

Efficient Soft Error Protection for Microprocessors

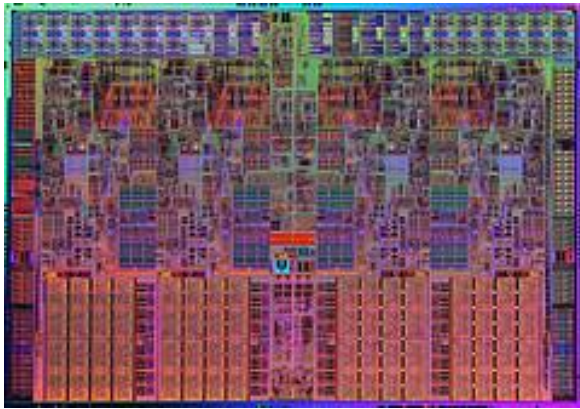
Sudhanva Gurumurthi

Students: Angshuman Parashar, Kristen Walcott, Blake Sutton,
Taniya Siddiqua

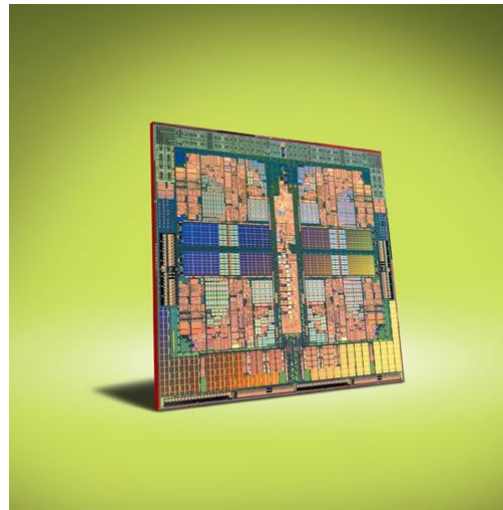
University of Virginia

E-mail: gurumurthi@cs.virginia.edu

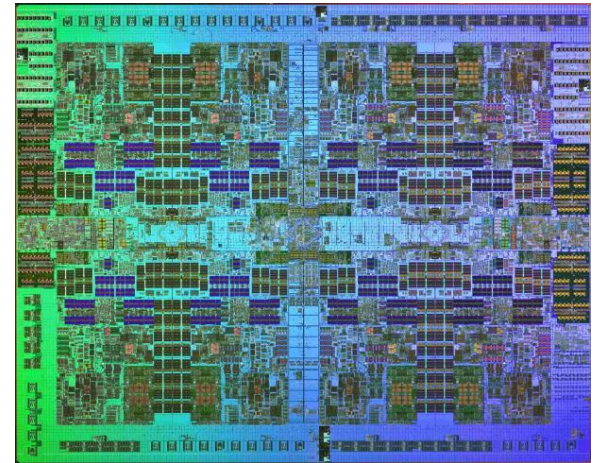
The Multicore Era



Intel Nehalem™



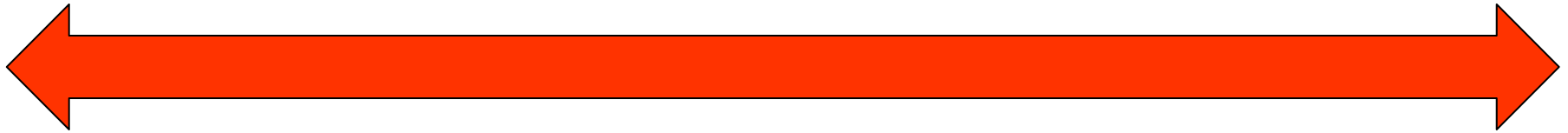
AMD Phenom™



IBM POWER7™

Future processors are expected to have more cores





RELIABLE HARDWARE

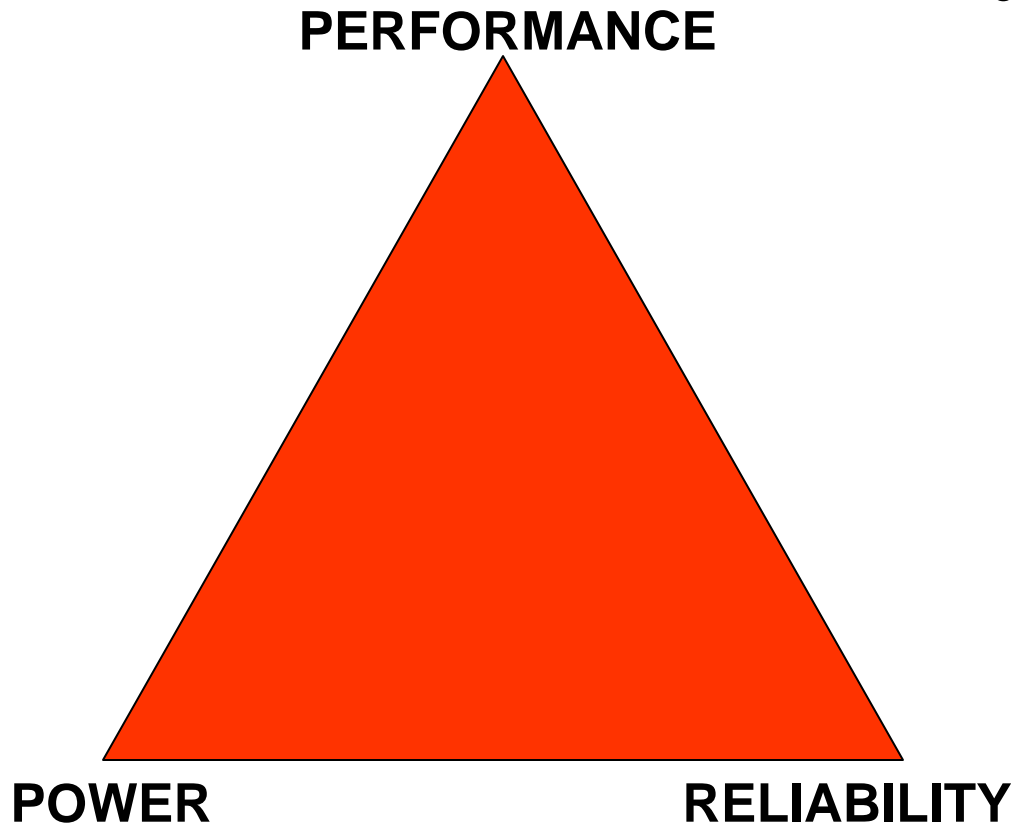
The Silicon Reliability Challenge

- **Soft Errors**
 - Random bit flips
 - Do not permanently damage the circuit
 - Latches and logic becoming more vulnerable
- **Hard Errors**
 - Affect the lifetime of the circuit
 - Can appear during fabrication or in the field
 - Examples: NBTI, Electromigration
- **Process Variation**
 - Affect circuit delay characteristics

Error Protection Techniques

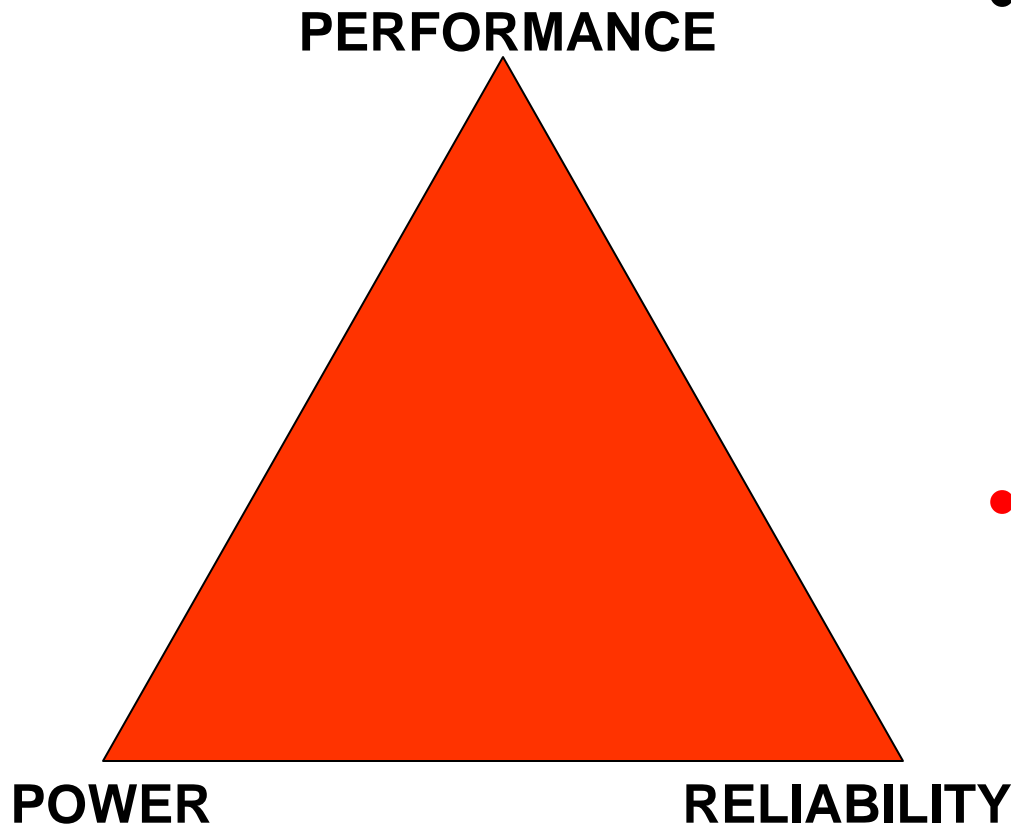
- **Redundancy**
 - Informational
 - Parity, ECC
 - Spatial
 - Executing instructions in different hardware units
 - Spare structures
 - Temporal
 - Executing instructions multiple times in the same hardware unit
- **Dynamic Reliability Management**
 - Dynamic voltage/frequency scaling

Error Protection Tradeoffs



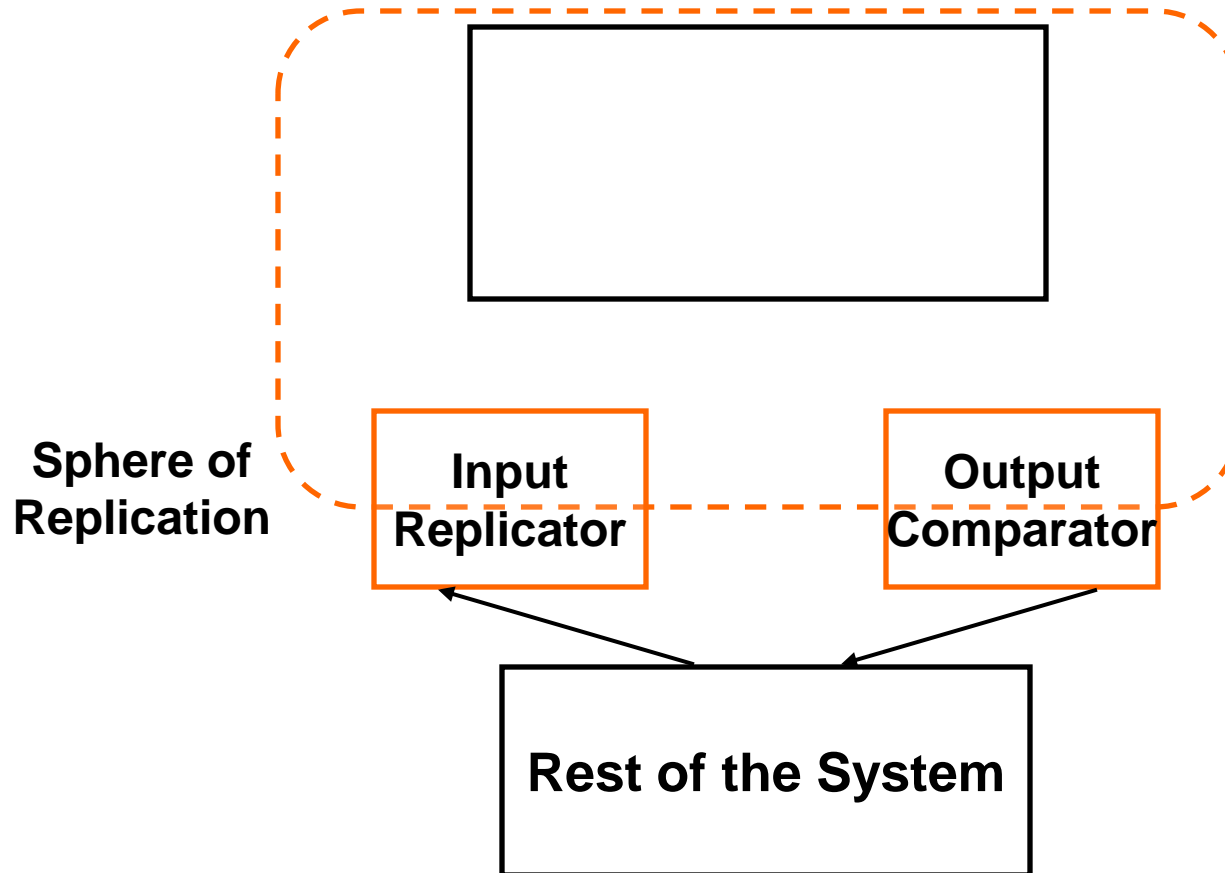
- Error protection entails overheads:
 - Performance degradation
 - Increased power consumption
 - Higher area

Error Protection Tradeoffs

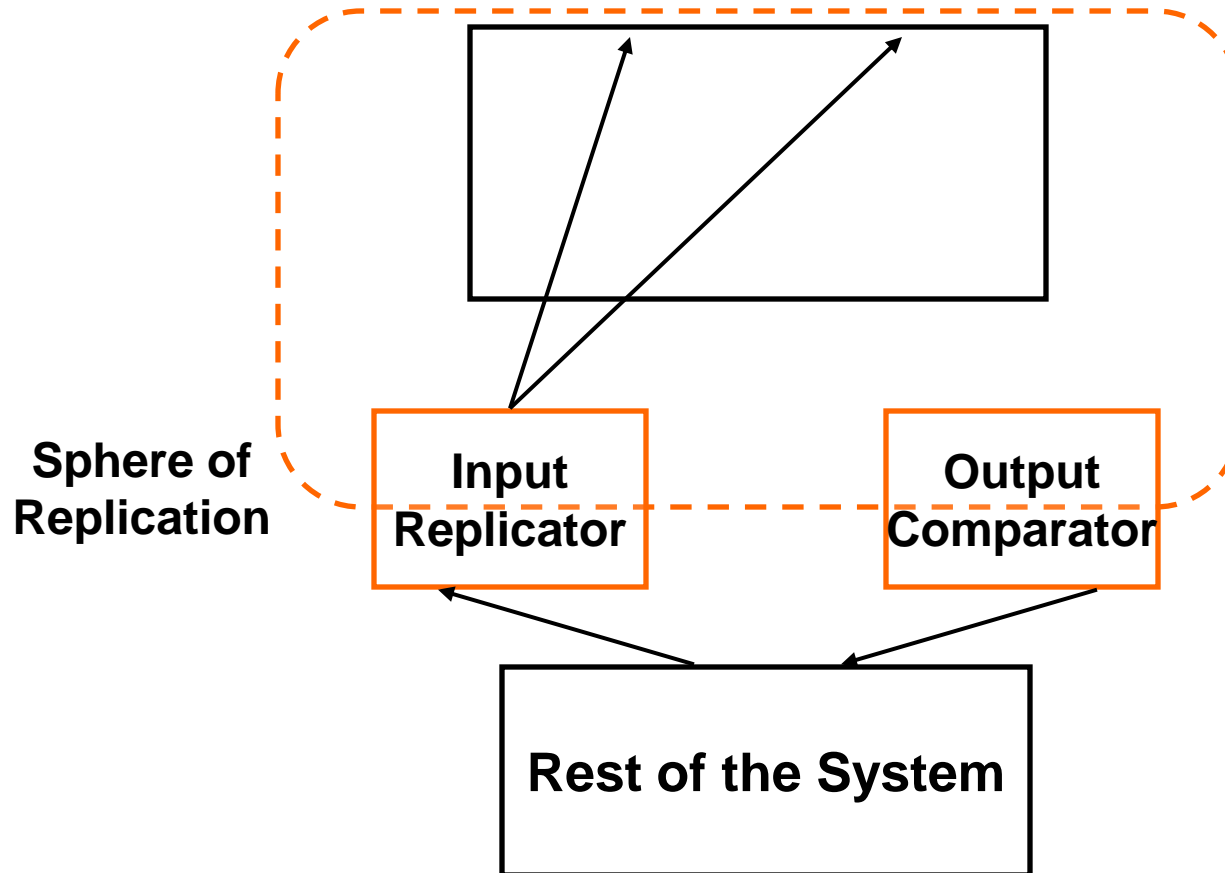


- Error protection entails overheads:
 - Performance degradation
 - Increased power consumption
 - Higher area
- **Goal:** Reduce the overheads while providing the **required** level of error protection

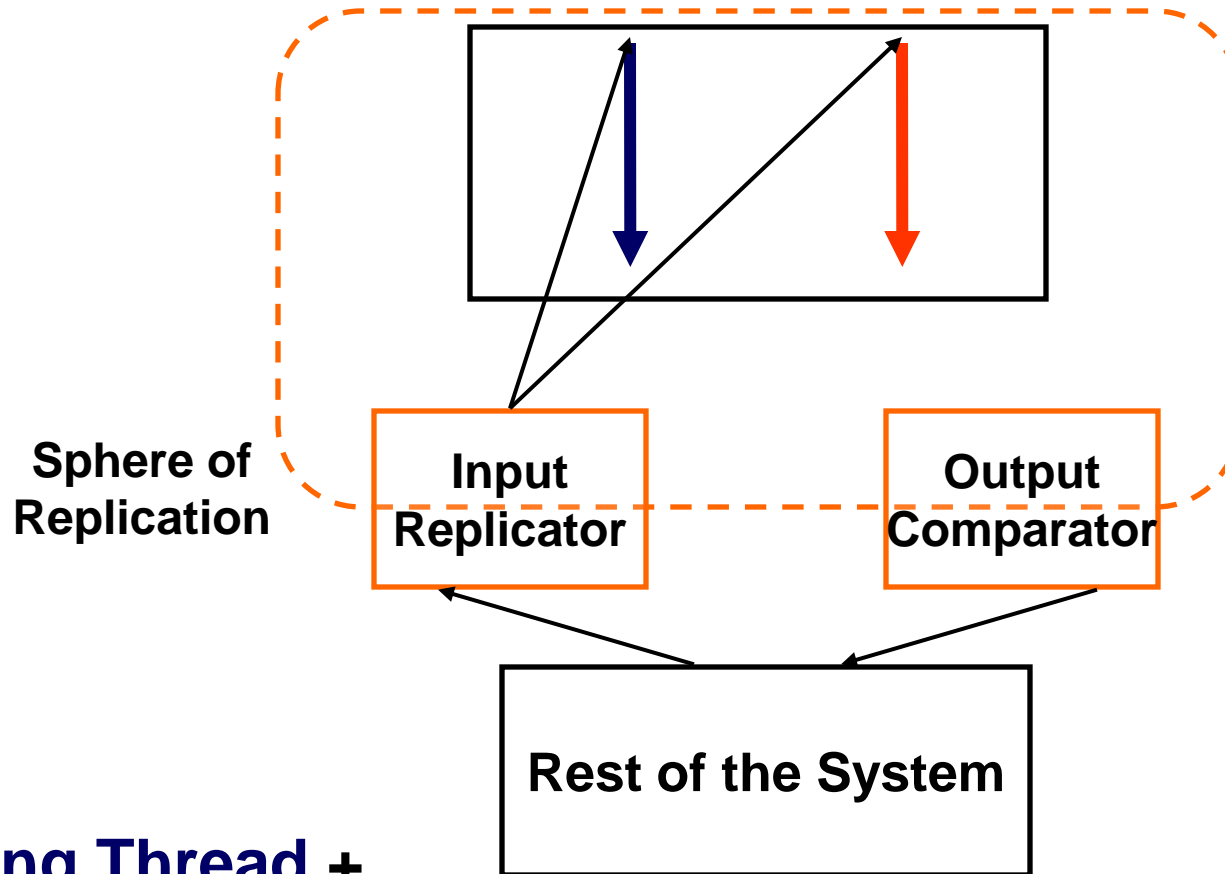
Redundant Multi-Threading (RMT)



Redundant Multi-Threading (RMT)

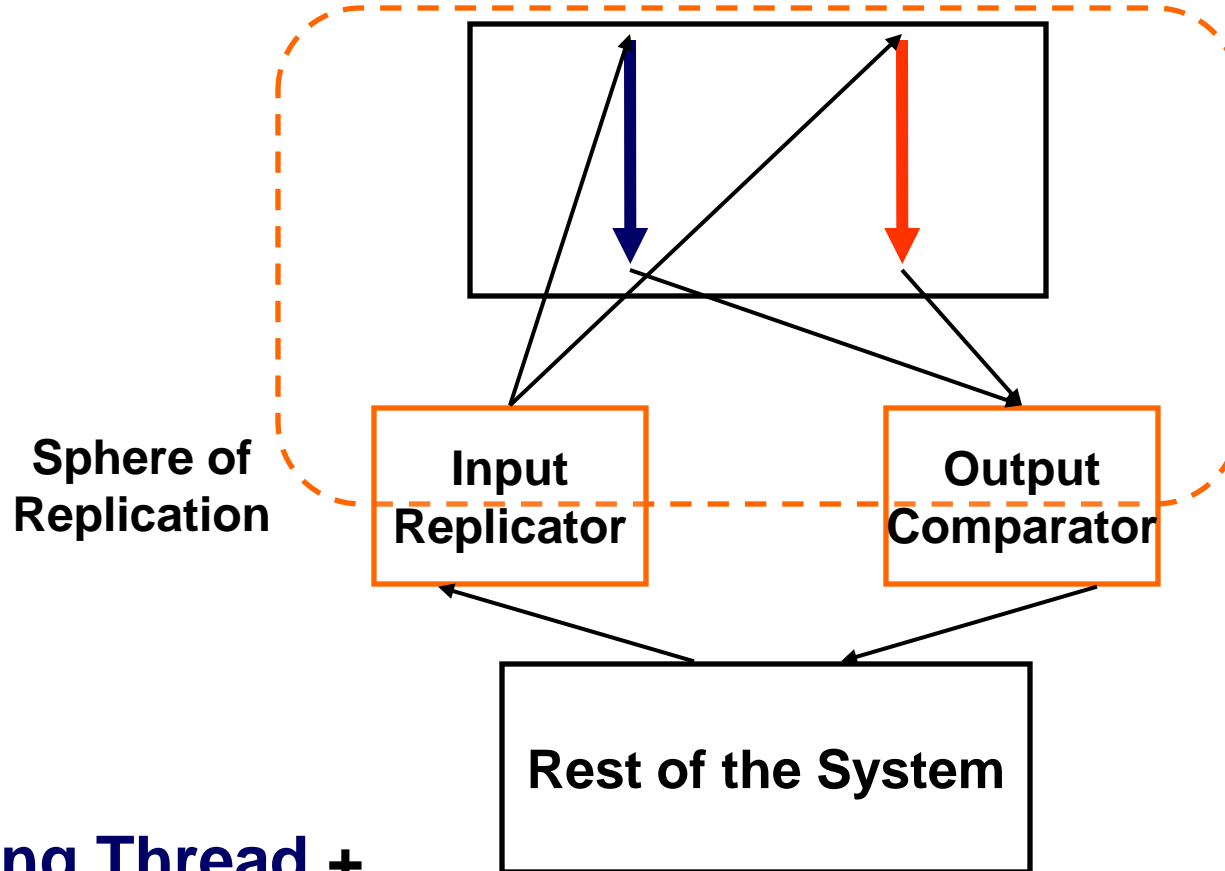


Redundant Multi-Threading (RMT)



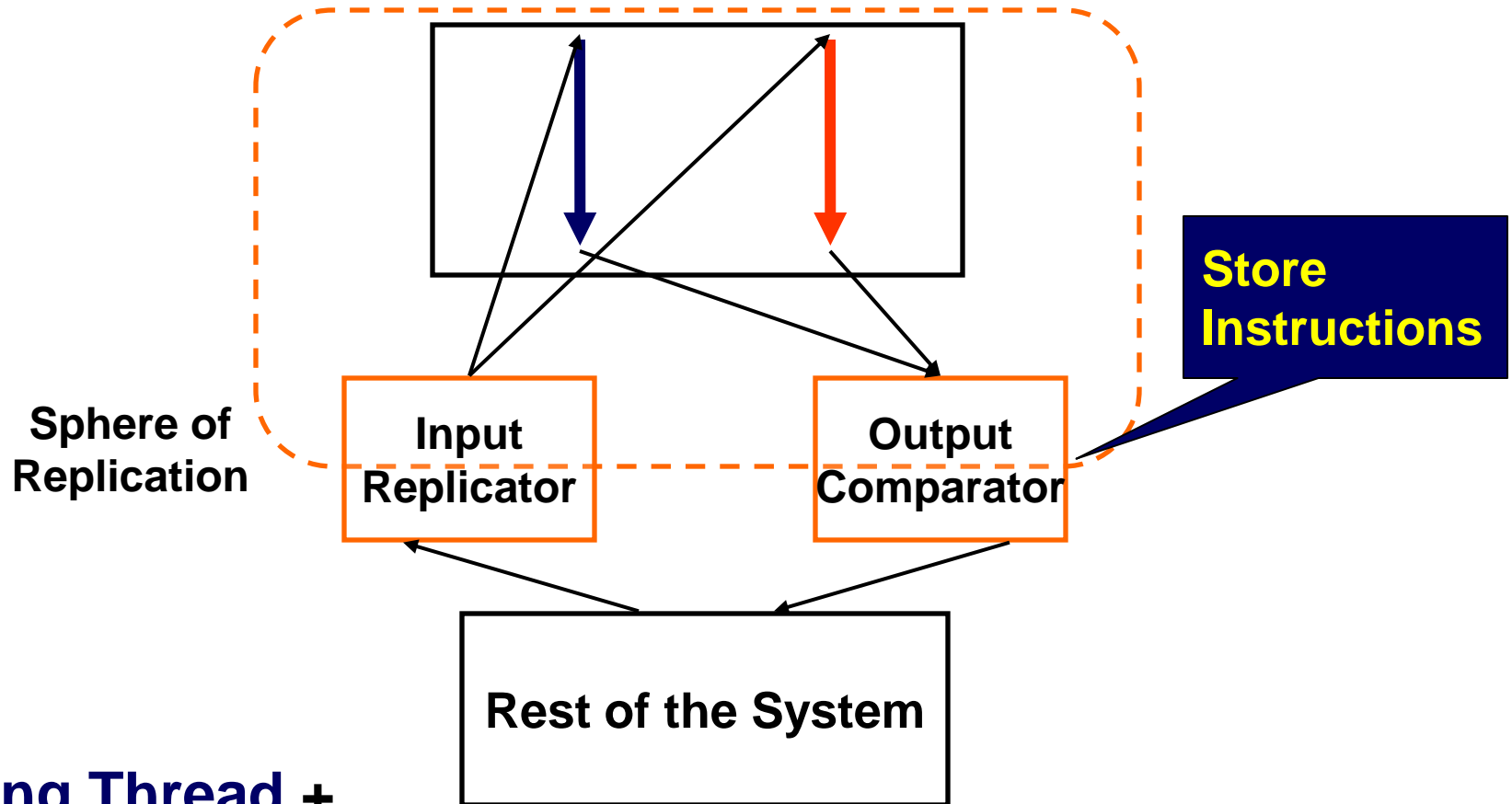
**Leading Thread +
Trailing Thread**

Redundant Multi-Threading (RMT)



**Leading Thread +
Trailing Thread**

Redundant Multi-Threading (RMT)



**Leading Thread +
Trailing Thread**

Talk Outline

- Partial Redundant Multi-Threading - Slick
- Runtime AVF Prediction
- NBTI Recovery Boosting
- Conclusions

RMT



All vs. No Coverage

Only two performance points

RMT



All vs. No Coverage

Only two performance points

Partial RMT



Tunable Error Coverage

Multiple performance points

Partial RMT

[Parashar et al., ASPLOS 2006]

- Reduce the number of instructions in the redundant thread
 - **“Knob”**: Number of instructions removed
 - Need to preserve register dependences
 - Examples: Slipstream, DIE-IRB, ReStore

Partial RMT

[Parashar et al., ASPLOS 2006]

- Reduce the number of instructions in the redundant thread
 - **“Knob”**: Number of instructions removed
 - Need to preserve register dependences
 - Examples: Slipstream, DIE-IRB, ReStore
- **Slice Kill (Slick)**

Partial RMT

[Parashar et al., ASPLOS 2006]

- Reduce the number of instructions in the redundant thread
 - **“Knob”**: Number of instructions removed
 - Need to preserve register dependences
 - Examples: Slipstream, DIE-IRB, ReStore
- **Slice Kill (SlicK)**
 - Execute at the granularity of dependence chains/Slices


Partial RMT

[Parashar et al., ASPLOS 2006]

- Reduce the number of instructions in the redundant thread
 - **“Knob”**: Number of instructions removed
 - Need to preserve register dependences
 - Examples: Slipstream, DIE-IRB, ReStore
- **Slice Kill (Slick)**
 - Execute at the granularity of dependence chains/Slices
 - Both program dataflow and control flow exhibit predictable behavior

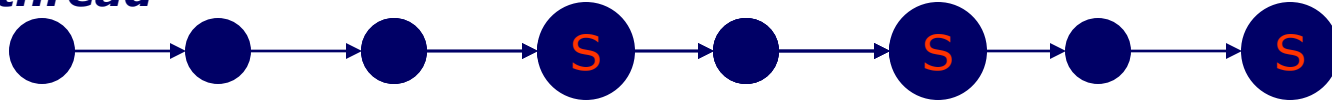
Partial RMT

[Parashar et al., ASPLOS 2006]

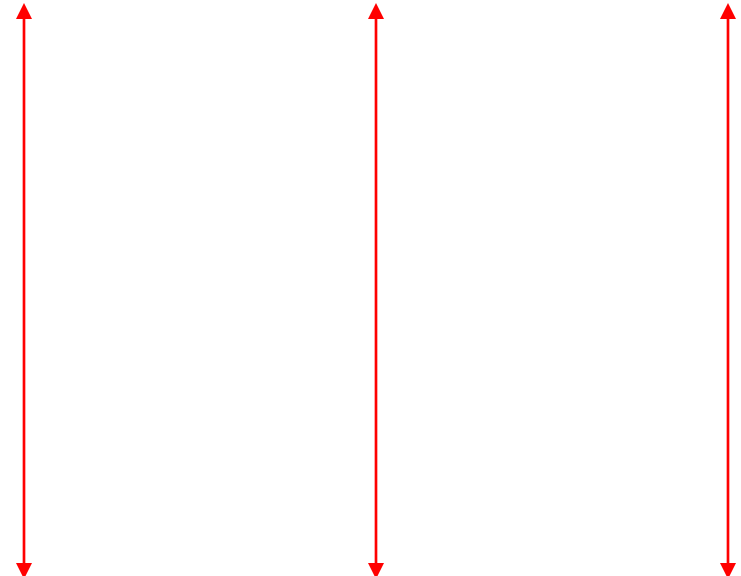
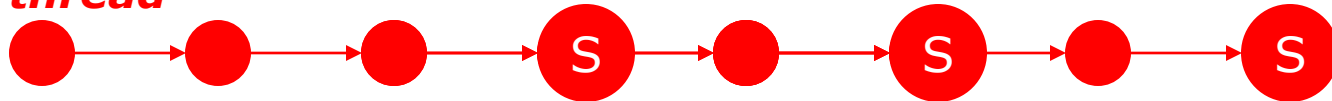
- Reduce the number of instructions in the redundant thread
 - **“Knob”**: Number of instructions removed
 - Need to preserve register dependences
 - Examples: Slipstream, DIE-IRB, ReStore
 - **Slice Kill (Slick)**
 - Execute at the granularity of dependence chains/Slices
 - Both program dataflow and control flow exhibit predictable behavior
-  **Use high-confidence speculation as a substitute for redundant execution**

The Slick Mechanism

leading thread

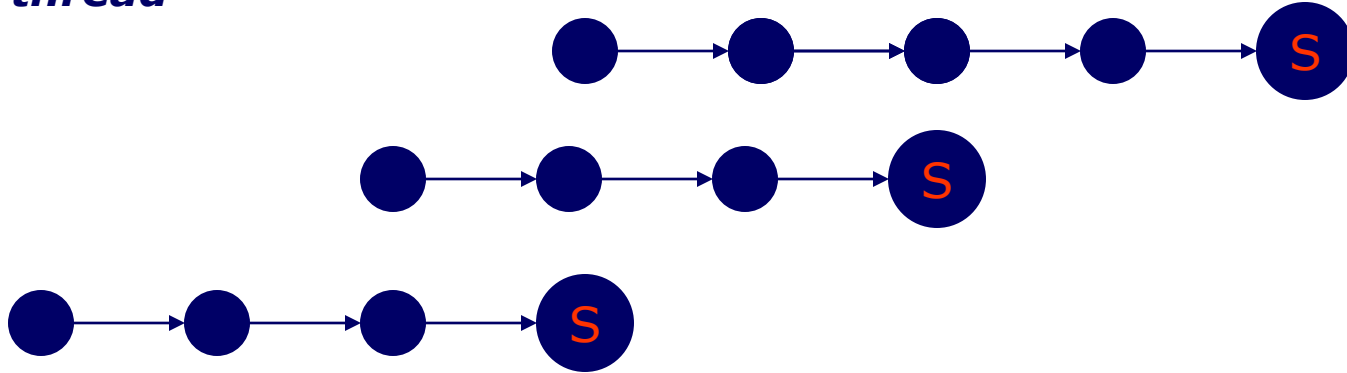


trailing thread

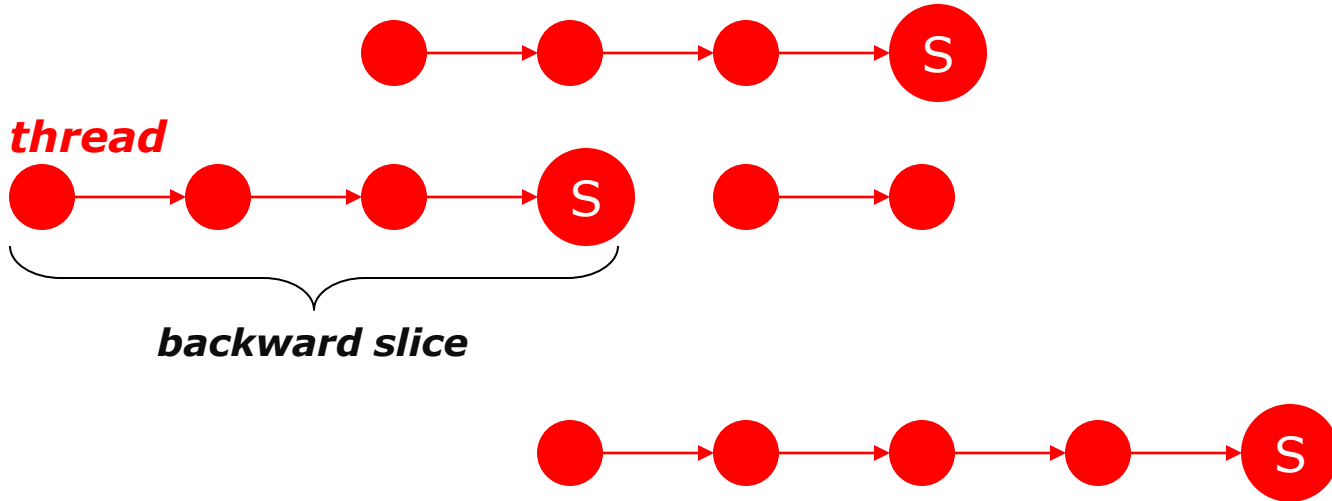


The Slick Mechanism

leading thread

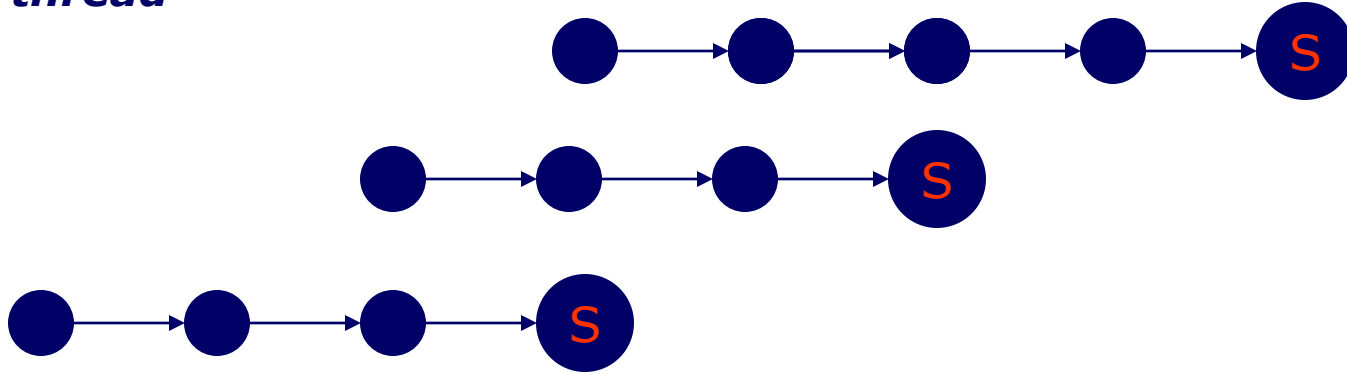


trailing thread

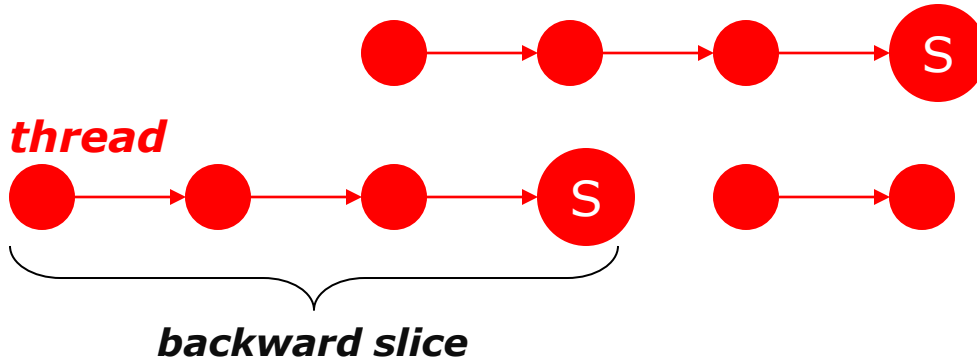


The Slick Mechanism

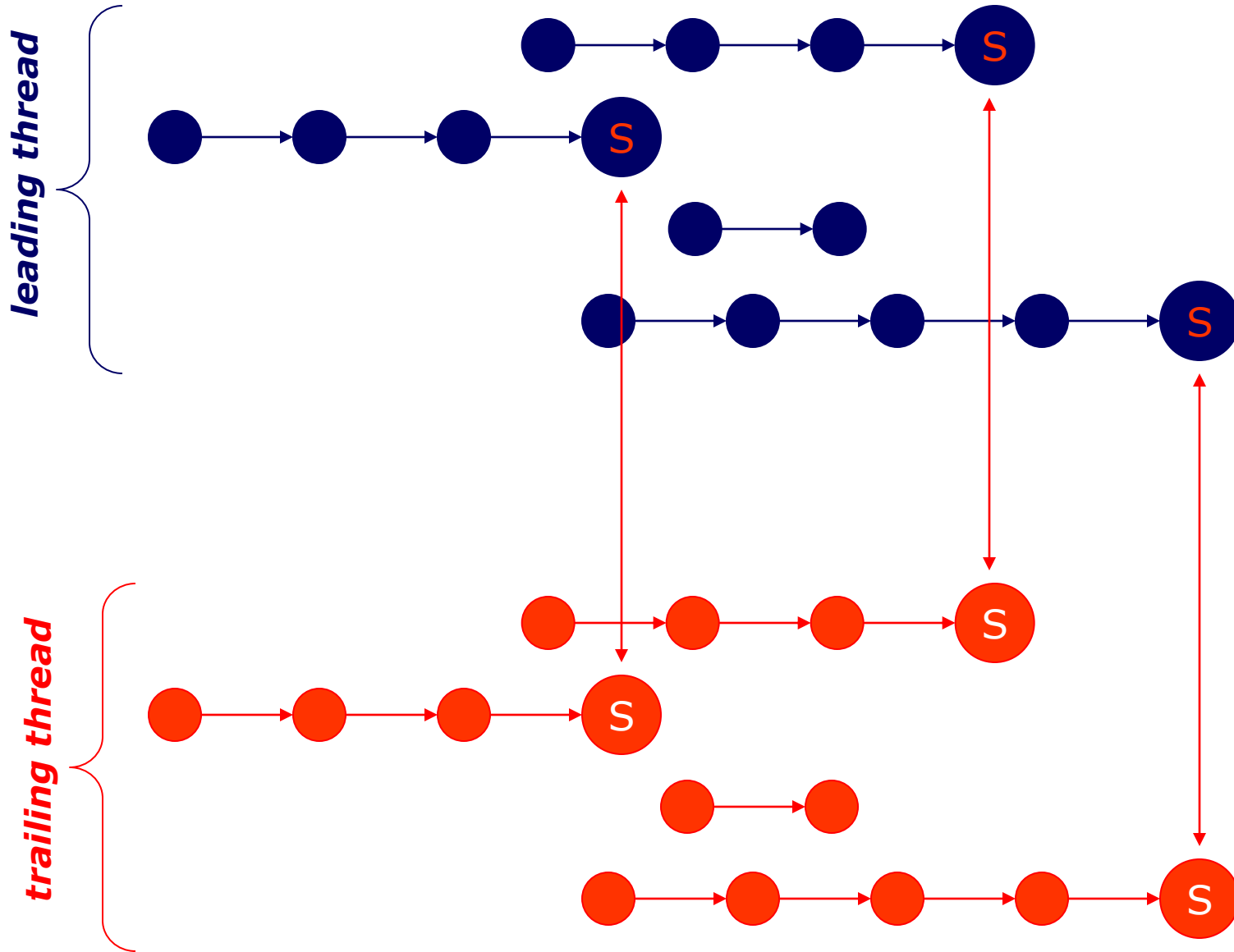
leading thread



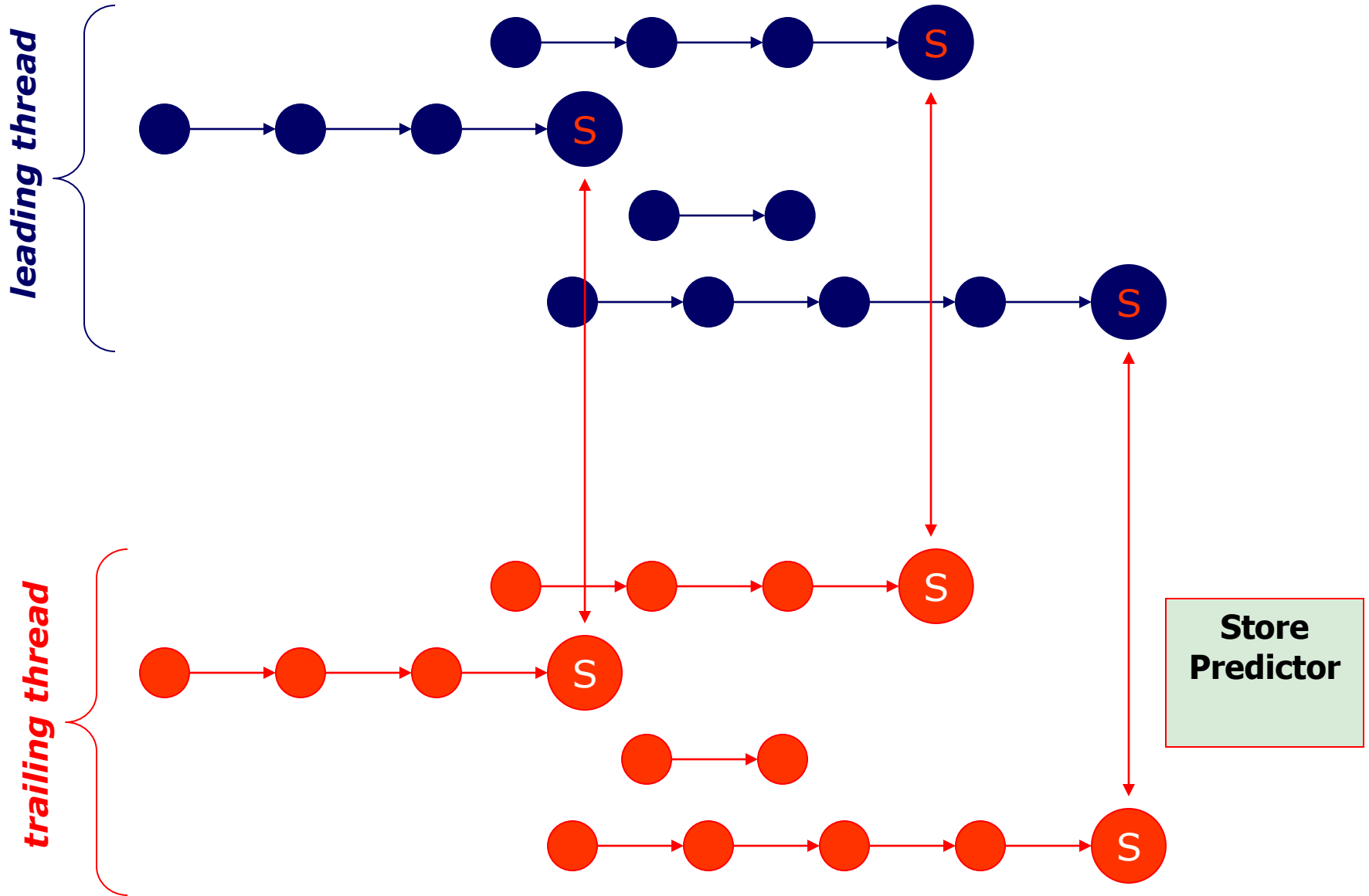
trailing thread



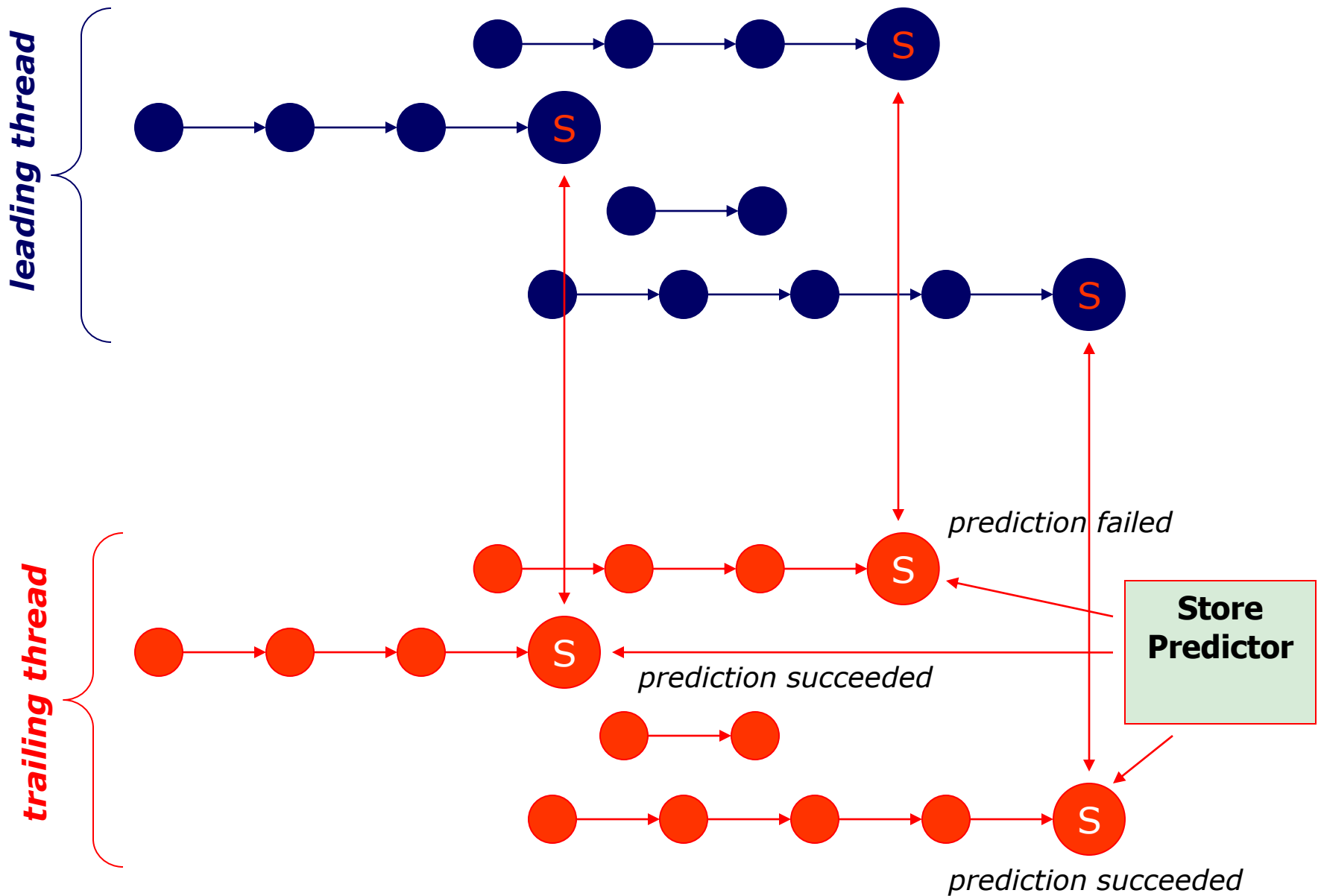
The Slick Policy



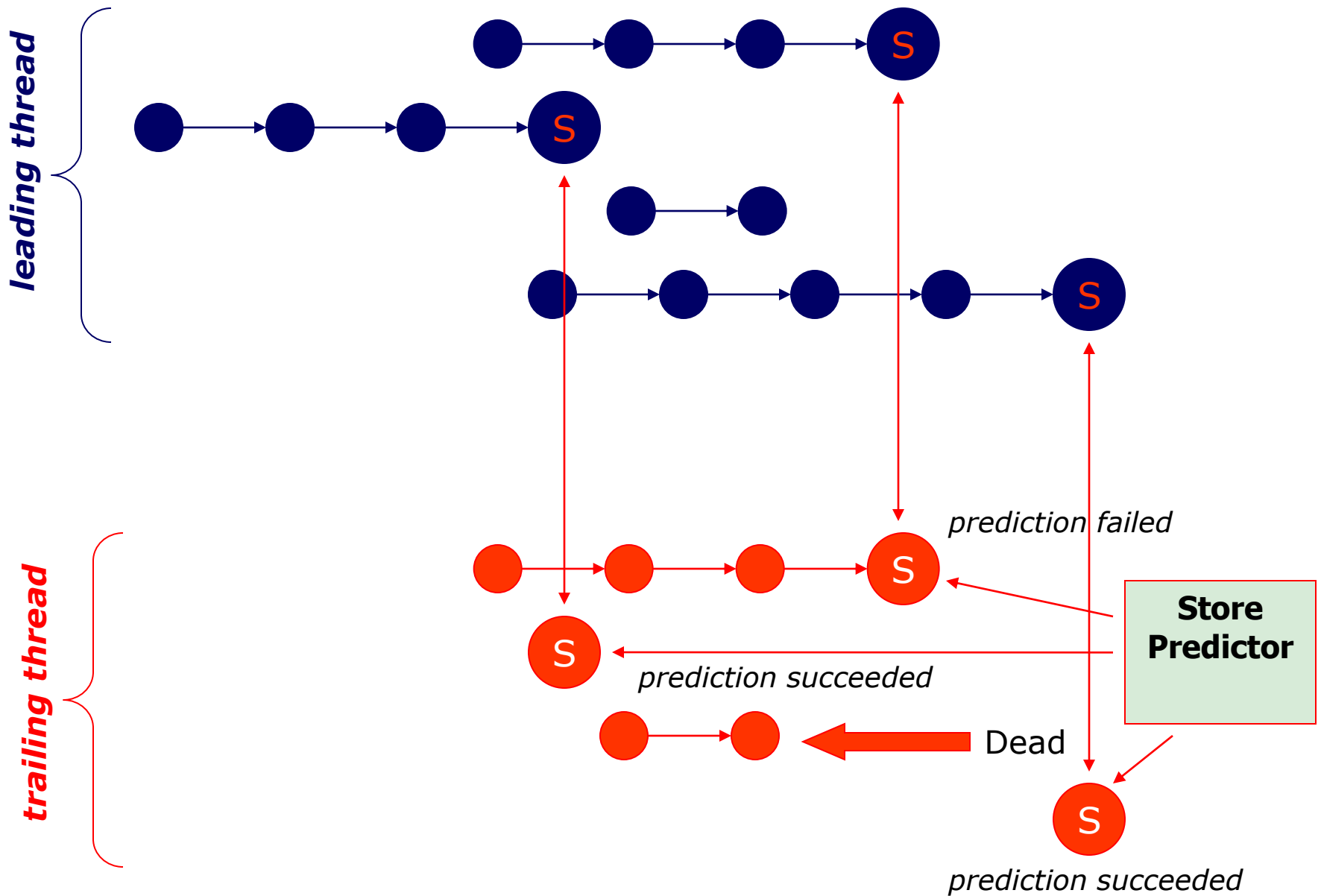
The Slick Policy



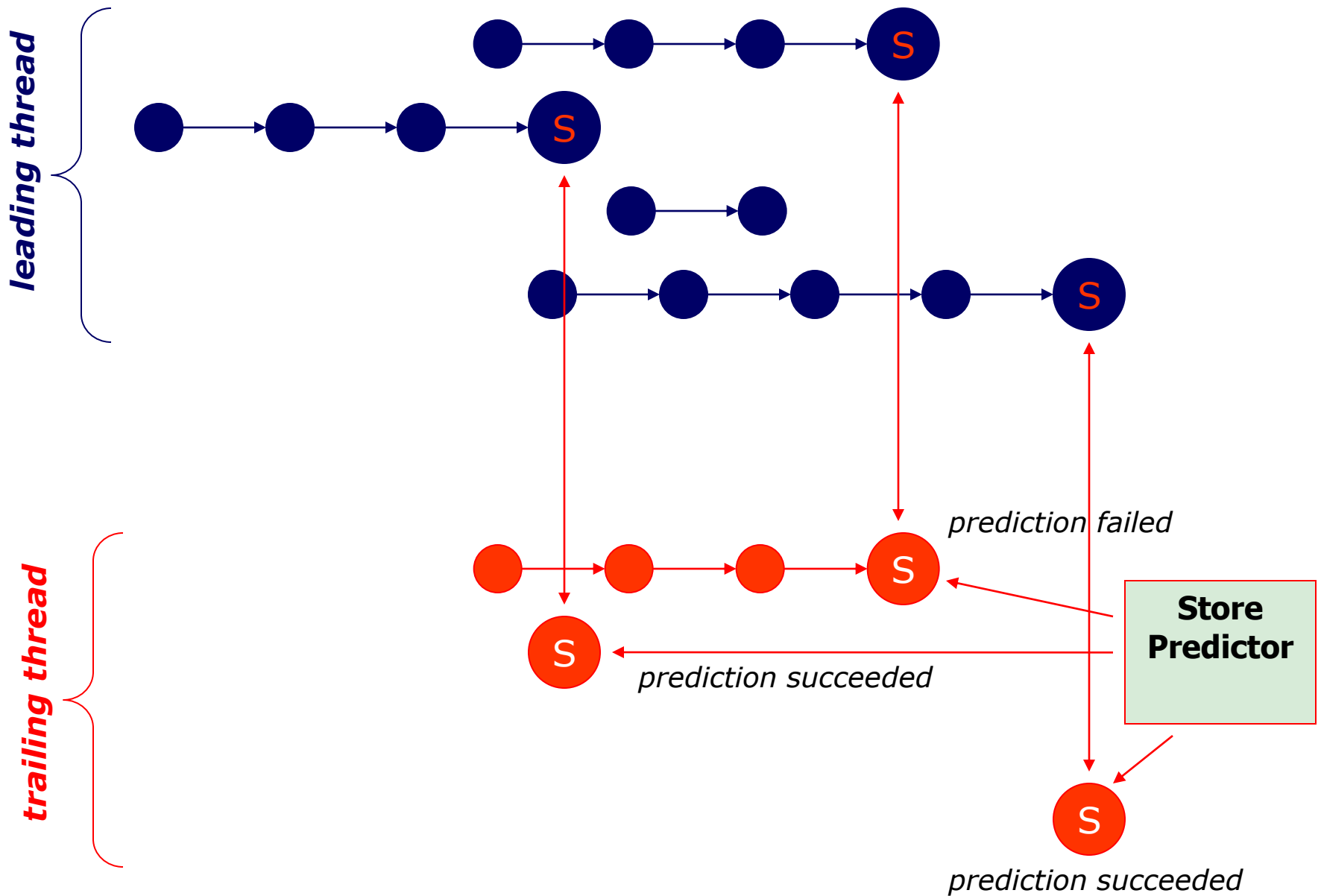
The Slick Policy



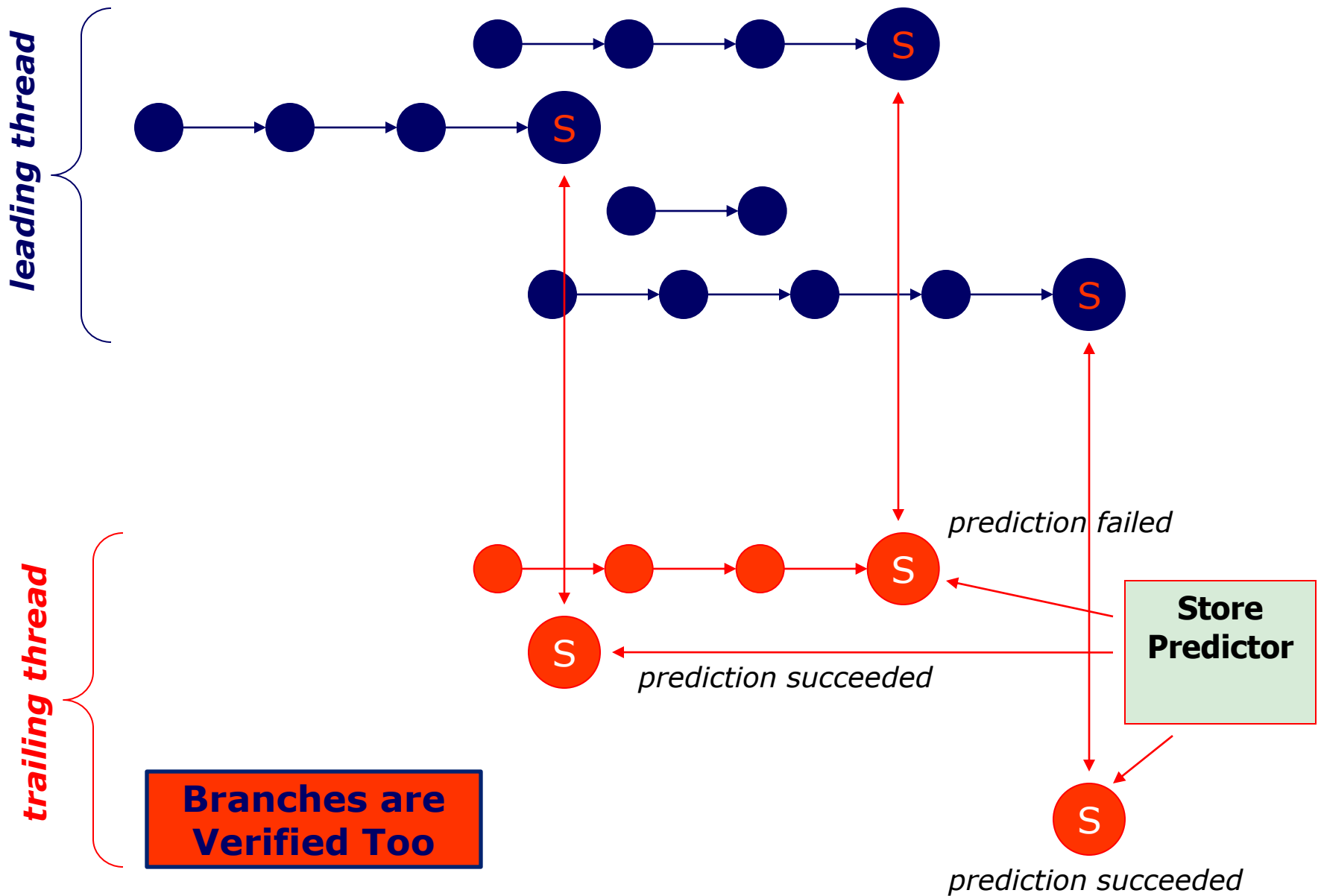
The Slick Policy



The Slick Policy



The Slick Policy



Overview of Slick Design

- Predictors Used:
 - Predictor Outputs: Prediction, **No-Predict**
 - **Stores**: Last-value predictor with saturating counters
 - **Branches**: Branch predictor + confidence estimator

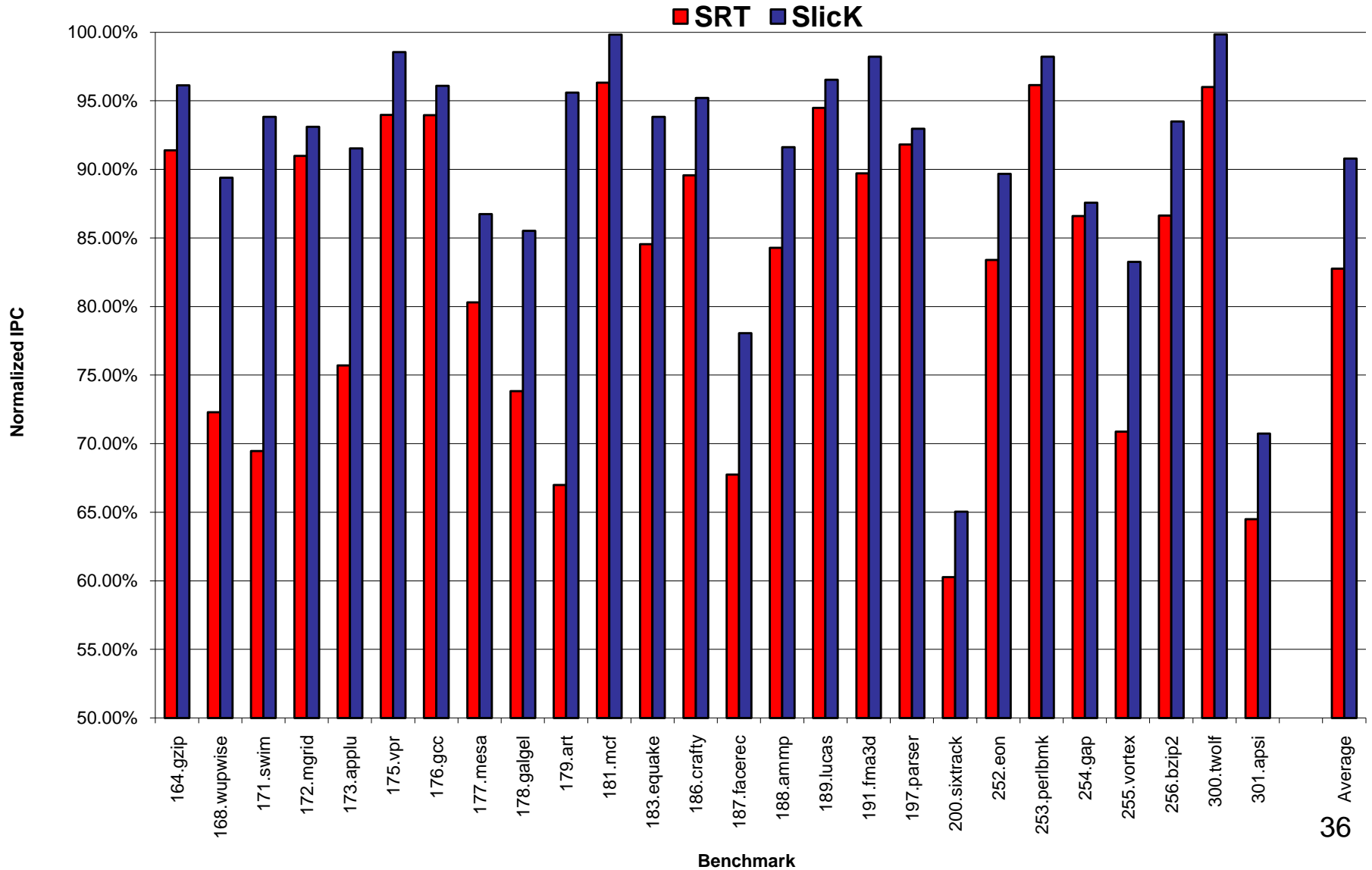
Overview of Slick Design

- Predictors Used:
 - Predictor Outputs: Prediction, **No-Predict**
 - **Stores**: Last-value predictor with saturating counters
 - **Branches**: Branch predictor + confidence estimator
- Slice extractor needs to provide a smooth flow of instructions through the pipeline
 - **Slice Extraction Matrix (SliceEM)**

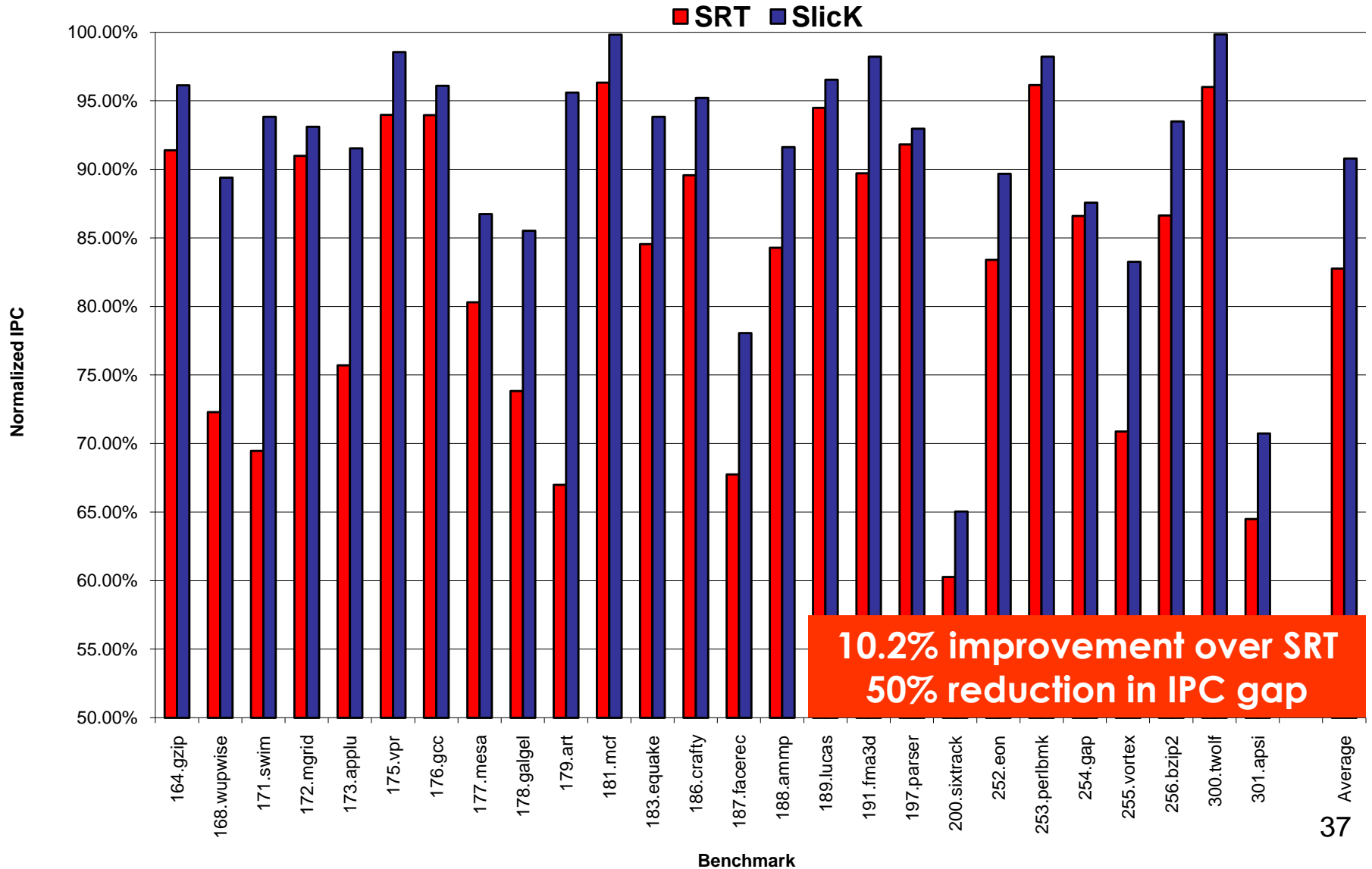
Overview of Slick Design

- Predictors Used:
 - Predictor Outputs: Prediction, **No-Predict**
 - **Stores**: Last-value predictor with saturating counters
 - **Branches**: Branch predictor + confidence estimator
- Slice extractor needs to provide a smooth flow of instructions through the pipeline
 - **Slice Extraction Matrix (SliceEM)**
- Evaluated for the RMT implementation of an SMT processor
 - Simultaneous and Redundant Threading (SRT)
[Reinhardt and Mukherjee, ISCA 2000]

Slick Performance Results



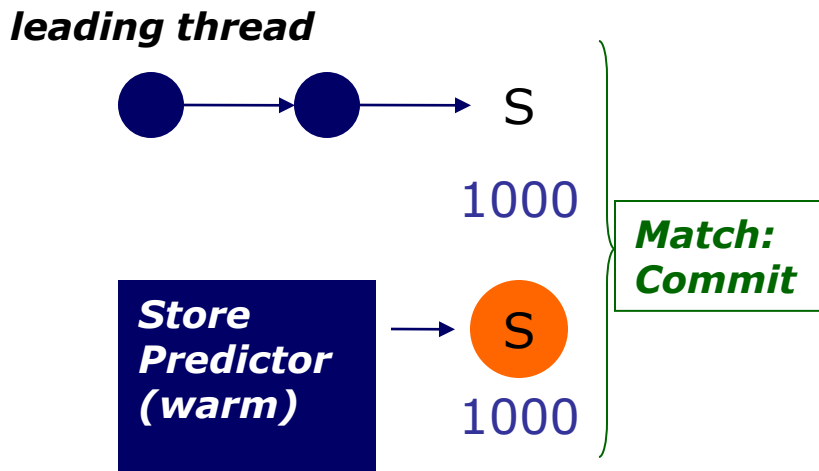
Slick Performance Results



Slick Fault Coverage: The Common Case #1

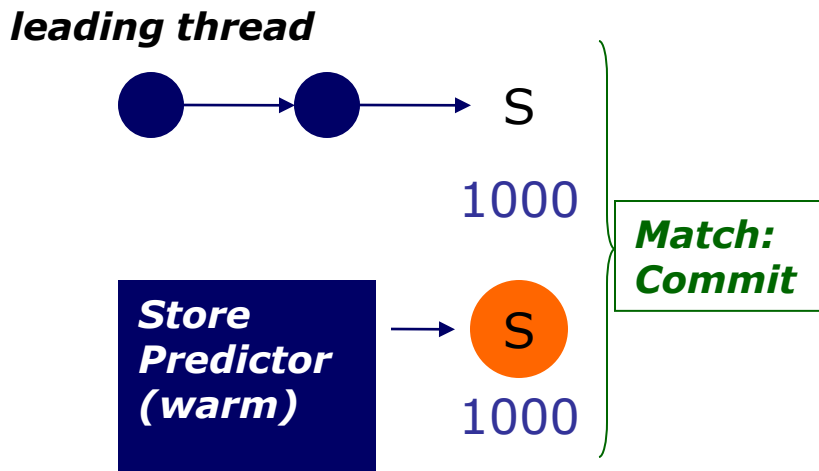
Slick Fault Coverage: The Common Case #1

Fault-Free Execution

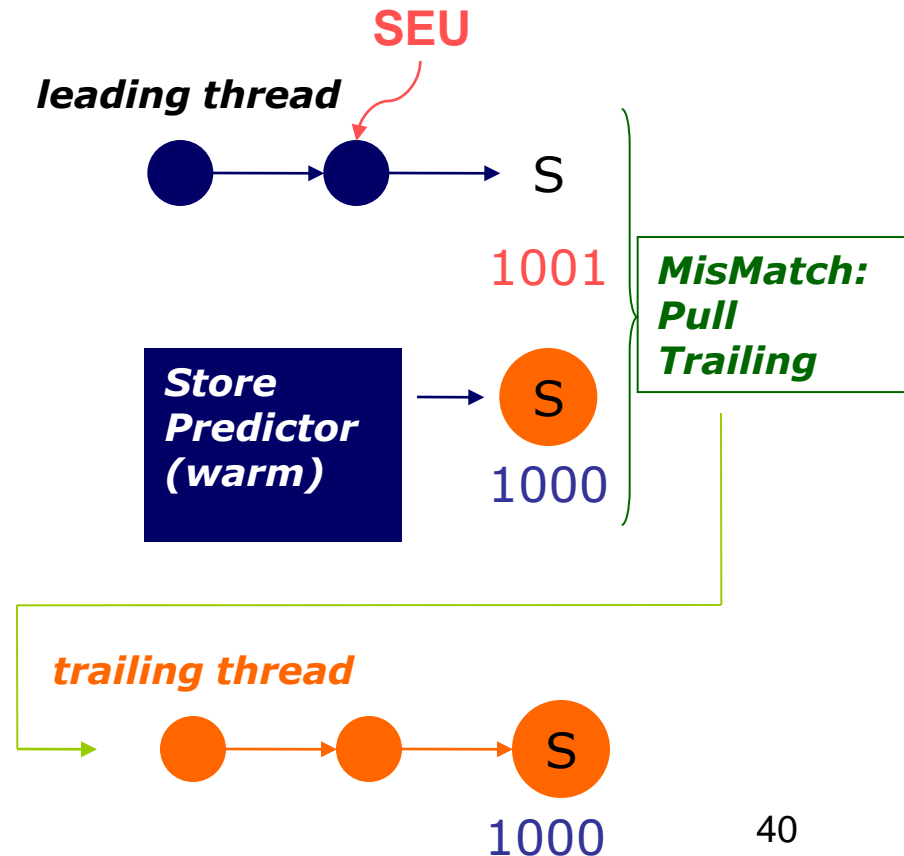


Slick Fault Coverage: The Common Case #1

Fault-Free Execution

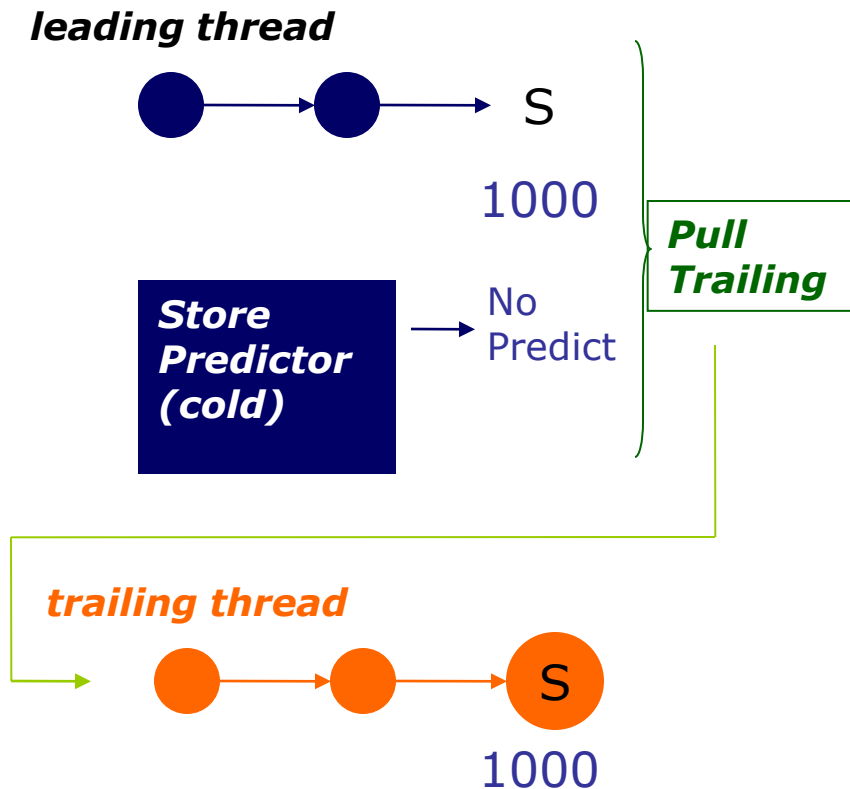


Erroneous Execution



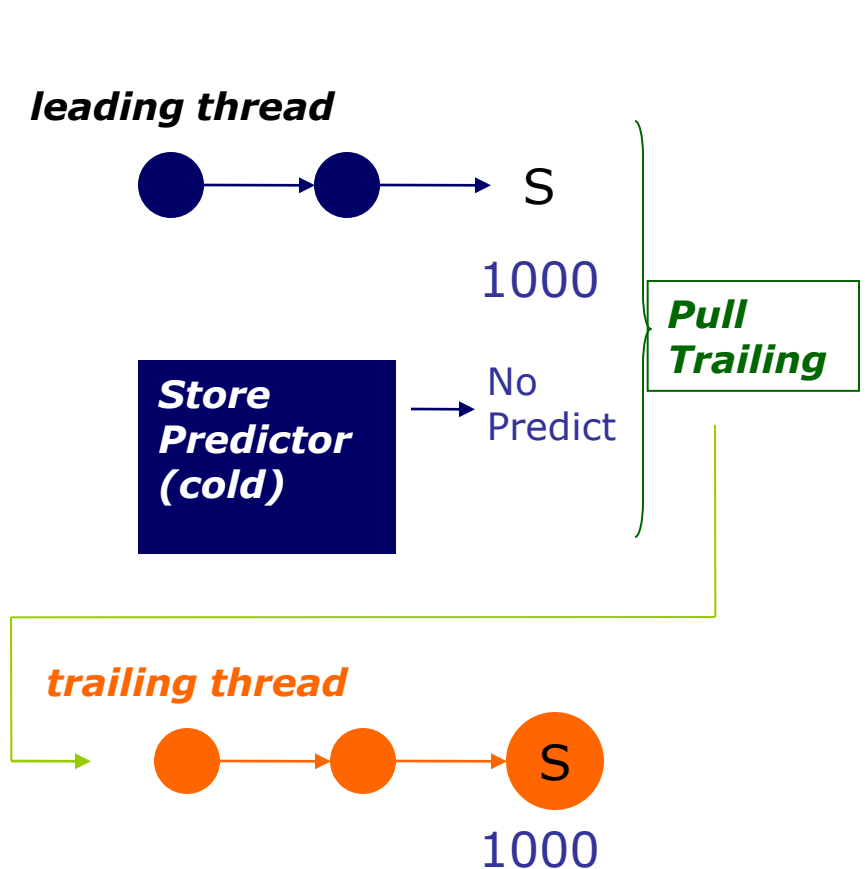
Slick Fault Coverage: The Common Case #2

Fault-Free Execution

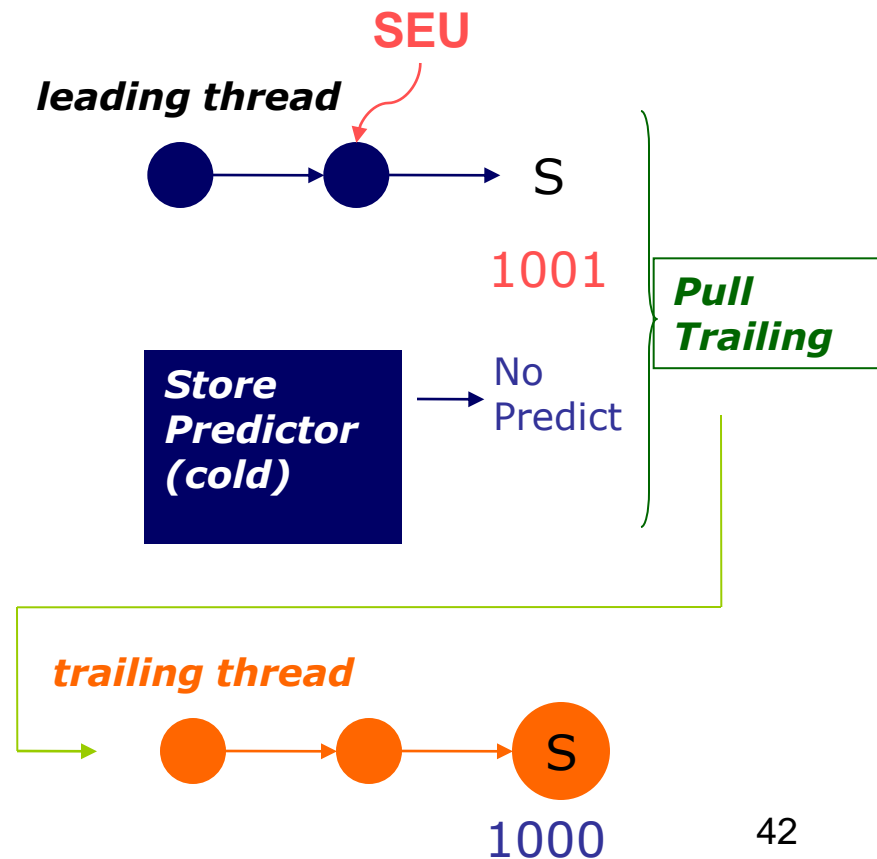


Slick Fault Coverage: The Common Case #2

Fault-Free Execution



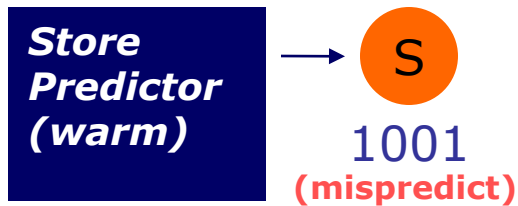
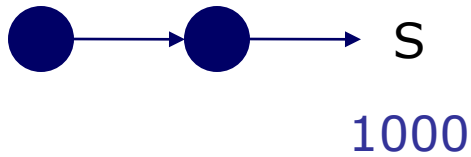
Erroneous Execution



Slick Fault Coverage: The Uncommon Case

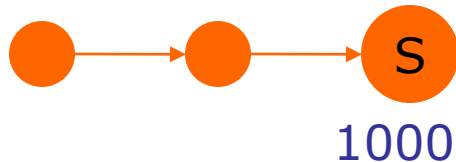
Fault-Free Execution

leading thread



**Mismatch:
Pull
Trailing**

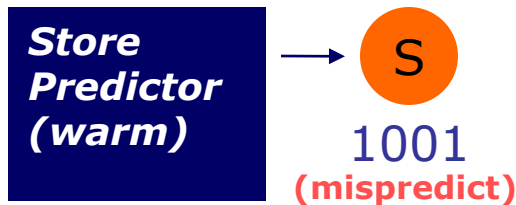
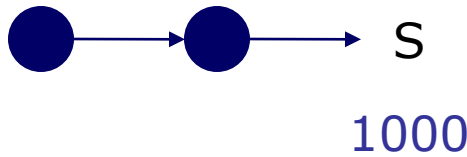
trailing thread



Slick Fault Coverage: The Uncommon Case

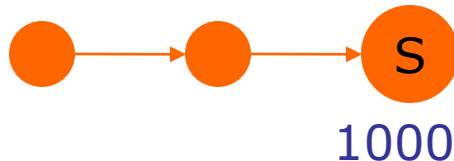
Fault-Free Execution

leading thread



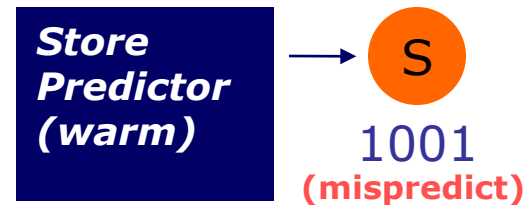
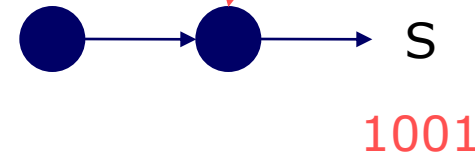
**Mismatch:
Pull
Trailing**

trailing thread



Erroneous Execution

leading thread



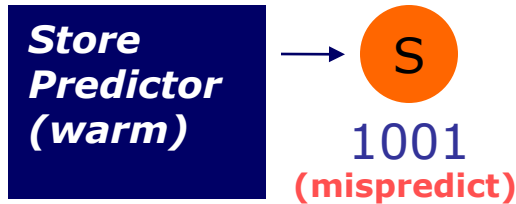
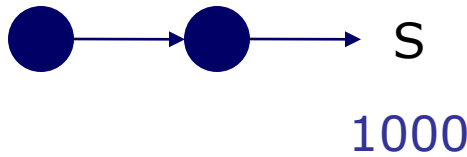
**Match:
Commit**

**Silent Data
Corruption!**

Slick Fault Coverage: The Uncommon Case

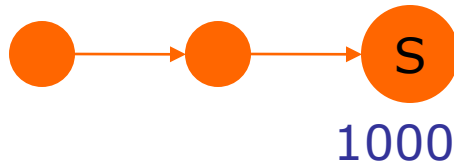
Fault-Free Execution

leading thread



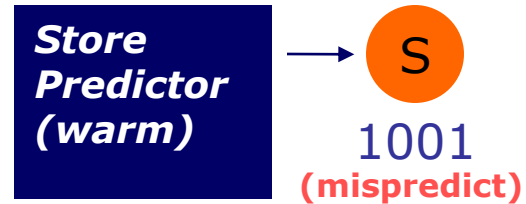
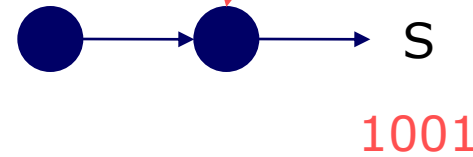
**Mismatch:
Pull
Trailing**

trailing thread



Erroneous Execution

leading thread



**Match:
Commit**

**Silent Data
Corruption!**

**The Store and its Backward
Slice are UNGUARDED**

Architectural Vulnerability Factor (AVF)

- The probability that a fault will result in an externally visible error
- Only a subset of the bits affect Architecturally Correct Execution (ACE bits)
- $AVF = 0\%$ for structures within Sphere of Replication in RMT

Architectural Vulnerability Factor (AVF)

- The probability that a fault will result in an externally visible error
- Only a subset of the bits affect Architecturally Correct Execution (ACE bits)
- AVF = 0% for structures within Sphere of Replication in RMT
- Unguarded instructions = ACE

Architectural Vulnerability Factor (AVF)

- The probability that a fault will result in an externally visible error
- Only a subset of the bits affect **Architecturally Correct Execution (ACE bits)**
- AVF = 0% for structures within Sphere of Replication in RMT
- **Unguarded instructions = ACE**
- **AVF ~ 0%-2% for RUU, ISQ, and LSQ**
 - Single threaded AVFs ~ 20%-30%

Partial RMT



Partial RMT



Soft Error Measurement

Runtime AVF Prediction

Measuring Runtime AVF

[Walcott et al., ISCA 2007]

- Little's Law AVF Estimate = $(B_{ace})(L_{ace})/\# \text{ Bits}$

B_{ace} : average bandwidth of the ACE bits into the structure

L_{ace} : average residence time of an ACE bit in the structure

- Direct measurement in hardware is difficult

Measuring Runtime AVF

[Walcott et al., ISCA 2007]

- Little's Law AVF Estimate = $(B_{ace})(L_{ace})/\# \text{ Bits}$

B_{ace} : average bandwidth of the ACE bits into the structure

L_{ace} : average residence time of an ACE bit in the structure

- Direct measurement in hardware is difficult
- **Our Approach:** Calculate AVF from very few, easily measurable metrics

Experimental Setup

- SimpleScalar 3.0 with SRT model
- Structures for AVF Analysis
 - RUU, ISQ, LSQ
- 26 SPEC2000 Benchmarks
 - Simulate all 100-million instruction SimPoints
 - Checkpoint simulation state (160 μ arch variables) and calculate AVF every 4 million instructions

Choosing the Right Metrics

- **IPC**

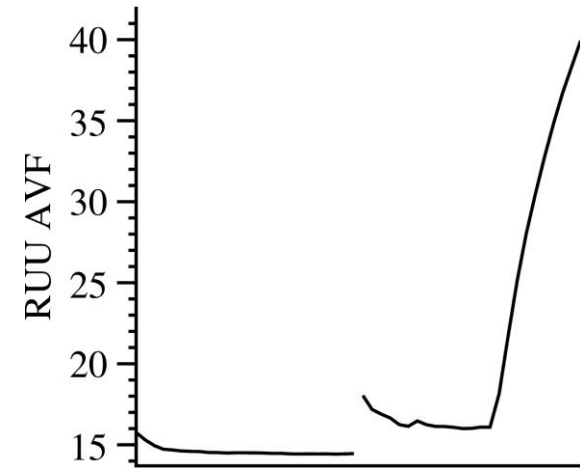
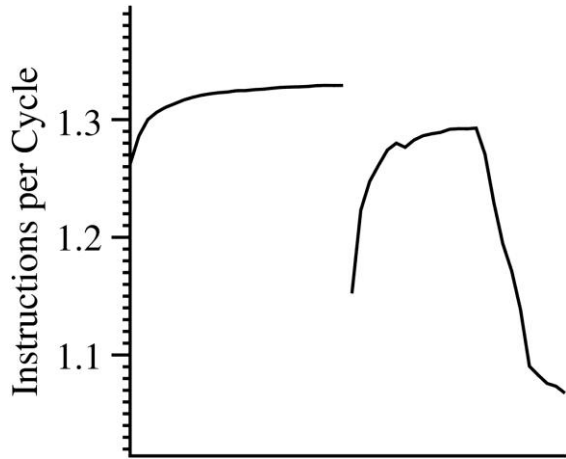
- Easy to measure
- Used to characterize program behavior

- Intuitively, High IPC could mean:

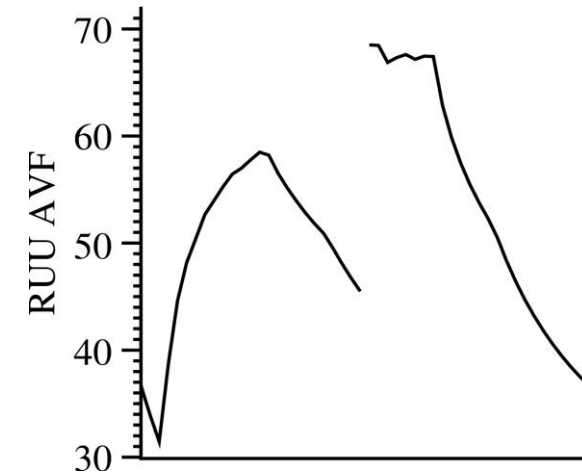
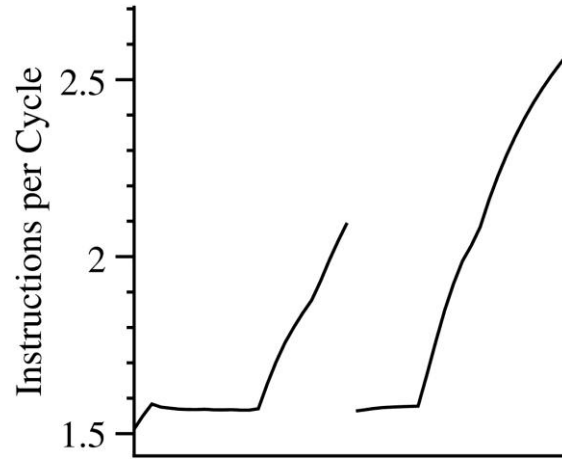
- More ACE bits ($\sim B_{ace}$) \Rightarrow Higher AVF
- Bits move faster through the pipeline ($\sim L_{ace}$) \Rightarrow Lower AVF

IPC vs. AVF

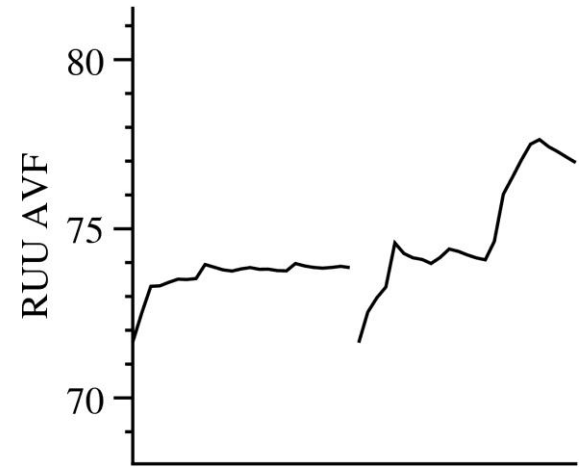
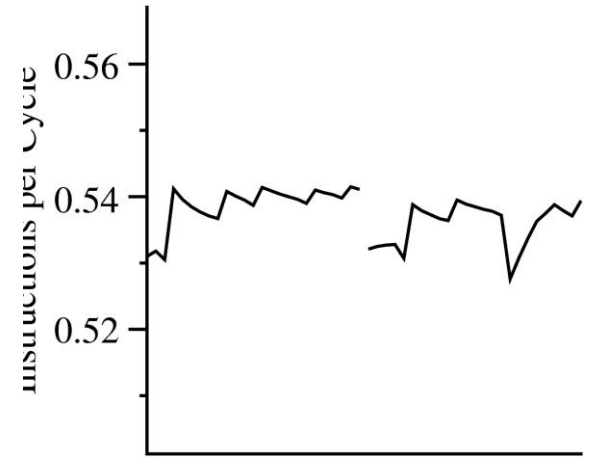
perlbnk



bzip2



art



Detecting Correlations

- Visual inspection is tedious
- Need a systematic way to identify the strongly correlated variables

Detecting Correlations

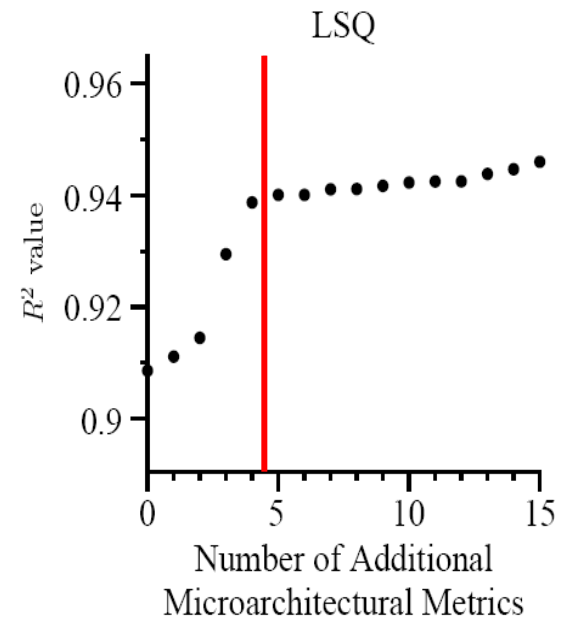
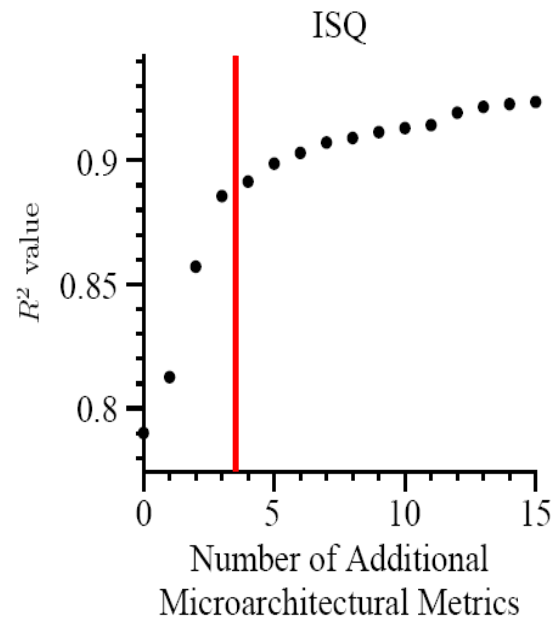
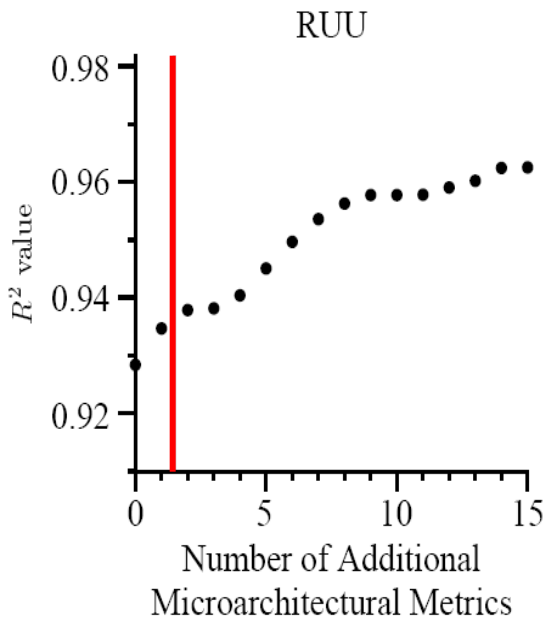
- Visual inspection is tedious
- Need a systematic way to identify the strongly correlated variables
- **Our Approach:** Use regression techniques
 - Chose 22 SPEC benchmarks for training
 - Remaining 4 used for testing the predictor
 - Used data from all the SimPoints

Linear Regression

- Iterative Process:
 - Include the single variable with the highest correlation
 - Consider each remaining variable one by one and compute regression of the 2-variable expression

Linear Regression

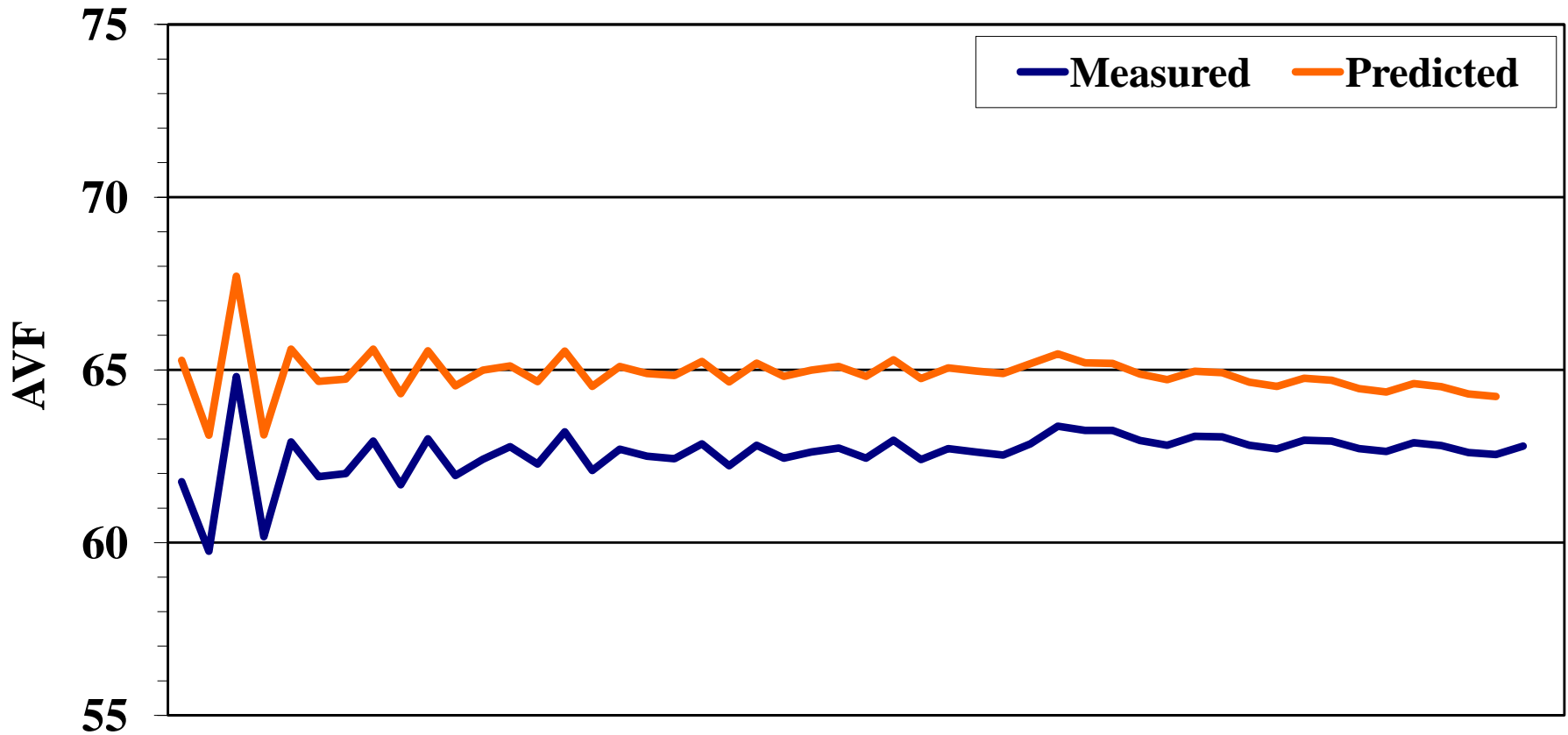
- Iterative Process:
 - Include the single variable with the highest correlation
 - Consider each remaining variable one by one and compute regression of the 2-variable expression



Predictor Testing

galge1 - RUU AVF

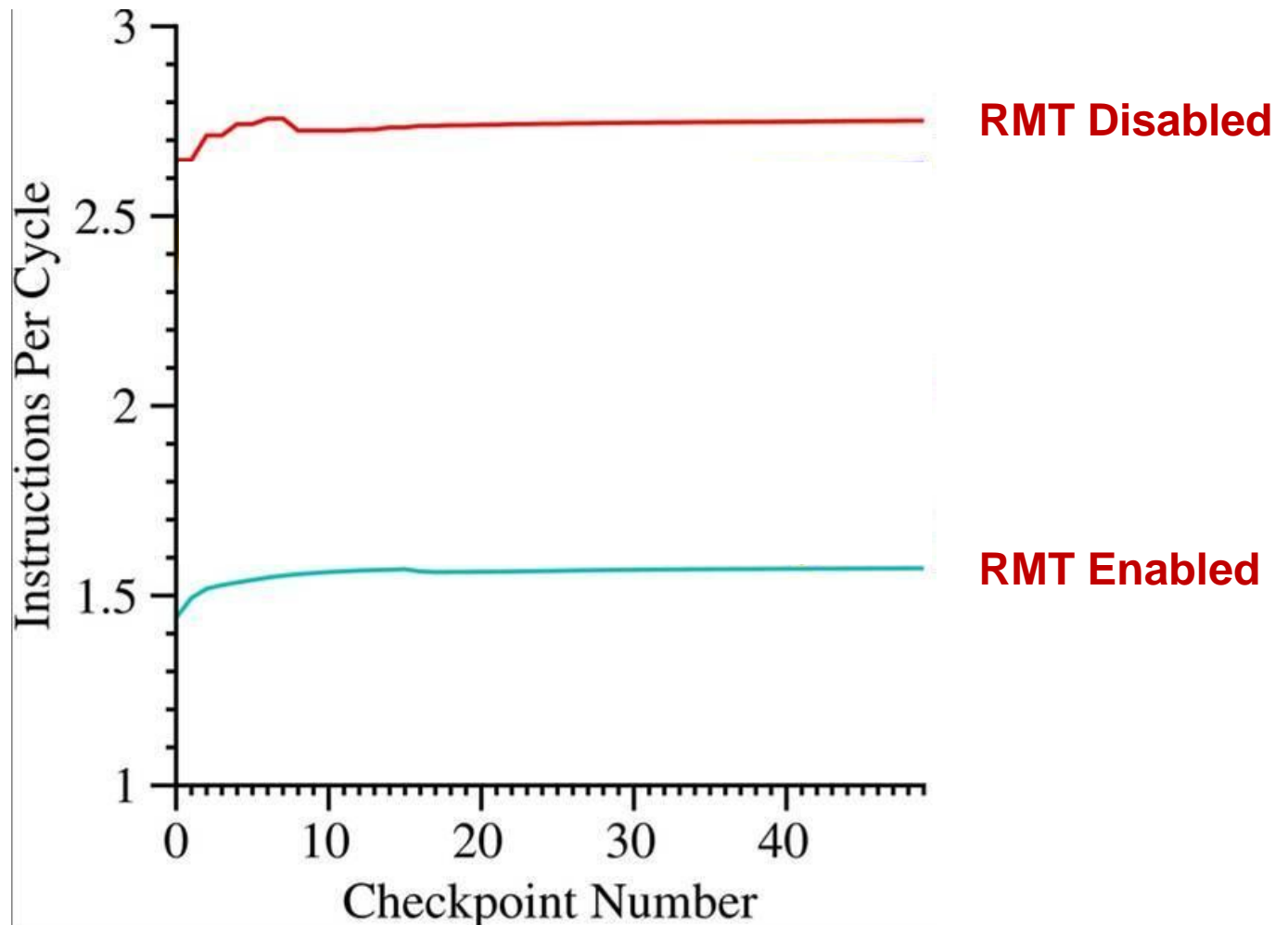
Multi-Variable Linear Predictor



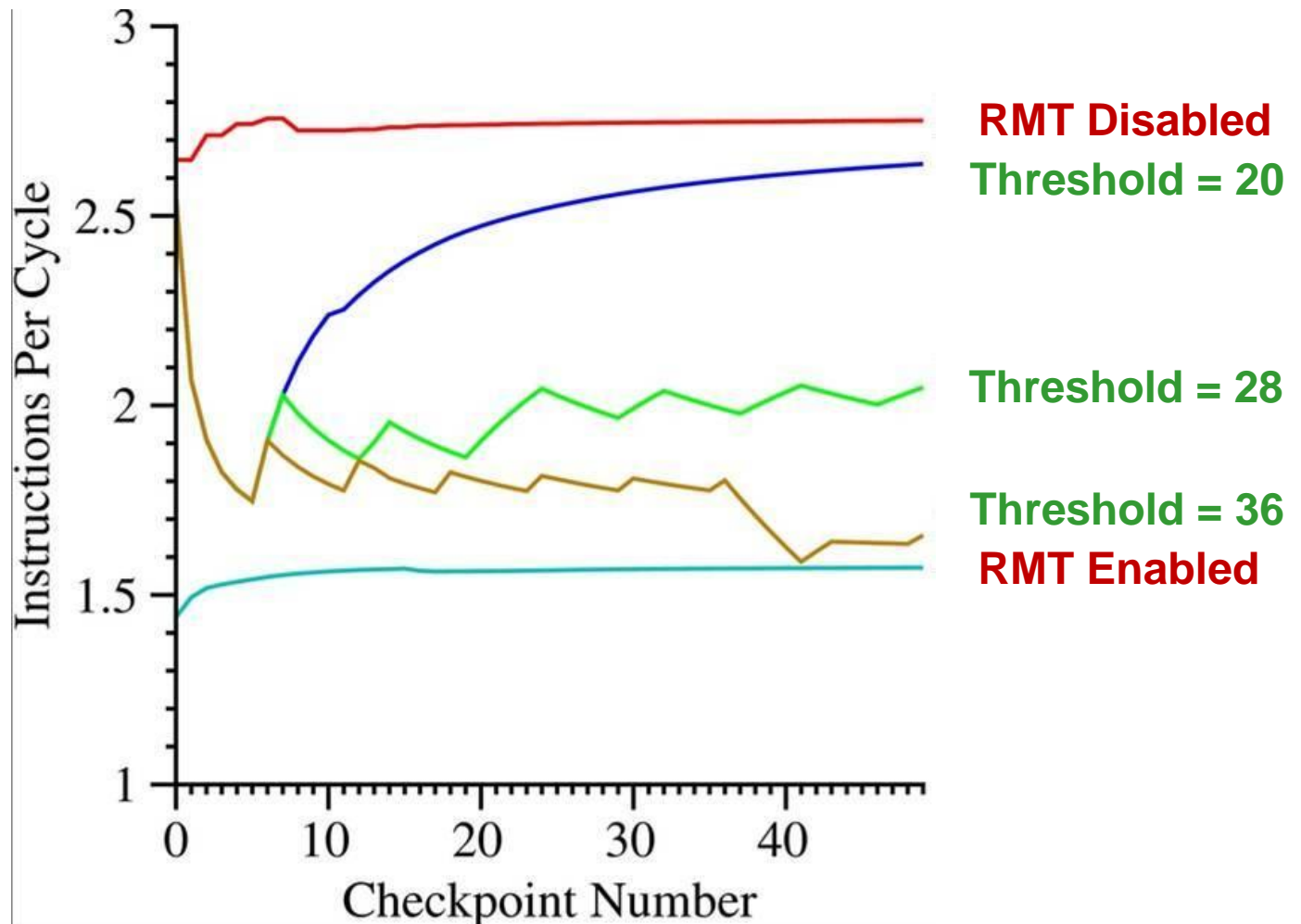
An AVF-Aware Partial RMT Policy

- Estimate the AVF of all structures every 2 million cycles.
- Sum the individual AVF values
- If (Sum > threshold): **enable RMT**
- After 10 million cycles: **disable RMT**

IPC Variations for `twolf`



IPC Variations for `twolf`



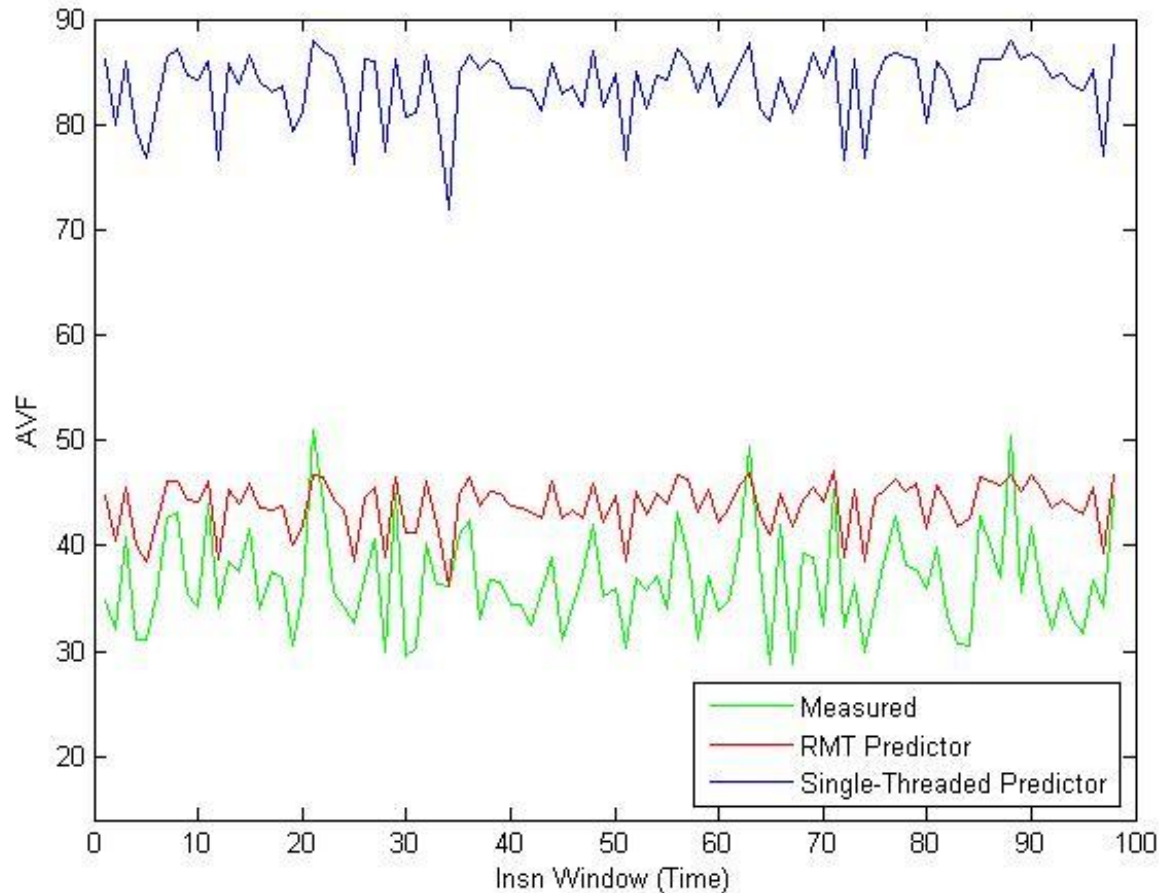
Predicting AVF in RMT Mode

[Sutton et al., SELSE 2009]

- To determine when RMT can be disabled
- Developed predictors of single-threaded mode AVF using metrics measured in RMT mode
- Conducted regression analysis between single-threaded mode AVF and μ arch metrics collected in RMT mode

RUU Predictor

mcf



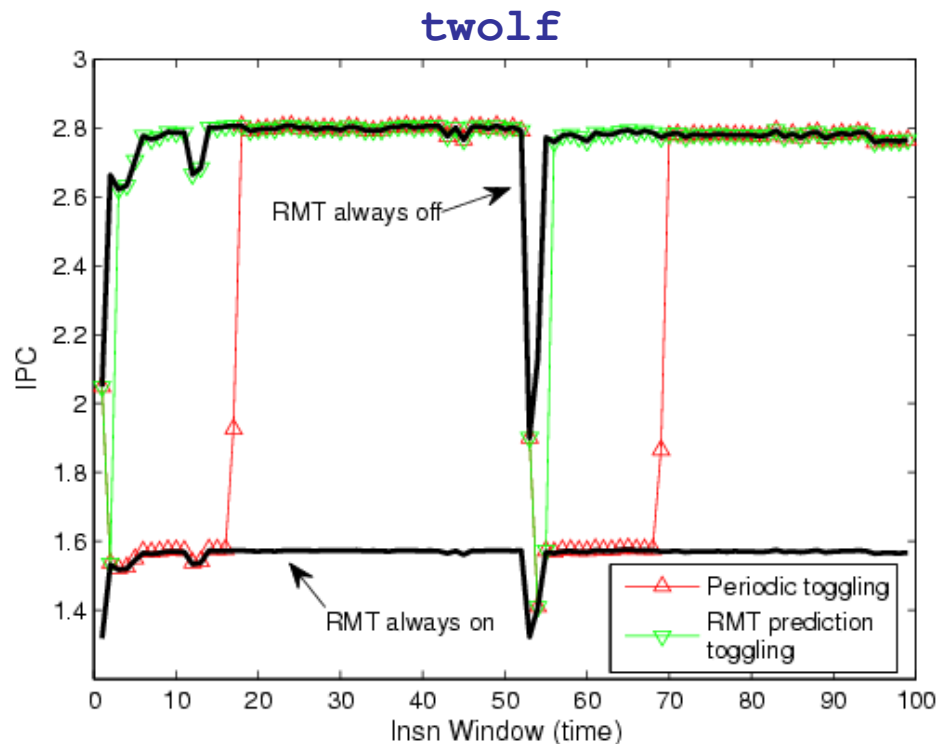
- Linear regression
- Predictor with most number of common variables with single-threaded mode predictor.

Another Partial RMT Policy

- Toggle RMT as long as AVF of any structure is greater than threshold
- Compare to periodic toggling every 10 million cycles

Another Partial RMT Policy

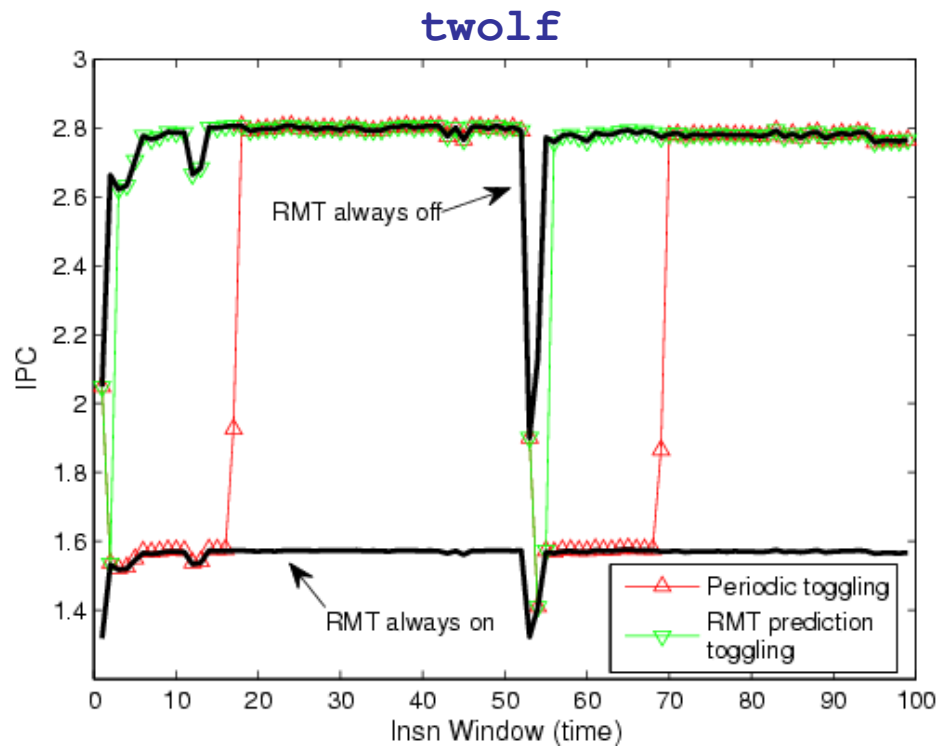
- Toggle RMT as long as AVF of any structure is greater than threshold
- Compare to periodic toggling every 10 million cycles



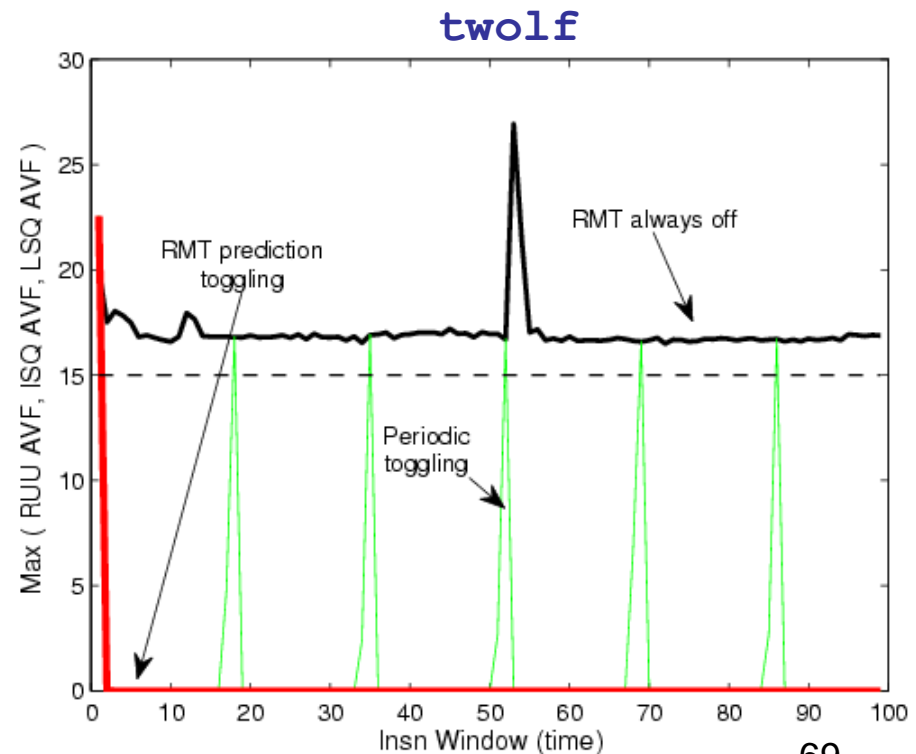
AVF Threshold = 25

Another Partial RMT Policy

- Toggle RMT as long as AVF of any structure is greater than threshold
- Compare to periodic toggling every 10 million cycles



AVF Threshold = 25



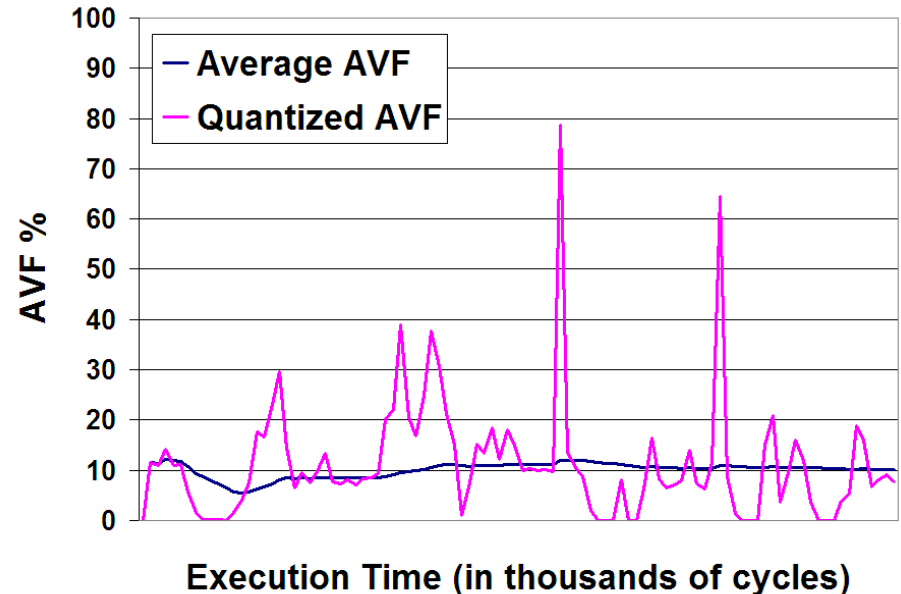
AVF Threshold = 15

Quantized AVF (Q-AVF)

[Biswas et al., SELSE 2009]

- Work with SPEARS Group, Intel
- **Goal:** Practical AVF predictor hardware
 - Fine-grained vulnerability tracking
 - Tracking vulnerability of groups of structures

`parser` - Store Buffer AVF x86 Pipeline



Average AVF: Average of Q-AVFs over all quantas

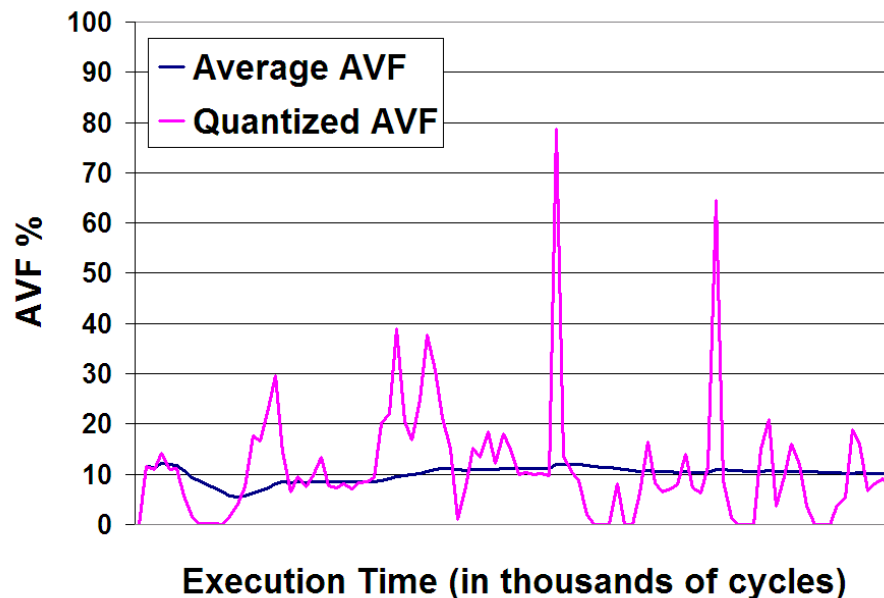
Q-AVF: AVF of a bit over a short interval of time

Quantized AVF (Q-AVF)

[Biswas et al., SELSE 2009]

- Work with SPEARS Group, Intel
- **Goal:** Practical AVF predictor hardware
 - Fine-grained vulnerability tracking
 - Tracking vulnerability of groups of structures
- Use linear regression approach to predict Q-AVF
- Intel Core™-like ASIM performance model
- Benchmarks: Spec2000, Spec2006, TPC-C

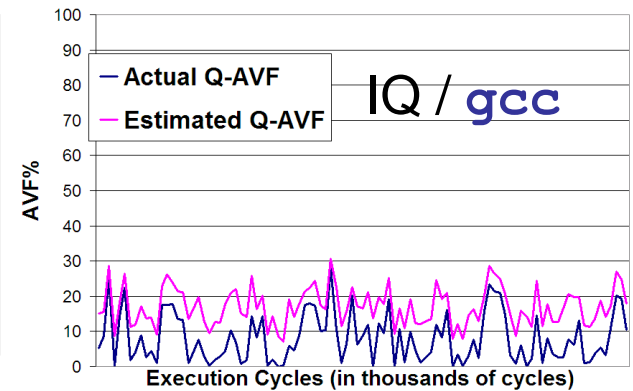
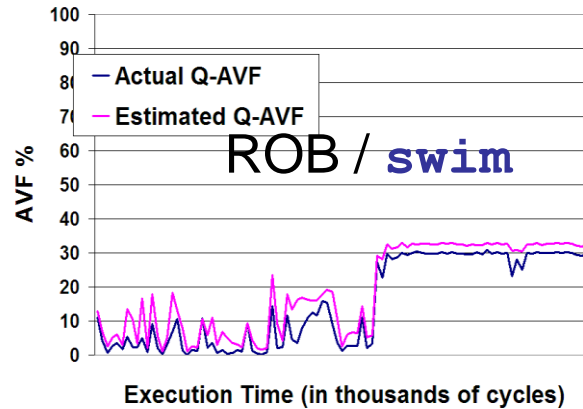
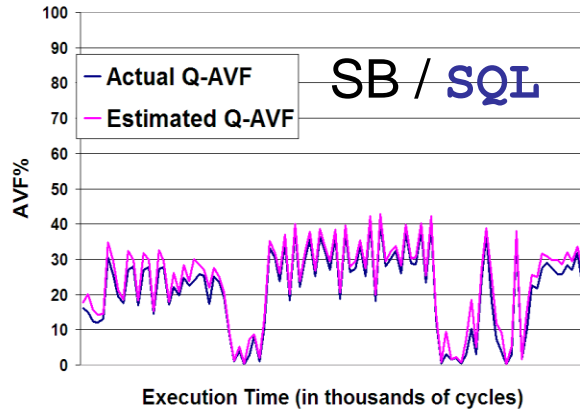
`parser` - Store Buffer AVF x86 Pipeline



Average AVF: Average of Q-AVFs over all quantas

Q-AVF: AVF of a bit over a short interval of time

Q-AVF Estimation Accuracy High (7 or fewer parameters per structure)



Structures	Mean Correlation across Benchmarks	Min Correlation across Benchmarks
IQ – Instruction Queue	0.87	0.81
MB – Memory Buffer	0.90	0.82
RS – Reservation Station	0.93	0.82
ROB – Reorder Buffer	0.95	0.86
STB – Store Buffer	0.98	0.94
LDB – Load Buffer	0.85	0.80

Aggregate Q-AVF Accuracy High (8 total parameters)

- **Only 8 input parameters used for all aggregate blocks:**

1. Stores Flushed before DTLB response (ST_Flush)
2. STB Utilization (ST_Util)
3. ROB Empty Cycles (ROB_Empty)
4. ROB Utilization (ROB_Util)
5. Branch Mis-predicts (Br_Miss)
6. RS Utilization (RS_Util)
7. IDQ Utilization (IDQ_Util)
8. Total Front-End Instruction Killed Latency (FE_Kill)

Aggregate Blocks	Mean Correlation across Benchmarks	Min Correlation across Benchmarks
Front End	0.86	0.80
Back End	0.90	0.81
Memory Order Buffer	0.93	0.92

NBTI Recovery Boosting

Negative Bias Temperature Instability

- Problem for the PMOS devices
- Increases V_t and hence the device delay

Negative Bias Temperature Instability

- Problem for the PMOS devices
- Increases V_t and hence the device delay
- **Stress phase:**
 - Negative bias ($V_{gs} = -V_{dd}$) at the gate of the PMOS
 - Leads to generation of interface traps

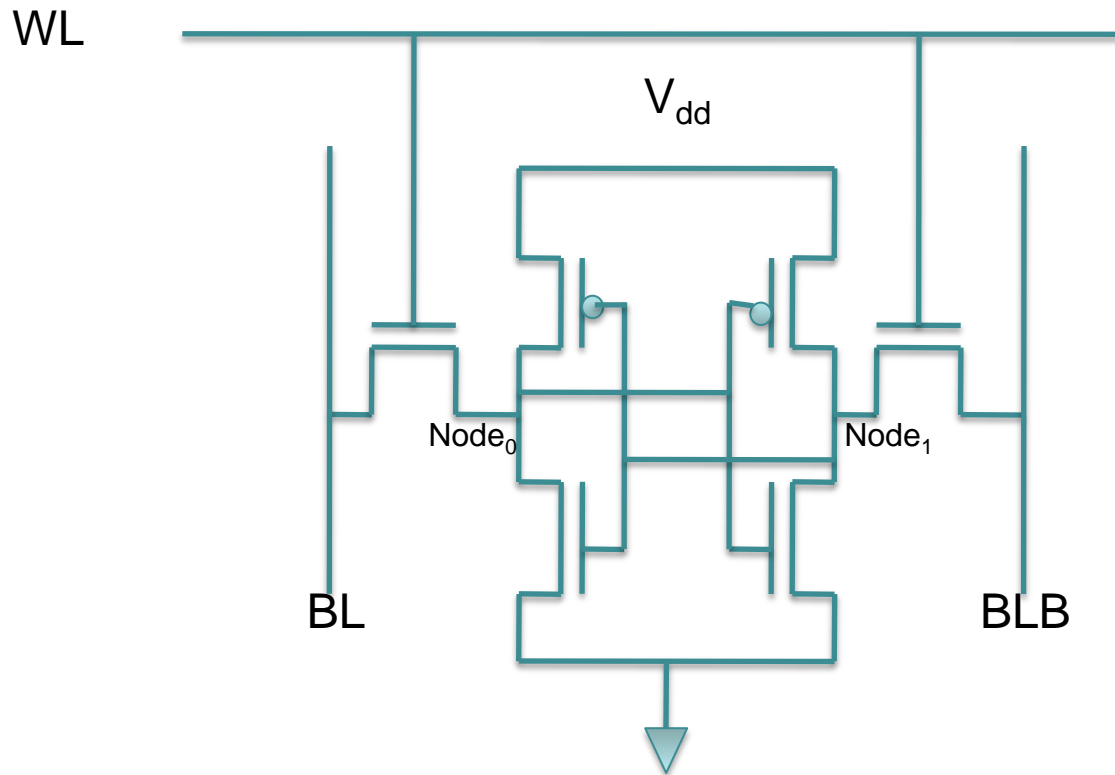
Negative Bias Temperature Instability

- Problem for the PMOS devices
- Increases V_t and hence the device delay
- **Stress phase:**
 - Negative bias ($V_{gs} = -V_{dd}$) at the gate of the PMOS
 - Leads to generation of interface traps
- **Recovery phase:**
 - No bias ($V_{gs} = 0$) at the gate of the PMOS
 - Eliminates some of the interface traps

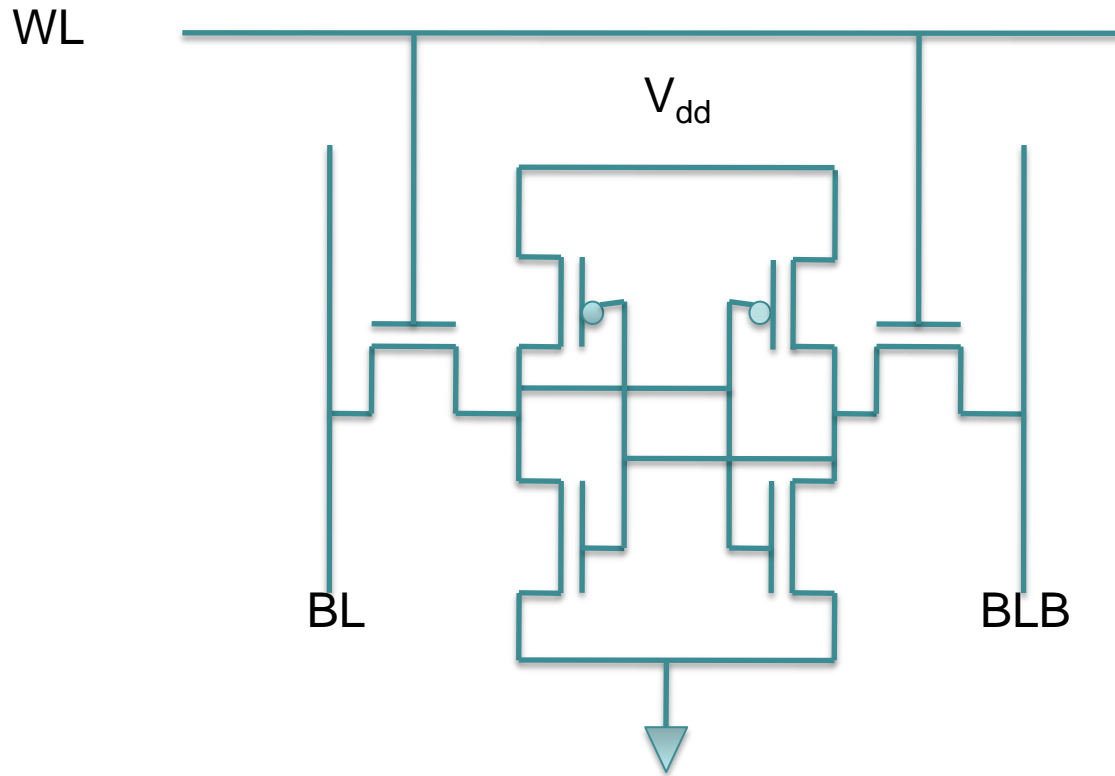
Negative Bias Temperature Instability

- Problem for the PMOS devices
- Increases V_t and hence the device delay
- **Stress phase:**
 - Negative bias ($V_{gs} = -V_{dd}$) at the gate of the PMOS
 - Leads to generation of interface traps
- **Recovery phase:**
 - No bias ($V_{gs} = 0$) at the gate of the PMOS
 - Eliminates some of the interface traps
- **Goal:** Enhance NBTI recovery for SRAM arrays

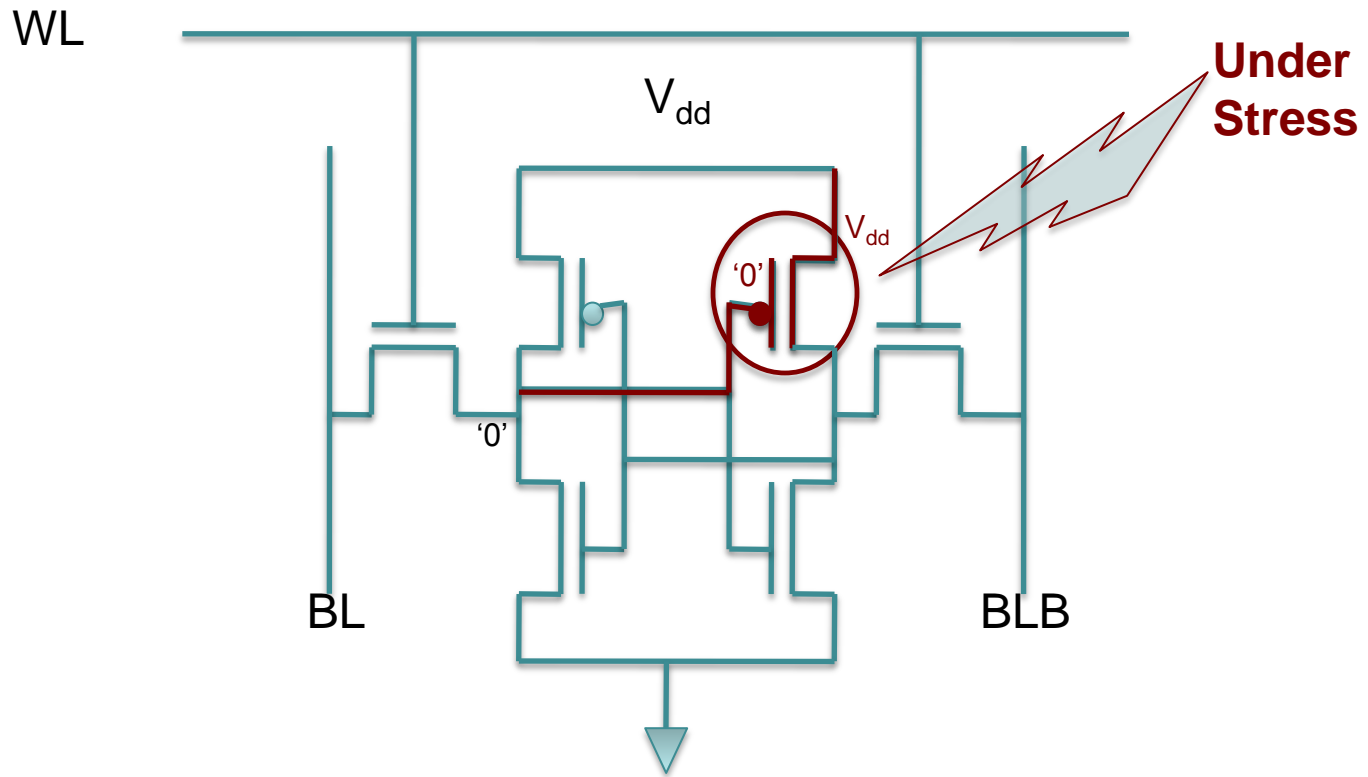
6T SRAM cell



Cell Holding a '0'



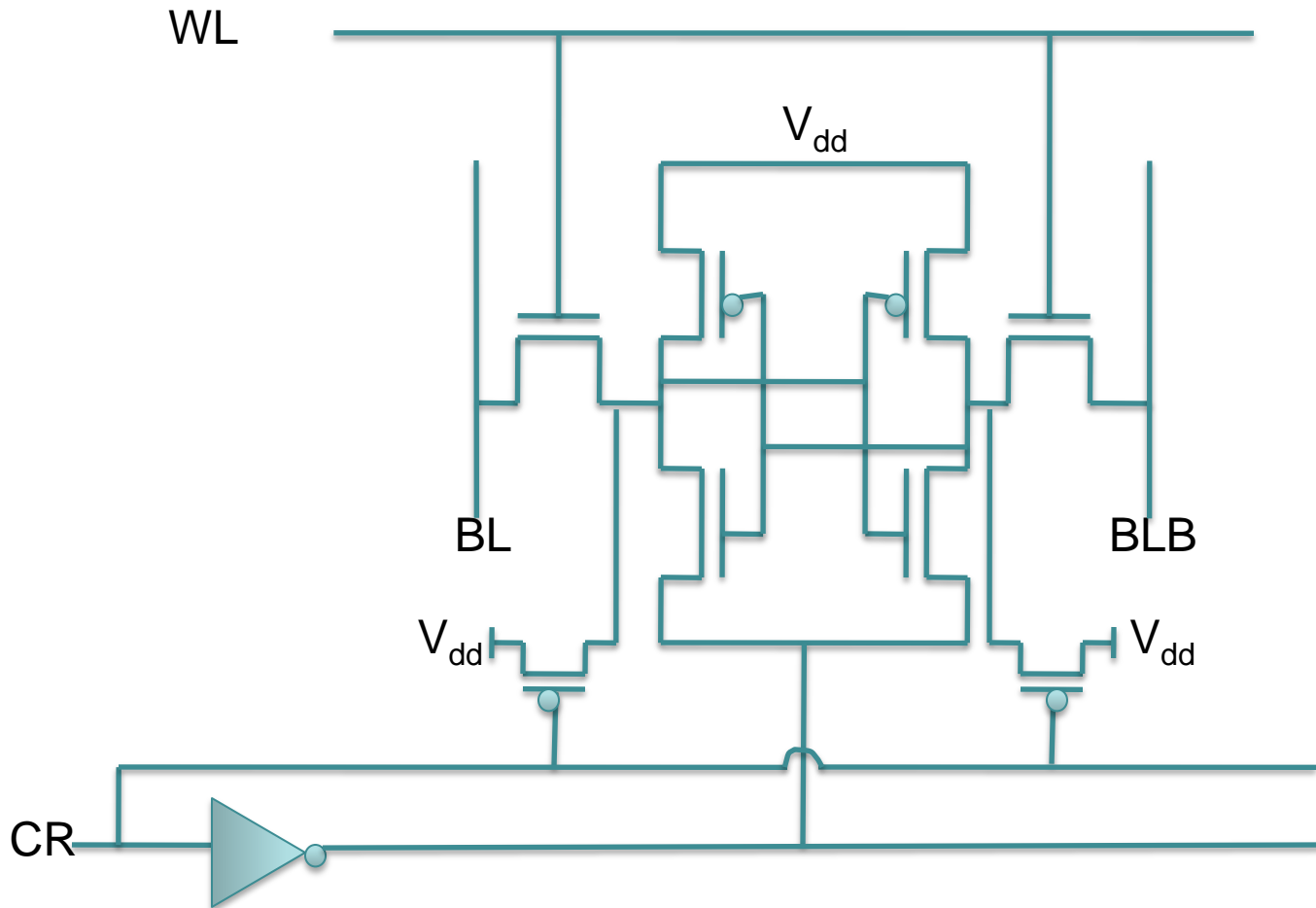
Cell Holding a '0'



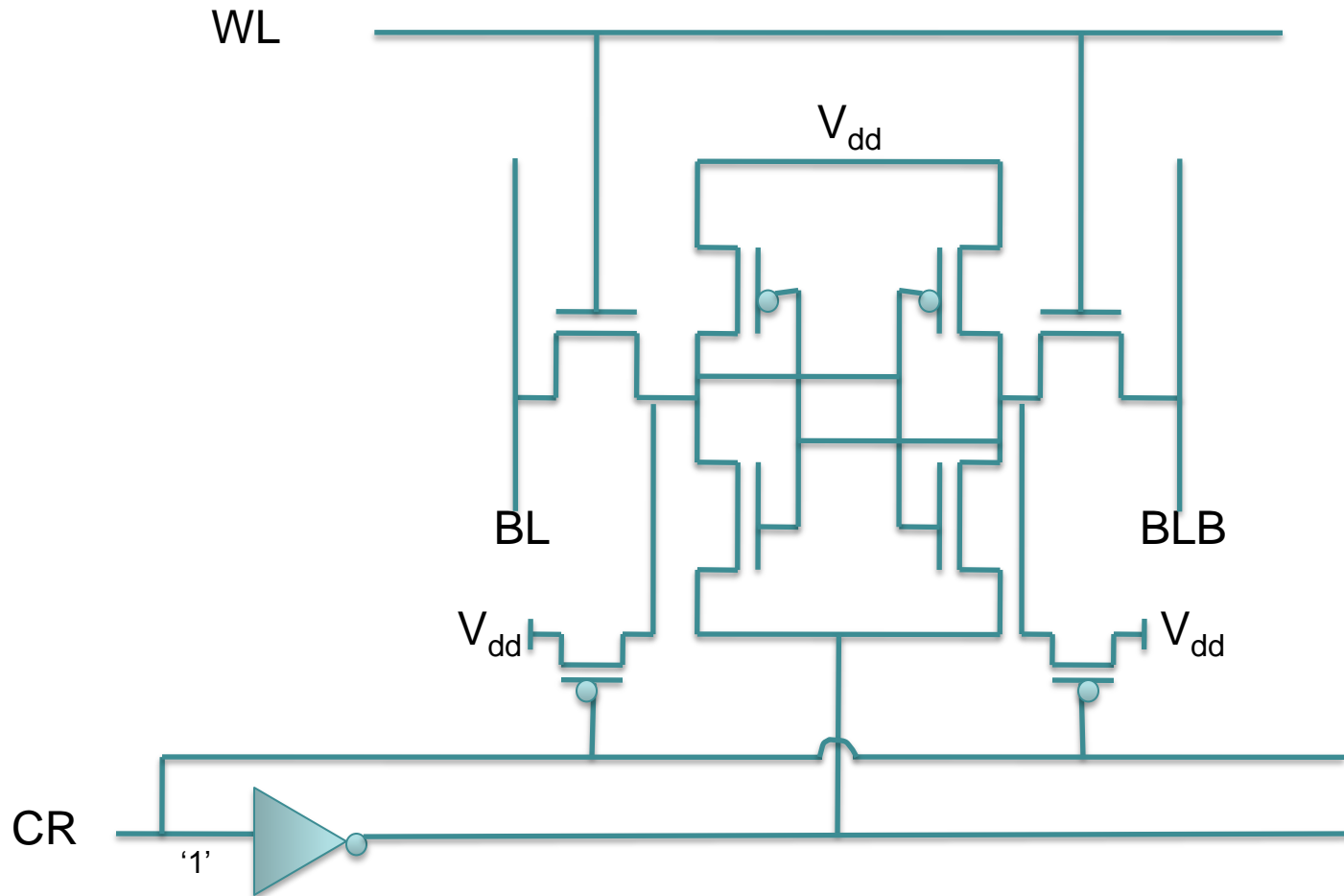
Approaches to NBTI Mitigation

- Stress reduction techniques:
 - **Power Gating**: Disconnect Vdd/GND connections
 - **Facelift**: Temperature-based job-scheduling with Vdd and Vt control [Tiwari et al., MICRO'08]
- Recovery enhancement techniques:
 - **Penelope**: Balance the degradation of the two PMOS devices in the cell [Abella et al., MICRO'07]

Recovery Boosting

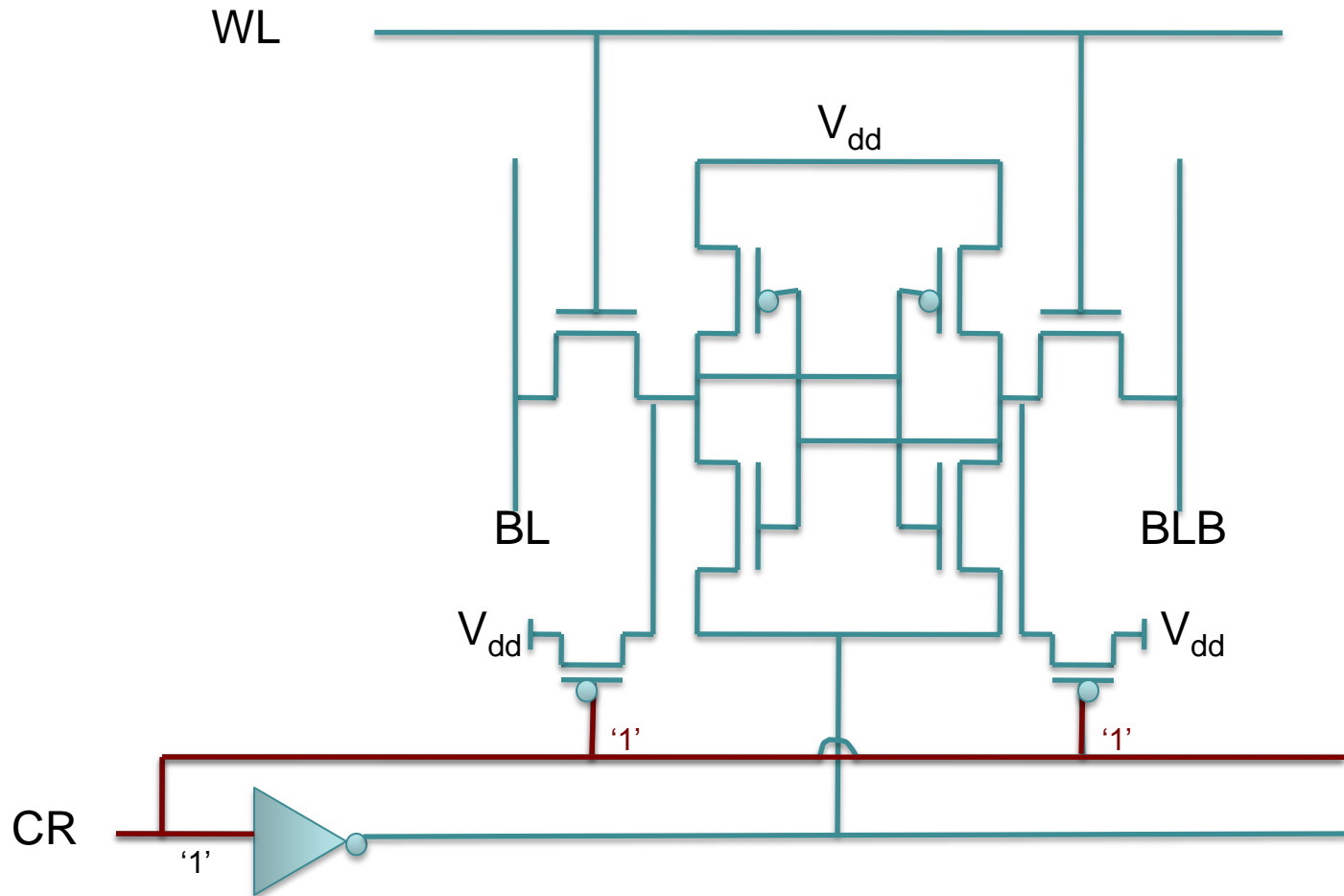


Recovery Boosting



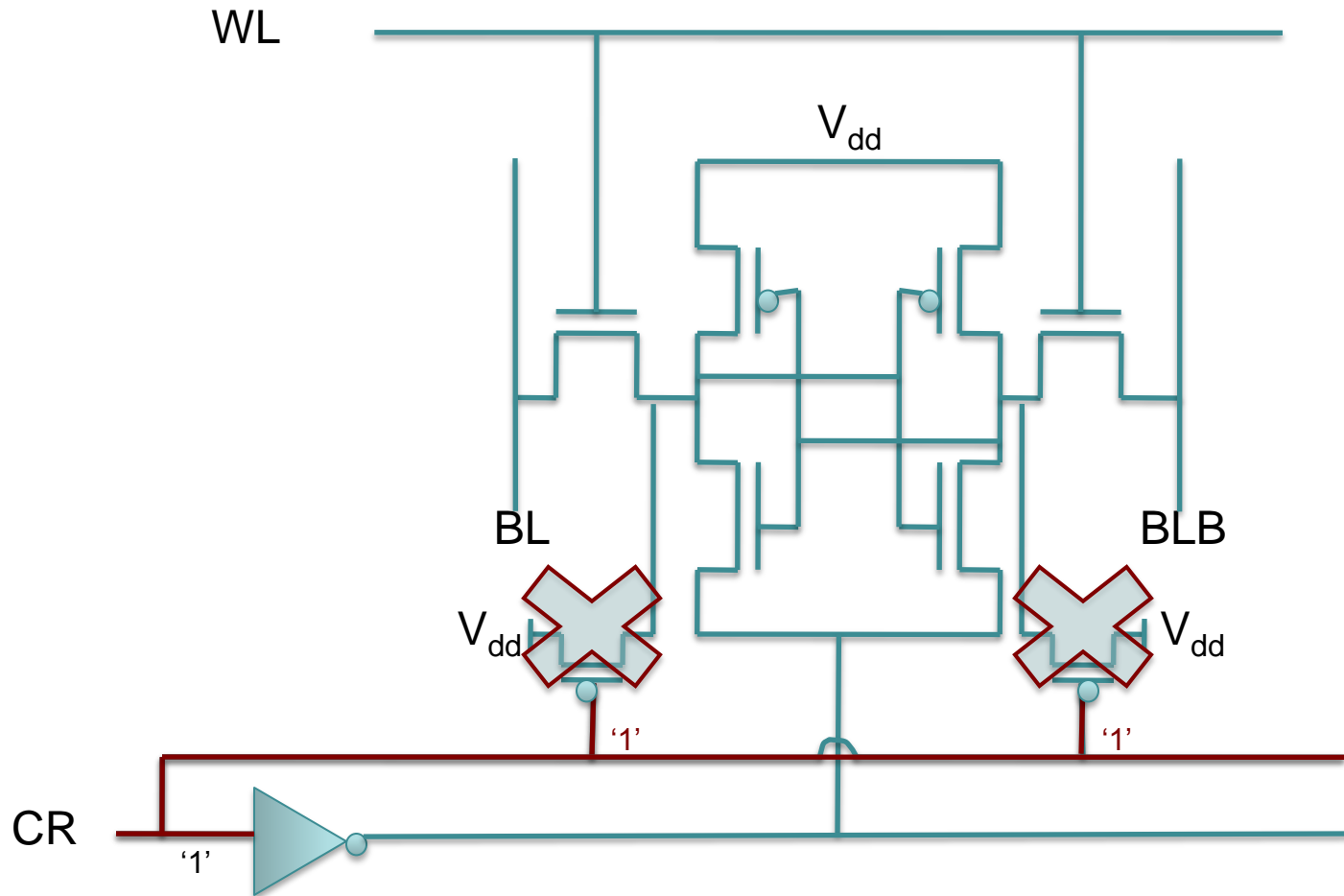
Valid Data:
CR = 1

Recovery Boosting



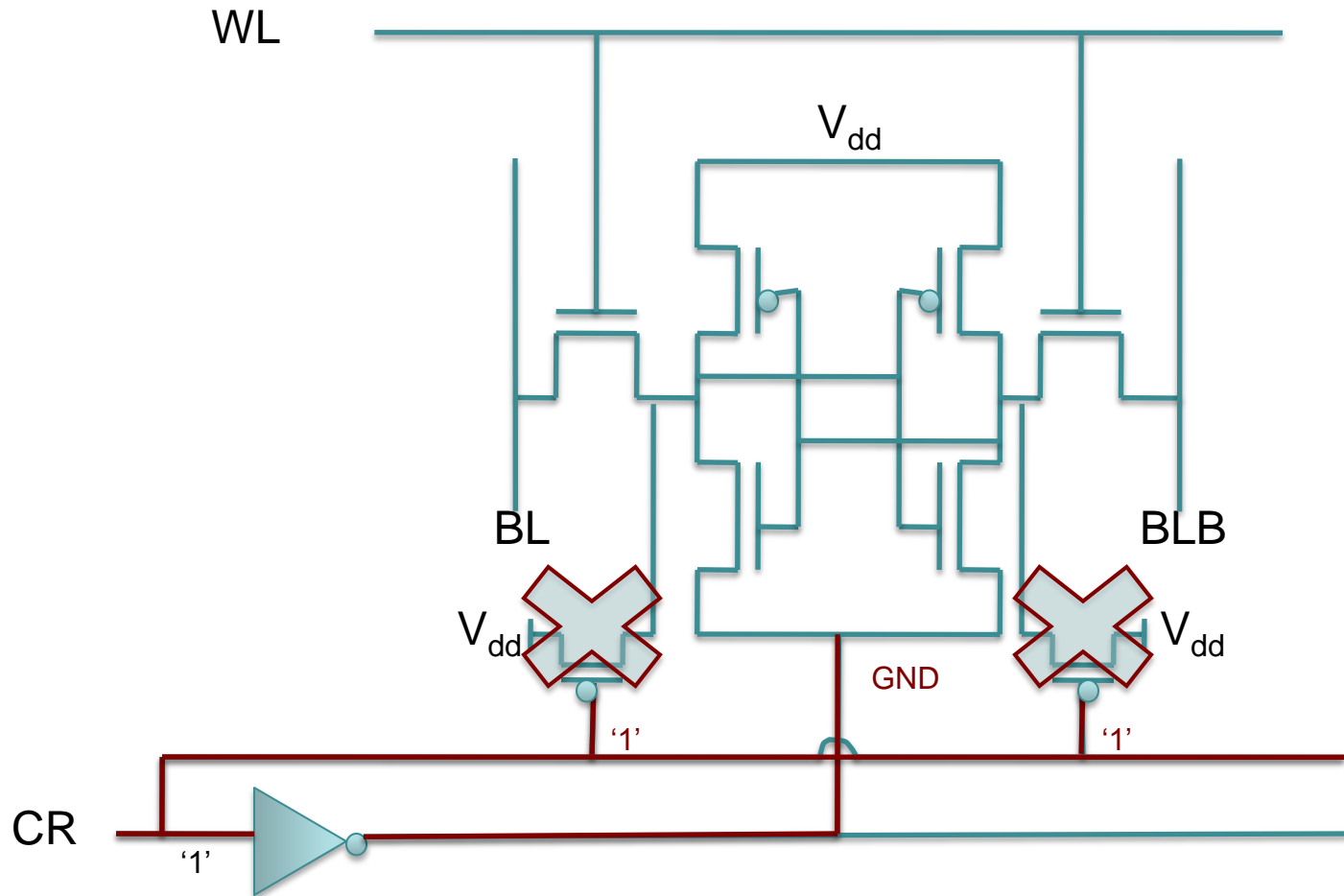
Valid Data:
CR = 1

Recovery Boosting



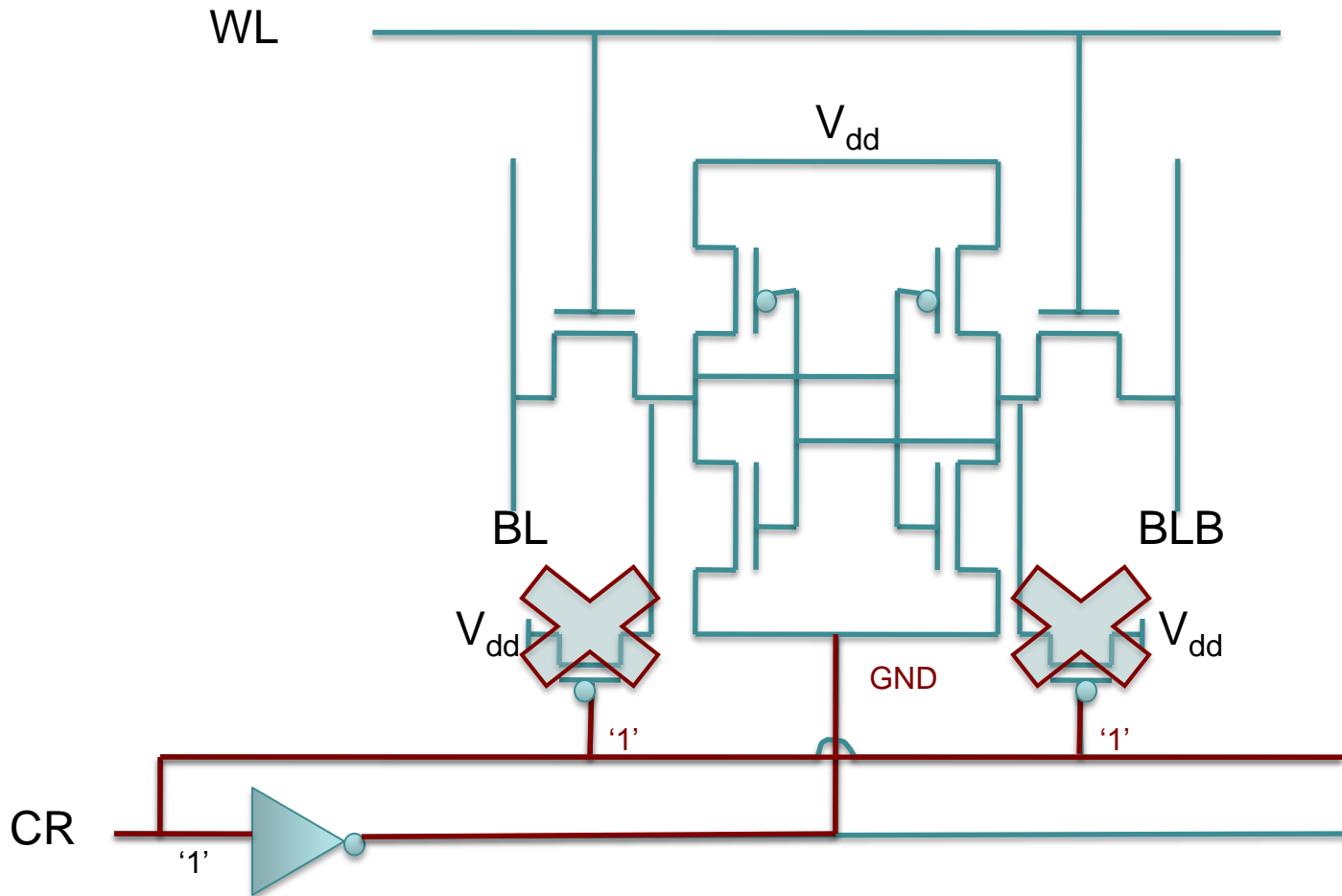
Valid Data:
CR = 1

Recovery Boosting



Valid Data:
CR = 1

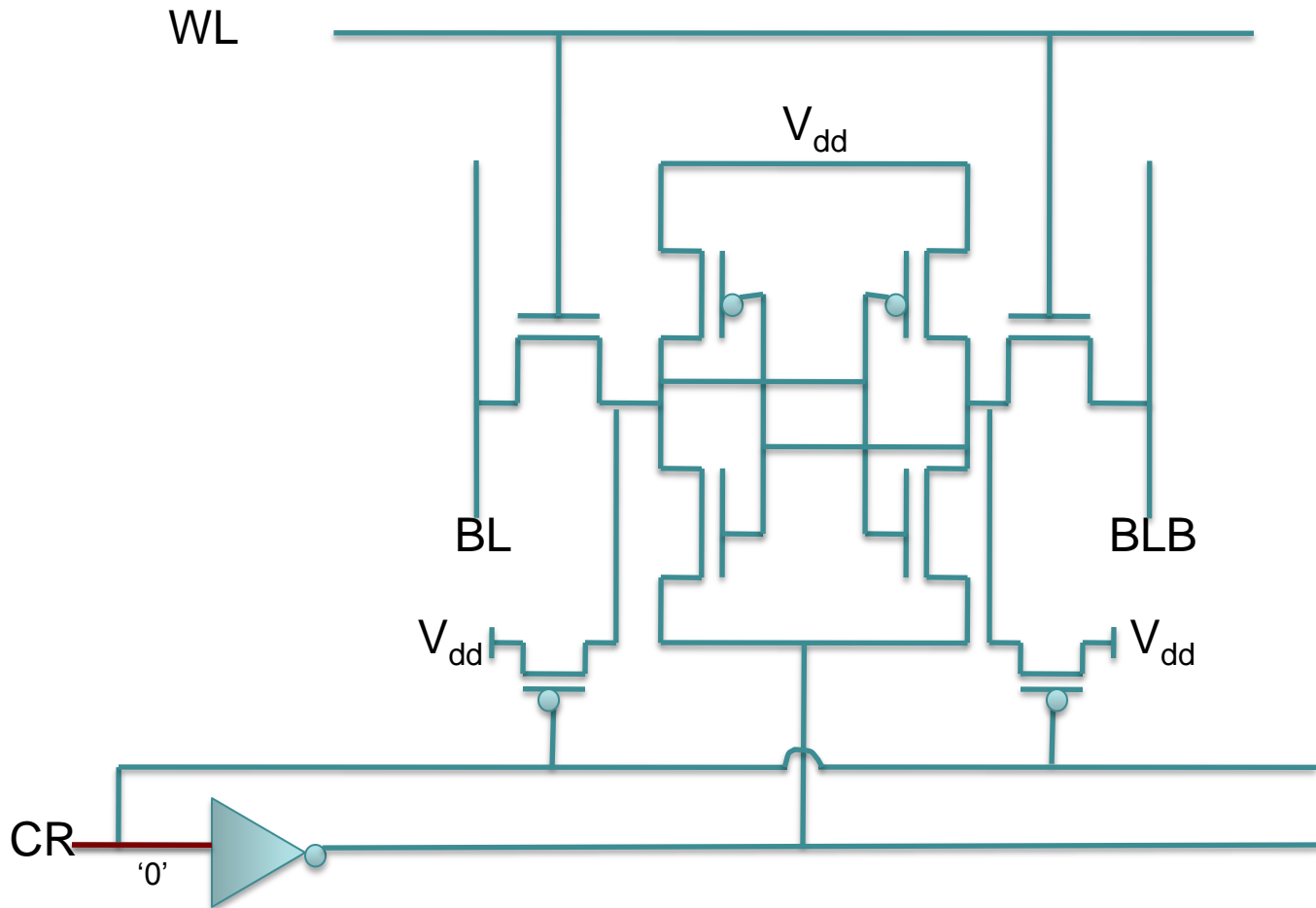
Recovery Boosting



Valid Data:
CR = 1

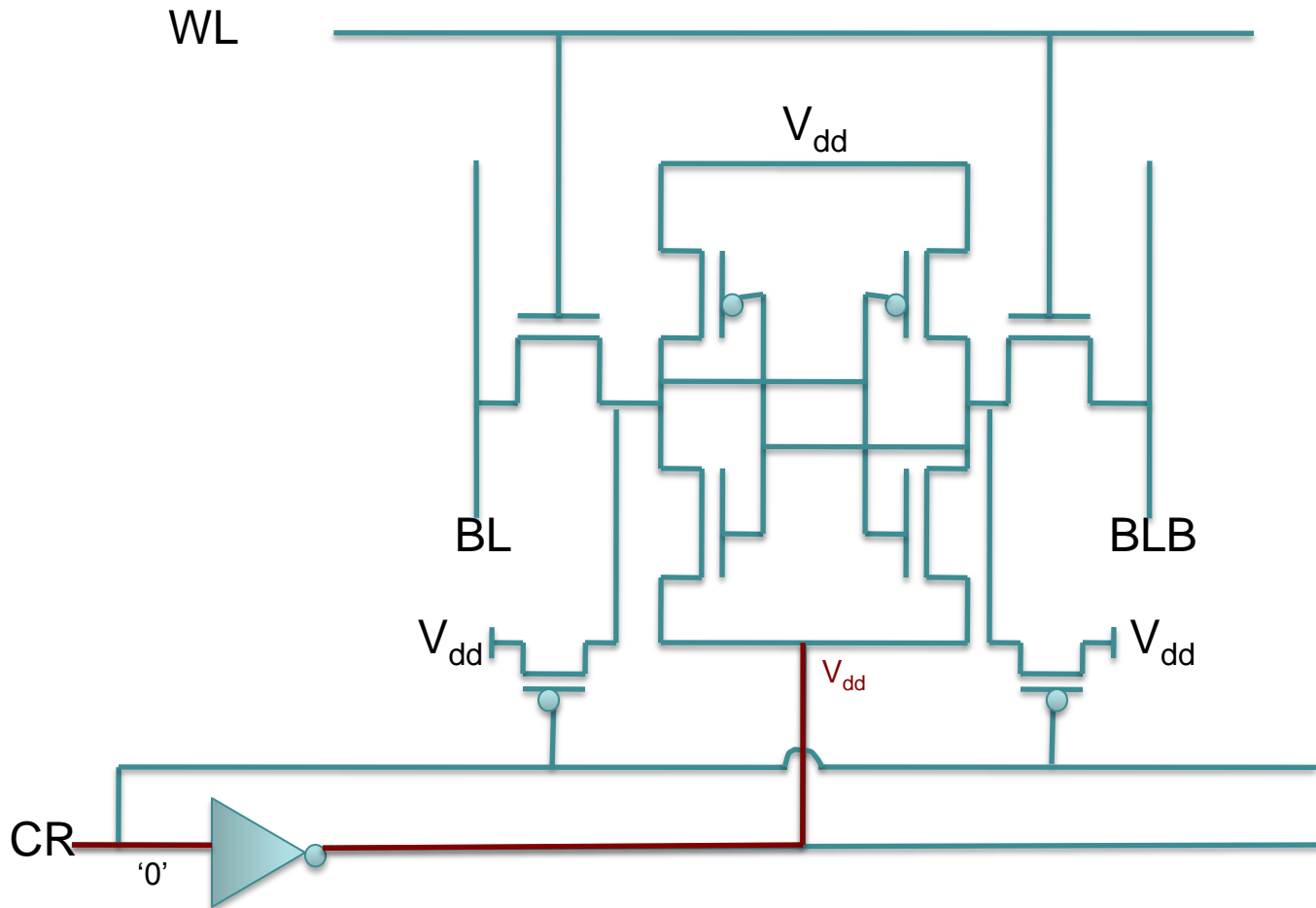
Normal
6T cell

Recovery Boosting



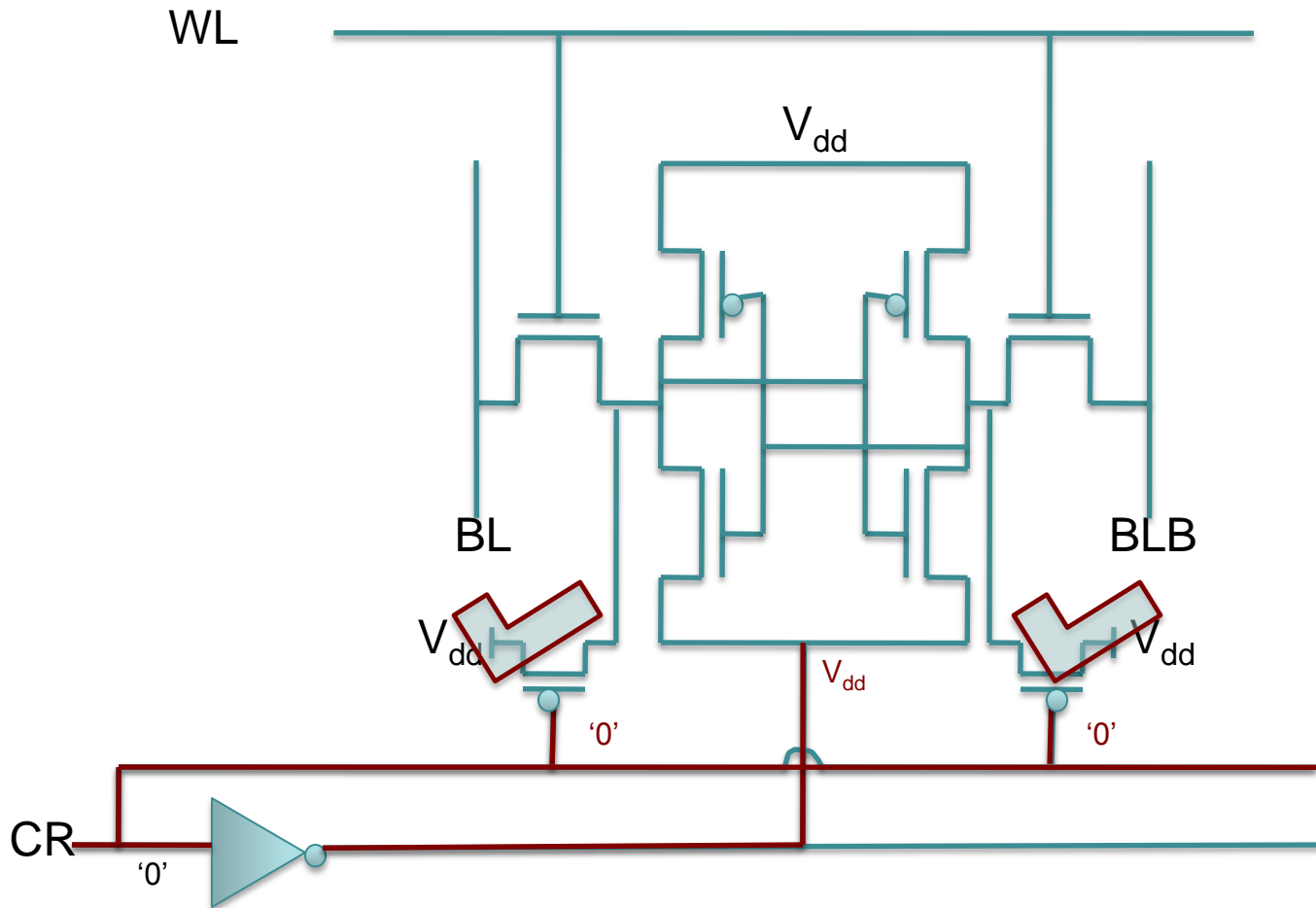
Valid Data:
CR = 0

Recovery Boosting



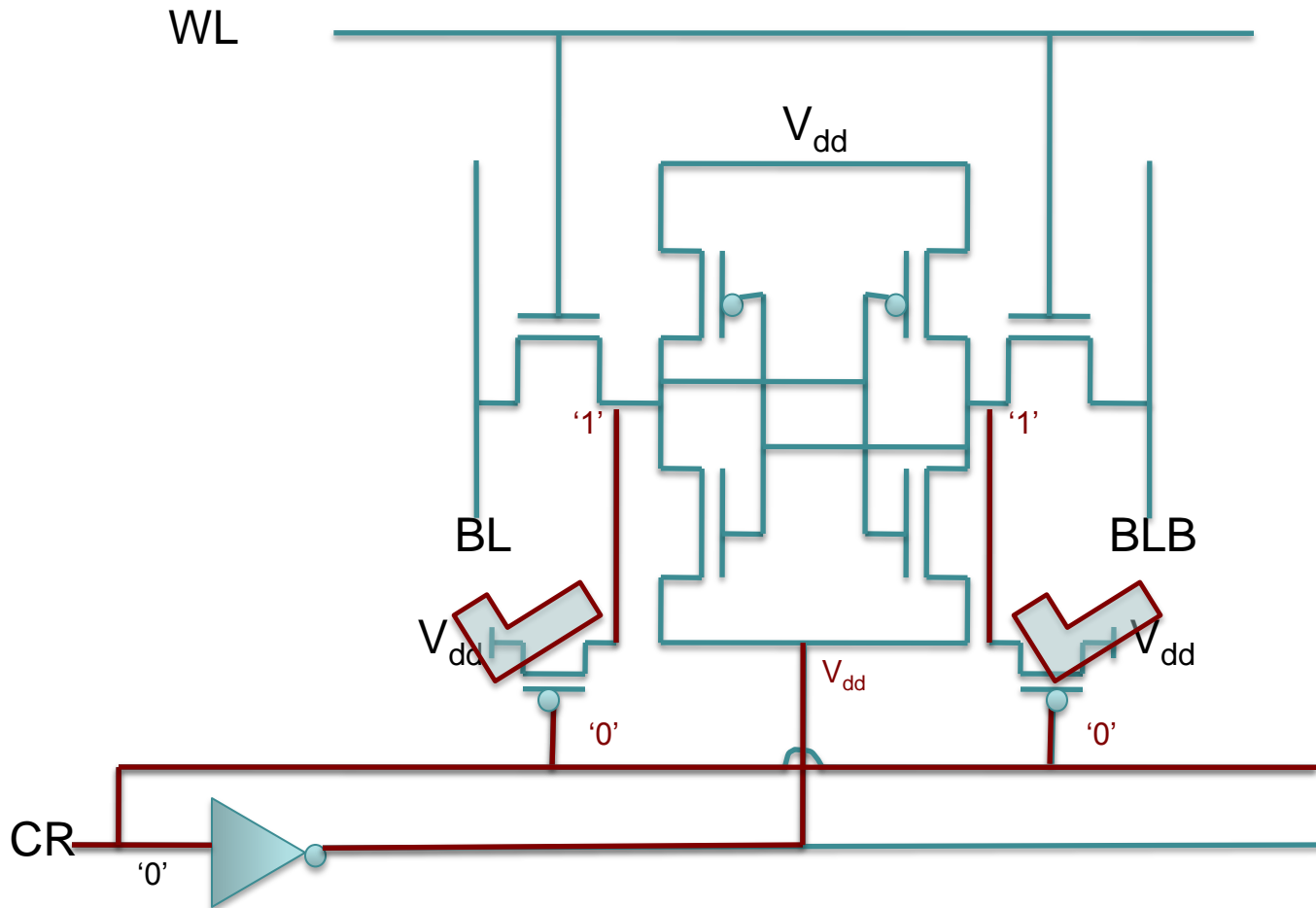
Valid Data:
CR = 0

Recovery Boosting



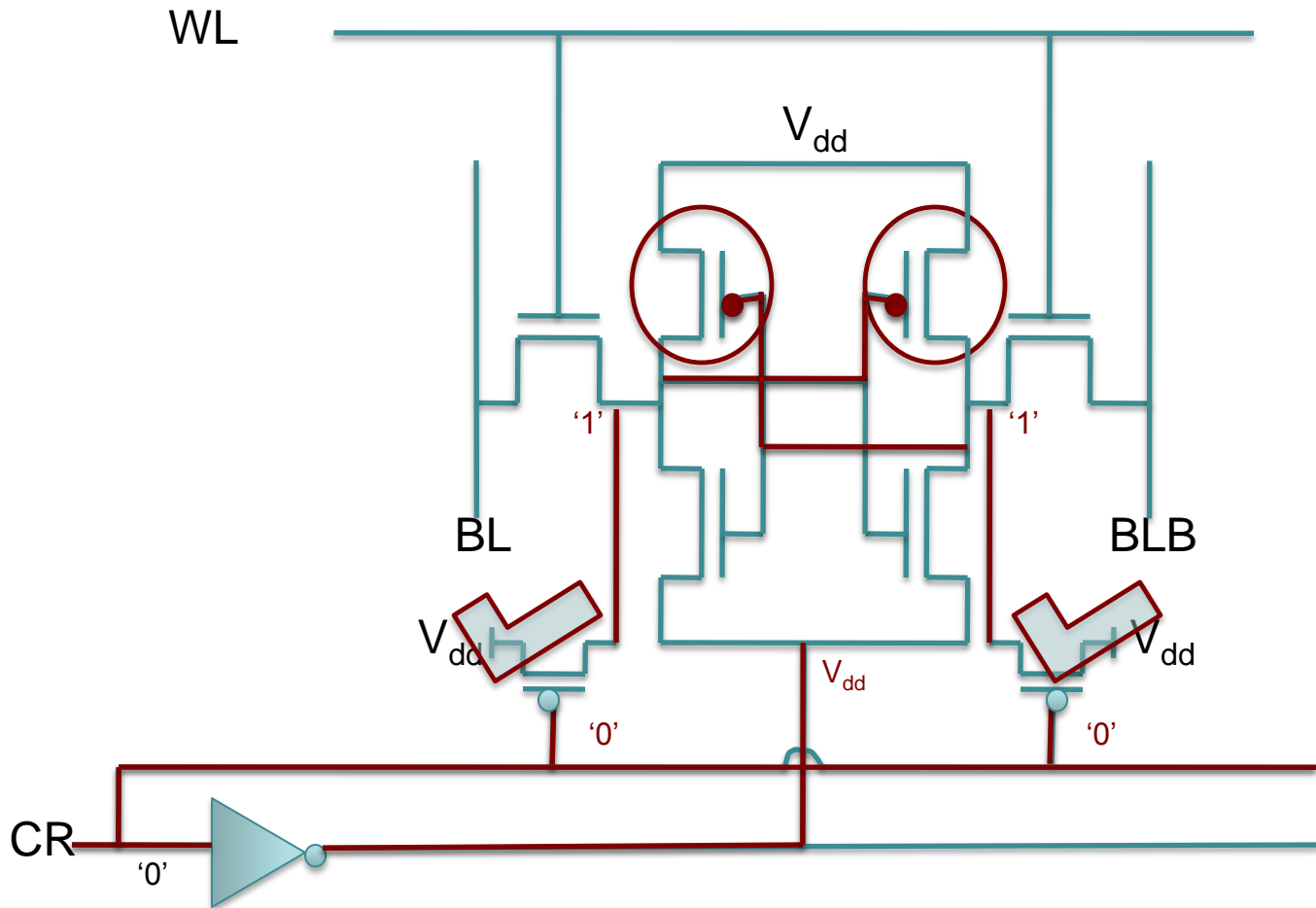
Valid Data:
CR = 0

Recovery Boosting



Valid Data:
CR = 0

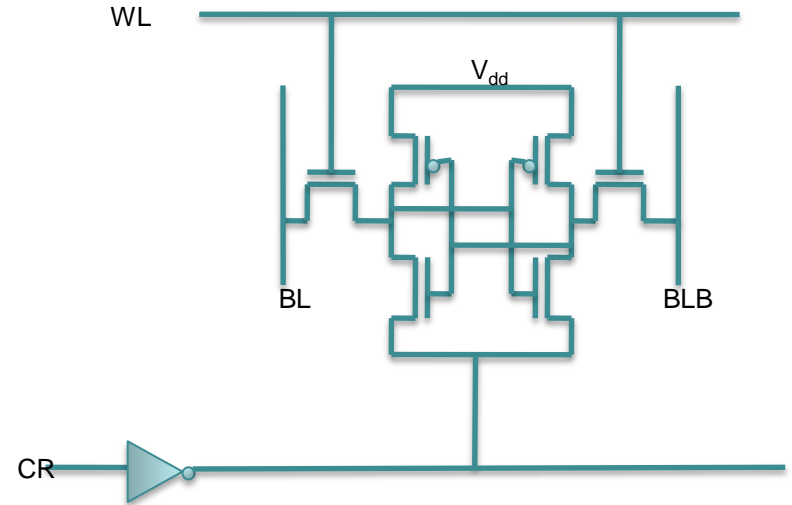
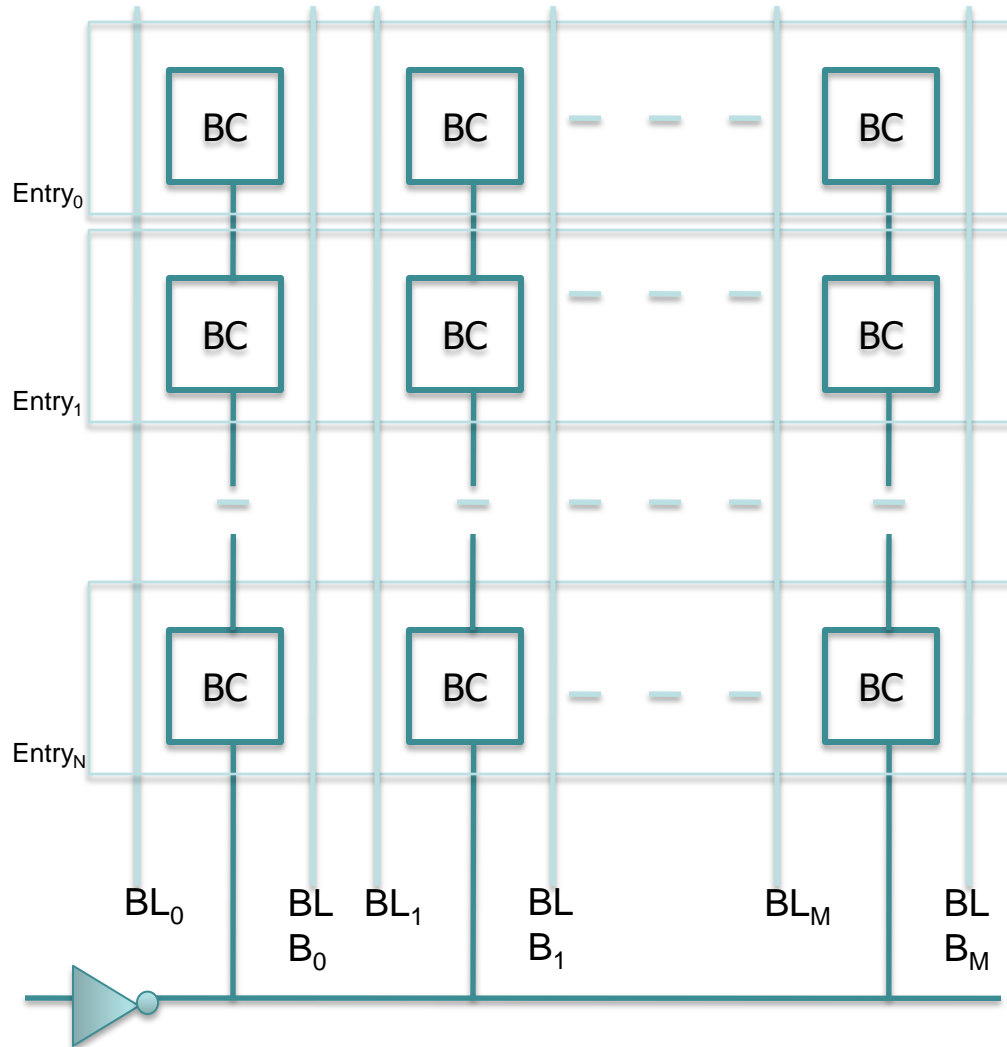
Recovery Boosting



Valid Data:
CR = 0

Both
PMOS
undergo
recovery

Coarse-Grained Recovery Boosting

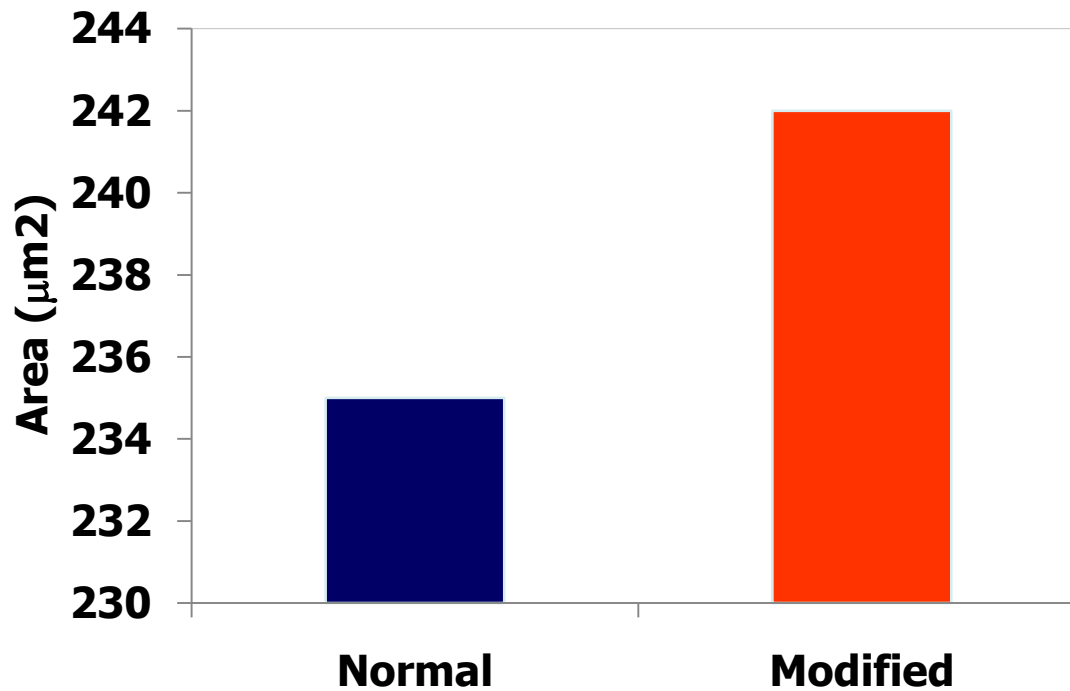


Example Design

- Issue Queue (ISQ)
 - 64-entry Non-collapsing ISQ with 4 read and 4 write ports [Folegnani and Gonzalez, ISCA'01]
 - Both CAM and RAM use modified bitcells
 - ISQ entries with invalid data put into recovery boost mode
- SPICE simulations (Cadence Spectre) + Architecture simulations (M5 and SPEC CPU2000 benchmarks) for 32nm process

SPICE Results

Area Overhead of Issue Queue

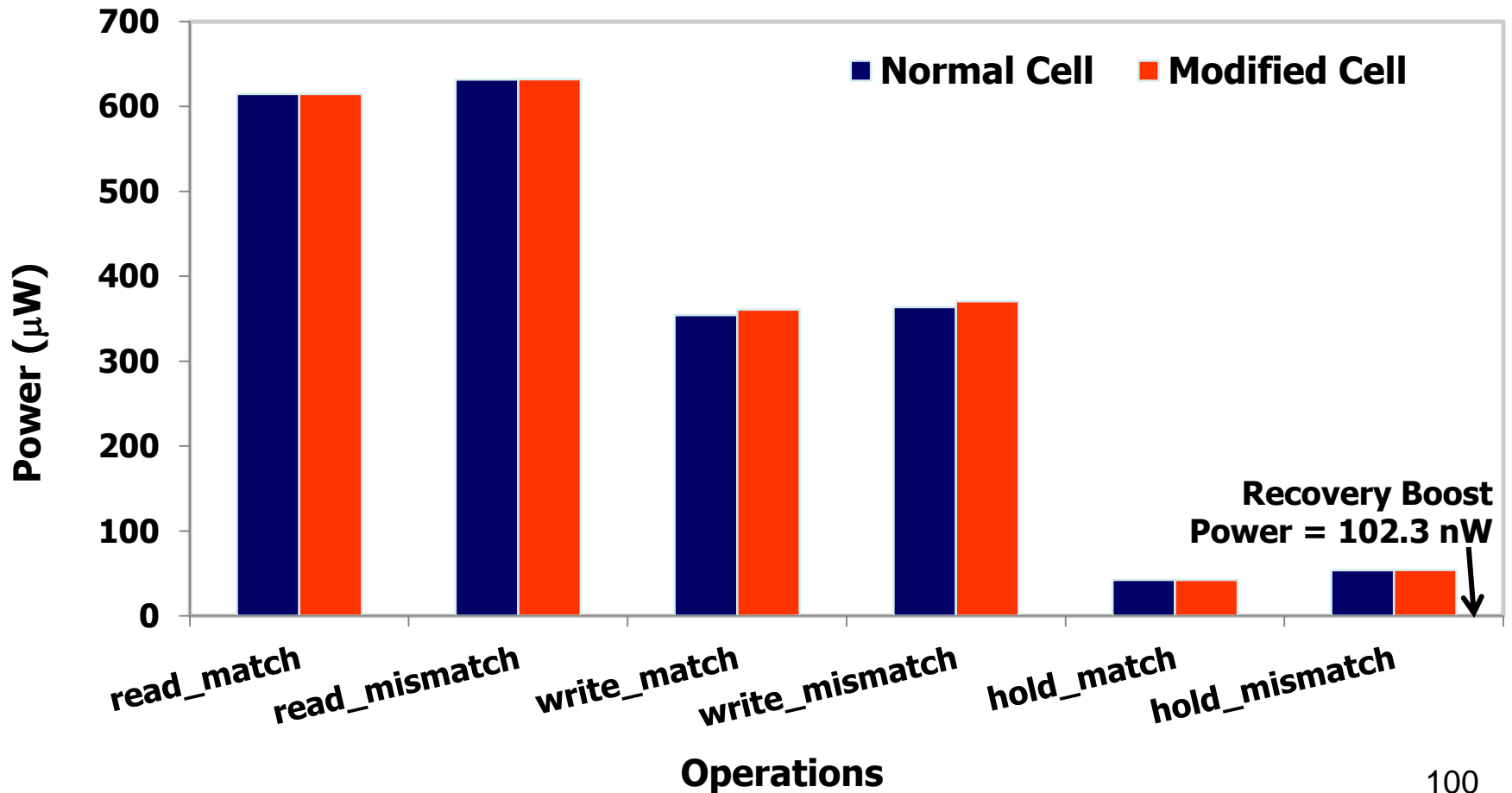


Area overhead = 3%

Overhead is small since the cell is highly multiported

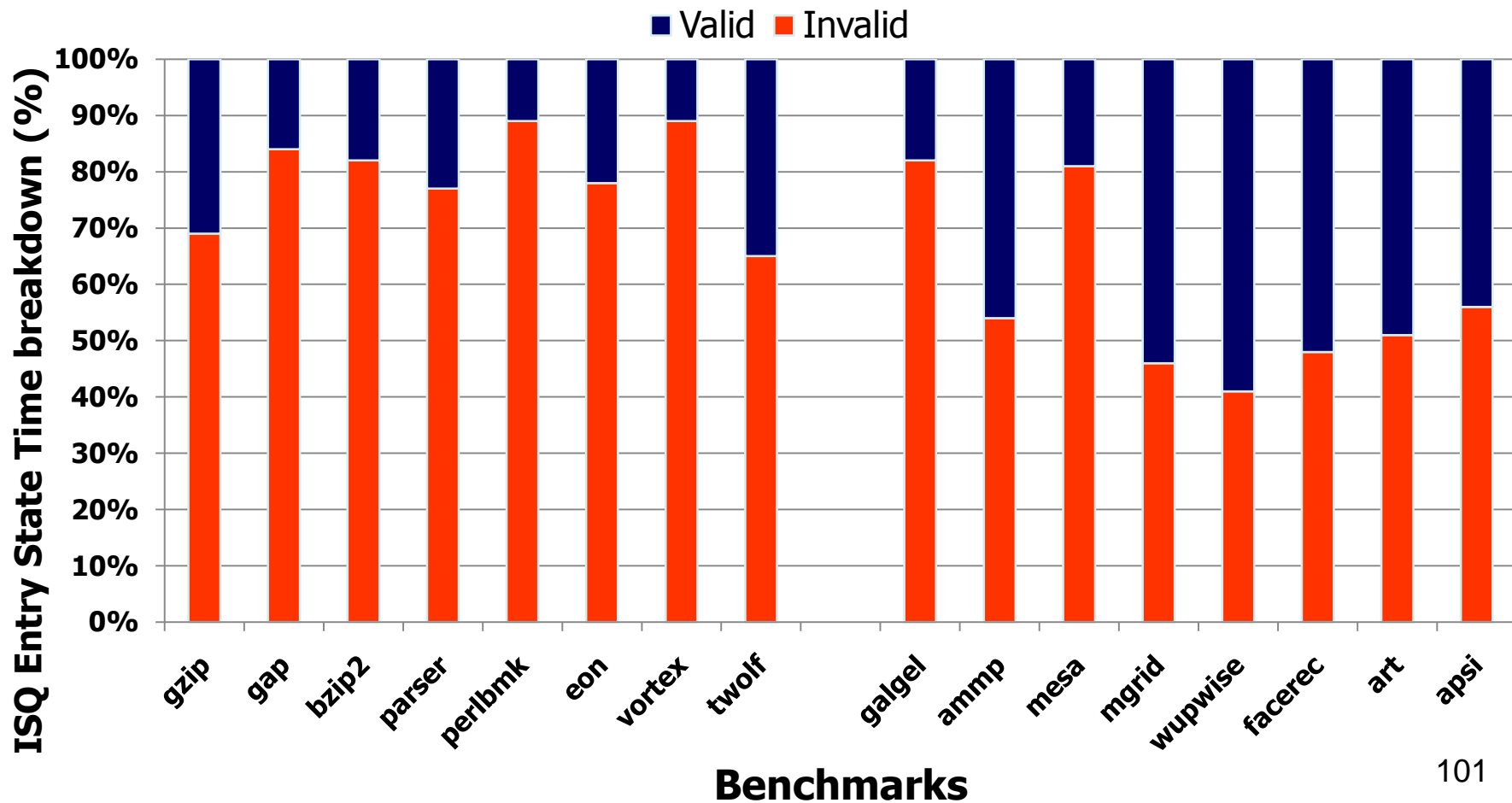
SPICE Results

Issue Queue Entry Power Consumption



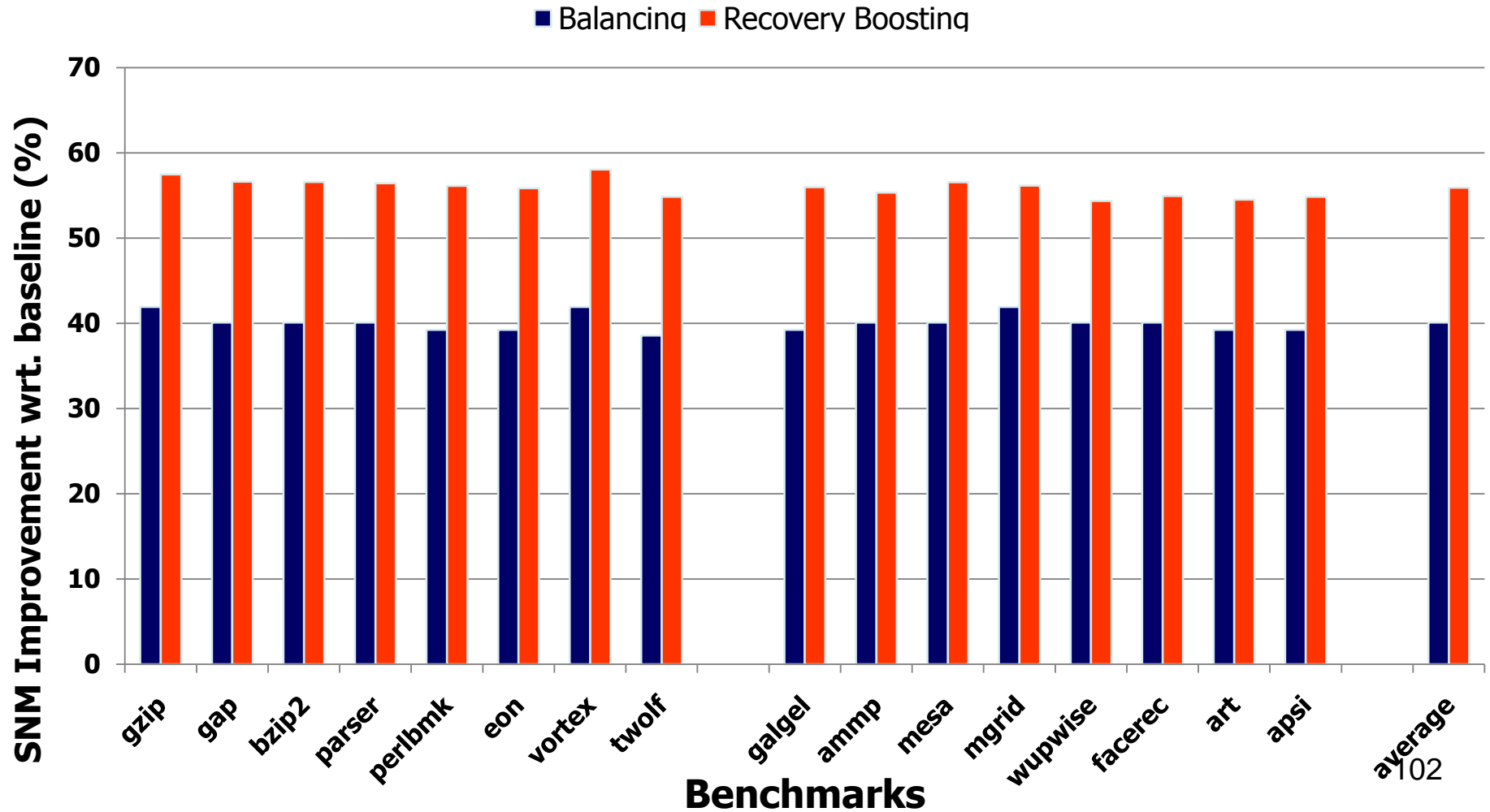
ISQ Entry State Time

Area Neutral wrt. Baseline – 2 Fewer ISQ entries



ISQ SNM Improvement

Initial $V_t=0.2$ V, Service Life=7 years



Conclusions

- Error protection imposes performance, power, and area overheads
- Adapt error protection to the protection needs at runtime
 - Partial RMT (Slick)
 - Runtime AVF Prediction
- Recovery boosting enhances NBTI recovery with little performance, power, or area overheads

Thank You

<http://www.cs.virginia.edu/~gurumurthi>