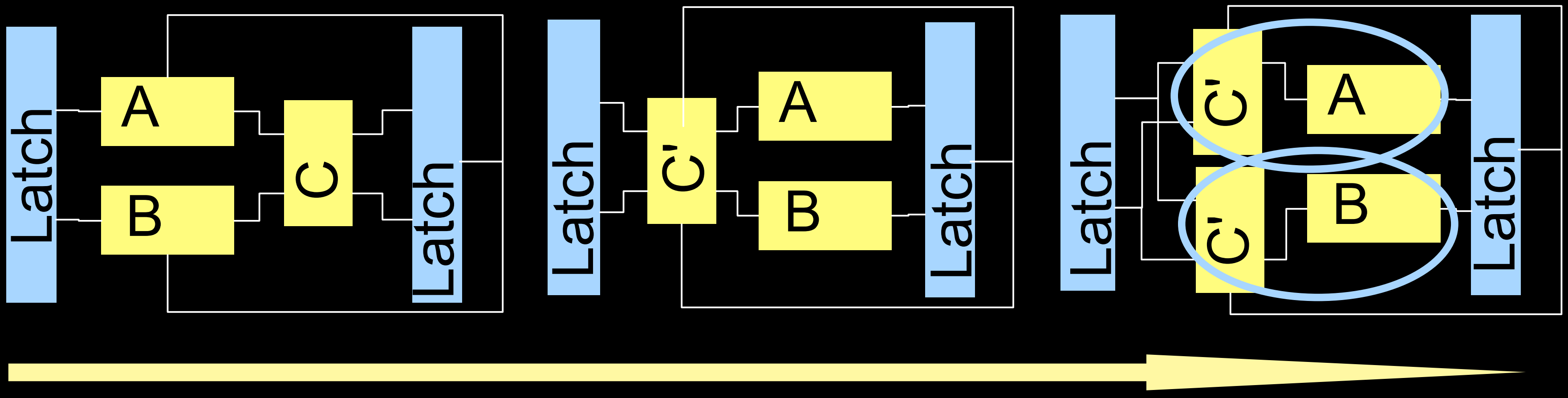
super component ICI

No dependence
within 1 cycle

- Increased area

+ No performance
penalty

## ICI Transformations (cntd.)



Dependent within 1cycle

Dependent within 1cycle

Independent within 1cycle

3) Dependence Rotation

For pipeline stages with loops (i.e. issue)

Allows logic privatization

Does not increase loop size

Choose transformation based on structure & penalty

## Overview

Introduction

Background: Scan chains

Intra-Cycle Logic Independence

Rescue

Methodology and Results

Conclusions

# Rescue Microarchitecture

Paper describes necessary ICI modifications to each stage
Also details of how defective components avoided at run time

Now describe enforcing ICI only in Issue due to time limitations
Warning: intricate issue implementation details coming

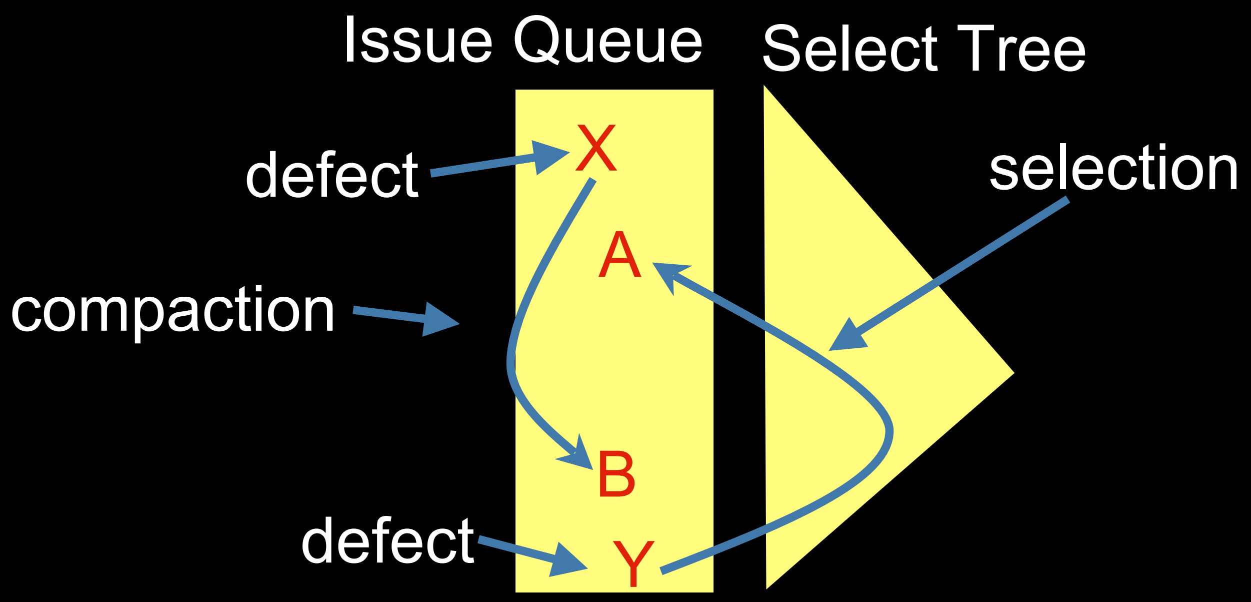
## Baseline Issue Stage

- Selection
  - chooses ready instructions for each ALU
- Broadcast
  - Notifies waiting instruction when others issue
  - Issued instructions wakeup dependents
- Compaction
  - Moves instructions toward head to maintain priority
  - Fills empty/issued entries

# Baseline Issue Stage (cntd.)

Entry affects others through selection and compaction

Defective behavior spreads within one cycle



Defect X corrupts entry B through compaction

Defect Y corrupts entry A through selection

Hard to attribute corruption to a specific (possibly distant) defect

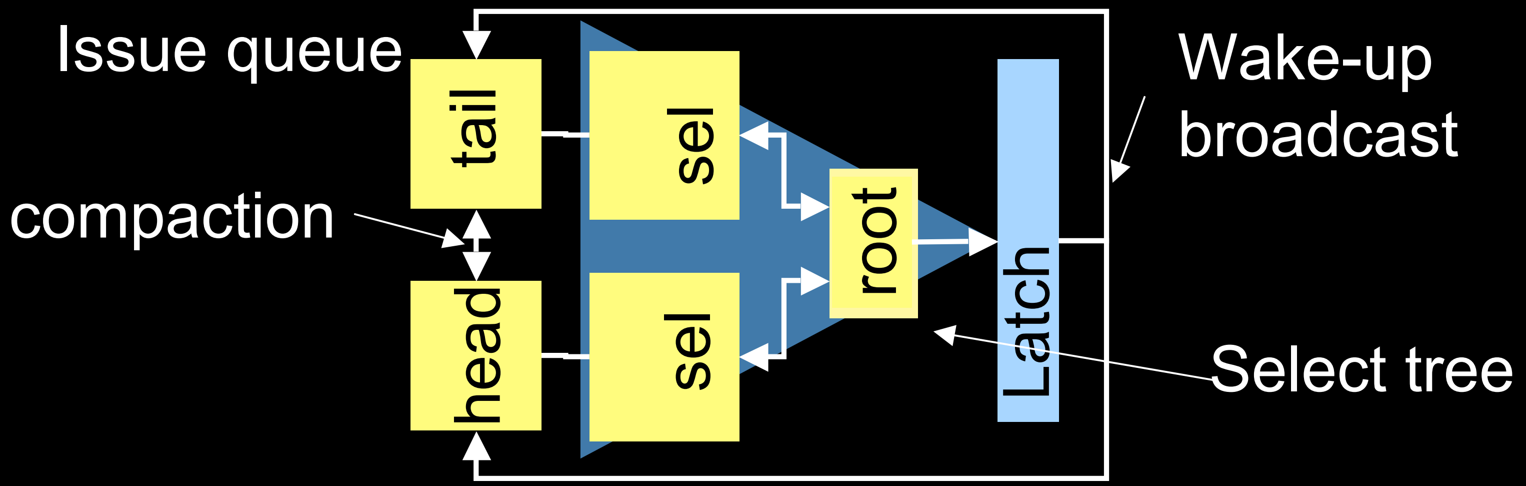
# Baseline Issue Stage (cntd)

Too much dependence to create ICI for every entry

Instead enforce ICI between two segments

So need to worry only about inter-segment ICI violations

- Compaction of instructions from tail to head
- Compaction counts from head to tail
- Selection from both segments



Enforce ICI at architecturally convenient granularity

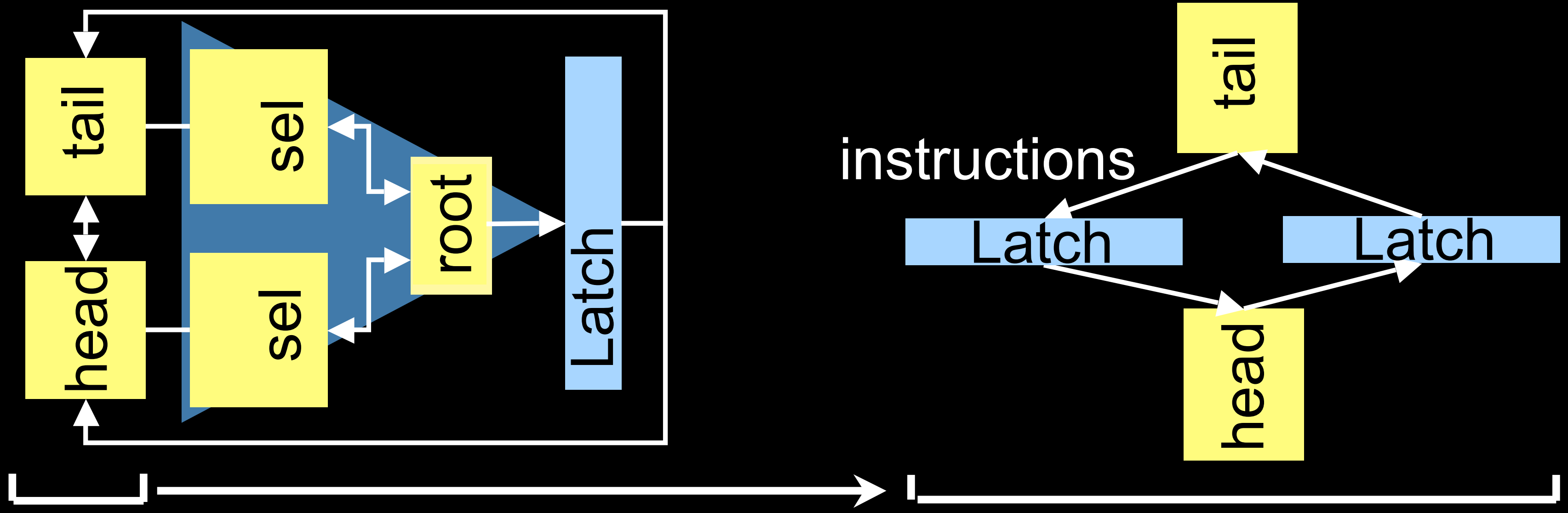
## Enforcing ICI in Rescue’s Compaction

Use cycle splitting

Compacted instructions & signals saved in temporary latch

Segments read only from latch => no dependence in 1 cycle

Free entry unavailable to insert instruction for 1 more cycle



Cycle splitting has small performance penalty here

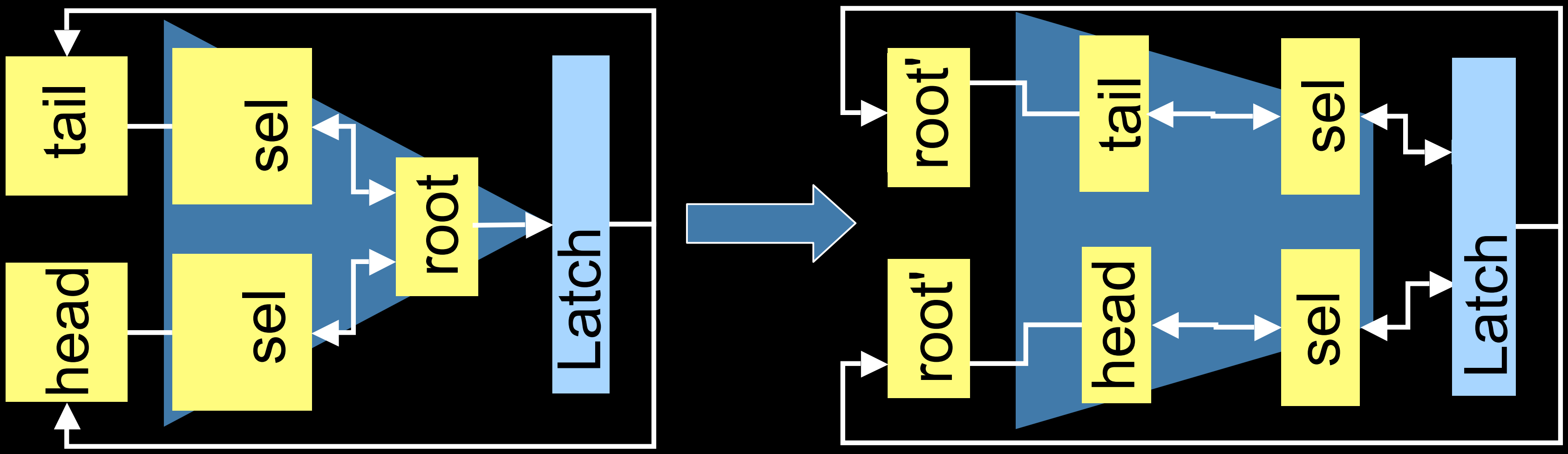
# Enforcing ICI in Rescue's Selection

Root violates ICI by communicating with both segments

Ensures instructions selected from both segments < width

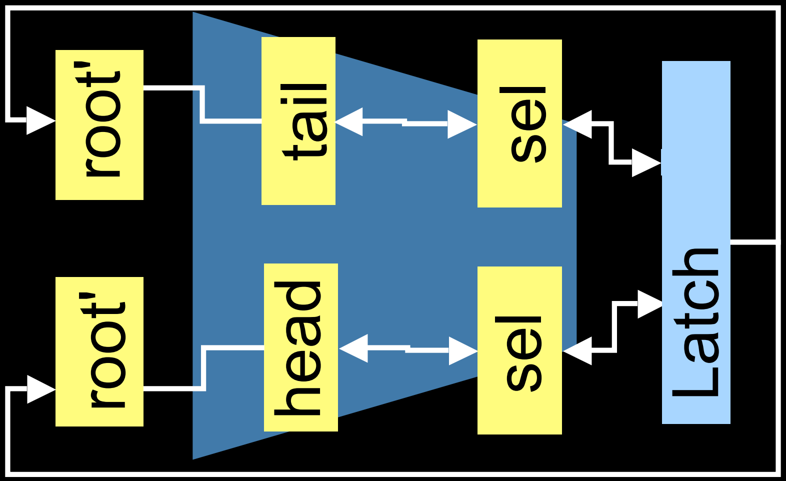
Use dependence rotation and logic privatization to enforce ICI

Function of root now done at the beginning of cycle



Segments are independent within 1 cycle

## Enforcing ICI in Rescue's Selection (cntd.)



Two segments are independent within the cycle

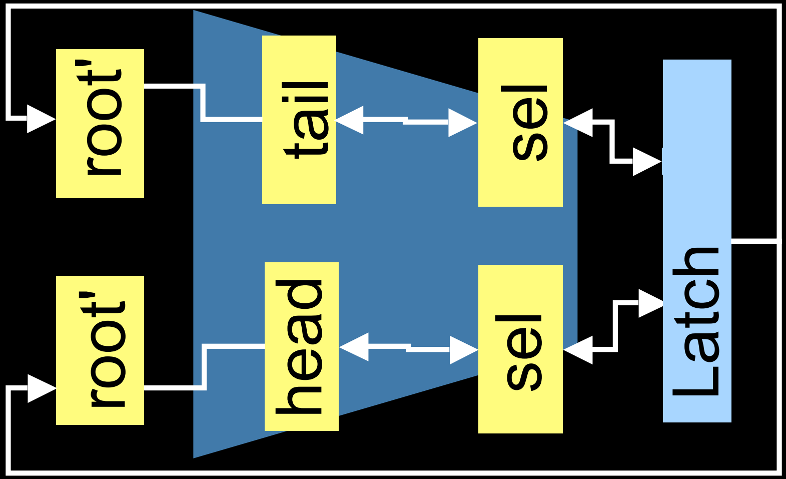
Each segment doesn't know how many selected in the other

Each segment obeys width constraint in selecting

But the sum between the two segments may exceed width

May select too many instructions

## Fixing Excess Select in Rescue



At beginning of cycle, roots check all broadcasts

If too many instructions issued:

- replay instructions from  one segment,
- block broadcast from that segment

Non-replayed instructions guaranteed to obey width

No penalty in common case of no excess

## Defect Isolated but How to Map it Out

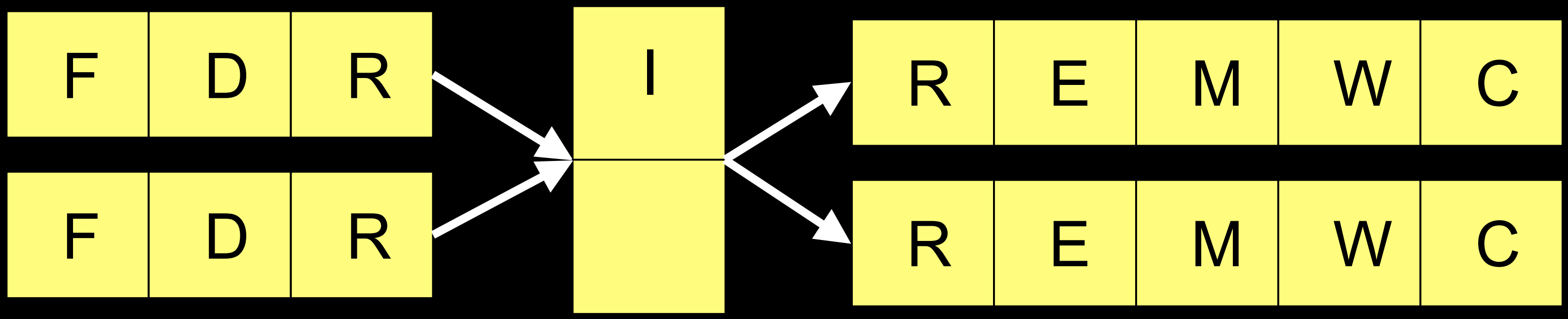
If one renamer in 2-way pipe defective:

- no point in trying to use 2 ways of decode

So we map out the entire frontend way (same for backend)

For issue and load/store queue map out the defective segment

2-Way Pipeline



Extremely fine-grain map out not needed => avoid overhead in map out

## Overview

## Methodology

- Evaluate Testability of Rescue
  - Create Verilog model and map to gate level
  - Simulate stuck-at faults
- Determine Yield Adjusted Throughput
  - Performance Simulations
  - Yield Calculations

| Performance | |
|---|---|
| Issue width | 4 |
| IQs, LSQ | 36, 36 |
| Int ALU, mult/div | 4, 2 |
| FP ALU, mult/div | 2, 2 |
| L1s, L2 | 64k, 2M |

| Area | |
|---|---|
| Total Base | 96 $mm^2$ |
| Total Rescue | 107 $mm^2$ |
| Front end | 12% |
| Int IQ | 4% |
| FP IQ | 4% |
| Int backend | 15% |
| FP backend | 21% |
| LSQ | 4% |
| Chipkill | 40% |

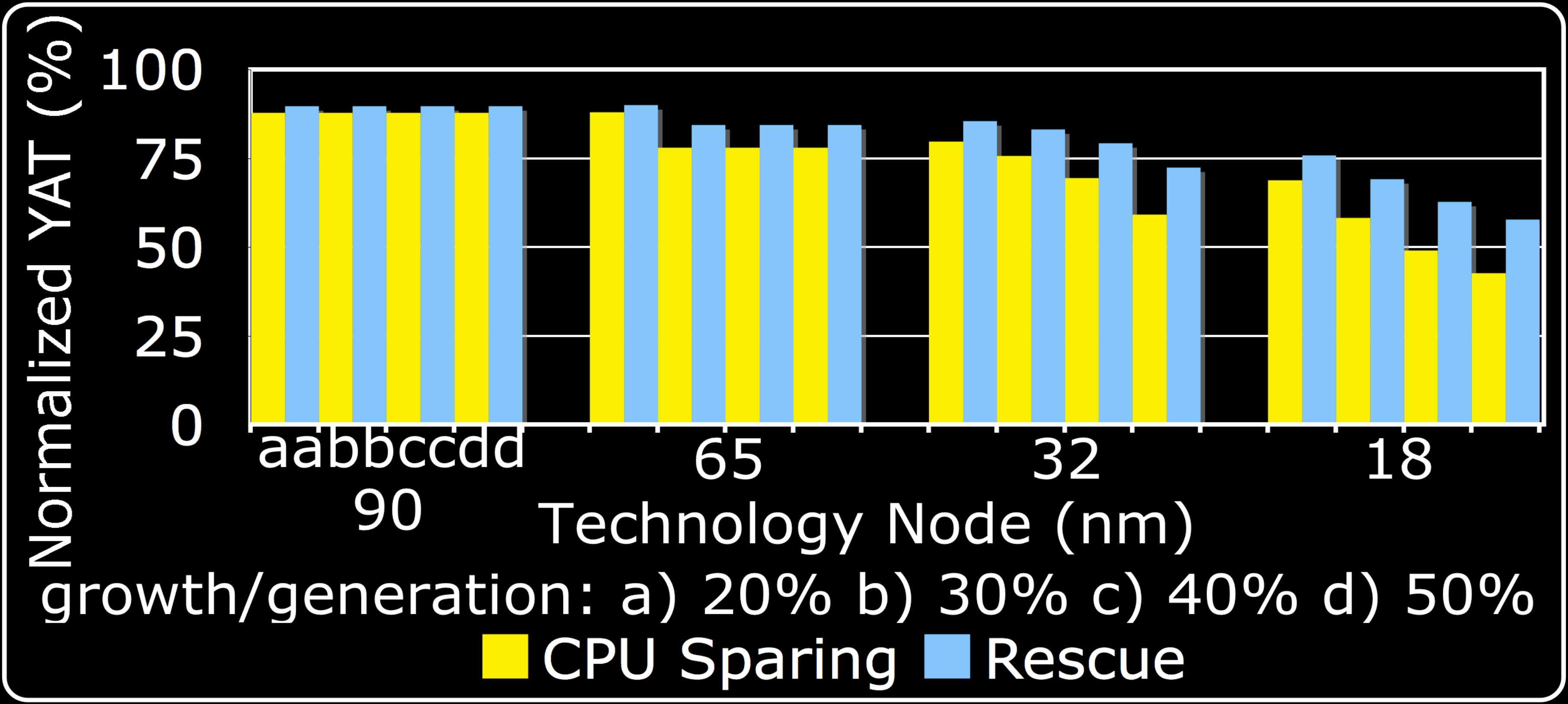
# Results

Isolate faults only 13% longer than detection in conventional

- Vs. orders of magnitude for diagnosis in conventional

Rescue 4% lower average IPC compared to no defect tolerance

Up to 40% YAT improvement at 18nm node



## Overview

## Conclusions

First paper to integrate testing and architecture
Showed fast isolation through architecture, *not testing*
Identified *Intra-Cycle Logic Independence* (ICI)
    Sufficient condition for fast isolation
Created ICI compliant microarchitecture called Rescue
Showed ICI modifications cause only small IPC reduction
Evaluated average throughput adjusting for yield
Showed Rescue improves over CPU sparing alone

Rescue important for future technologies