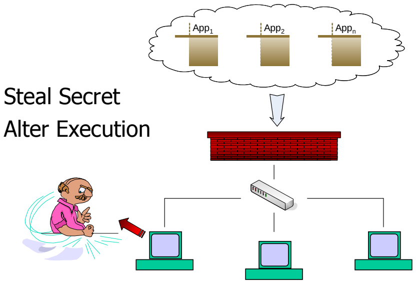
A Low-Cost Memory Remapping Scheme for Address Bus Protection

Lan Gao*, Jun Yang§, Marek Chrobak*,
Youtao Zhang§, San Nguyen*, Hsien-Hsin S. Lee¶

*University of California, Riverside
§ University of Pittsburgh
¶Georgia Institute of Technology

Security Breaches
--- from another way around

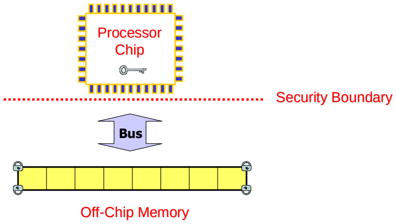
- ❖ Steal Secret
- ❖ Alter Execution



Outline

- ❖ Background & Motivation
  - ➢ Secure Processor Model
  - ➢ Address Information Leakage
- ❖ Previous Address Bus Protection Solutions
  - ➢ The HIDE Scheme
  - ➢ The Shuffle Scheme
- ❖ Our Low-Cost Address Permutation Scheme
- ❖ Performance Evaluation
- ❖ Conclusion

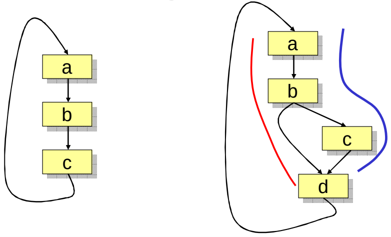
Secure Processor Model



[D. Lie et al. ASPLOS 2000, G. Edward Suh et al. MICRO 36, J. Yang et al. MICRO 36]

Address Information Leakage

[X. Zhuang, ASPLOS 2004]



Address Sequence	abc abc abc …	abcd abd abcd abd …
CFG Hint	Loop	Conditional Branch

Oblivious Memory Access

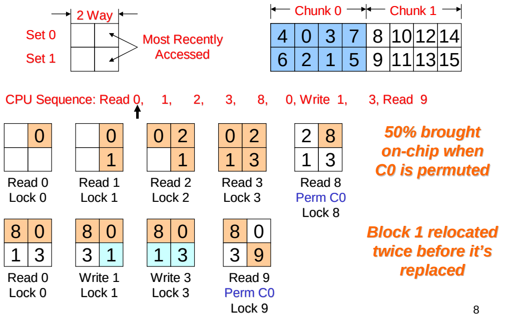
- ❖ The idea: [Oded Goldreich et al.]
  - ➢ Replace each memory access by a sequence of redundant accesses
  - ➢ Satisfactory from a theoretical perspective
- ❖ Overhead:

	"naive"	"square root"	"hierarchical"
Memory	m	$m+2\sqrt{m}$	$\mathrm{O}(t\cdot\log^2 t)$
Runtime	$t\cdot m$	$\mathrm{O}(t\cdot\sqrt{m})$	$\mathrm{O}(t\cdot\log^3 t)$

The HIDE Cache

- The Idea: break the correlation between repeated addresses [Xiaotong Zhuang et al. ASPLOS 2004]
  - Permute the address space at suitable intervals
  - Permute blocks within a “chunk”
- How: Lock and Permute
  - Lock a block in the cache
    - A new read from memory
    - A dirty block since last permutation
  - Permute a chunk when replacing a locked block

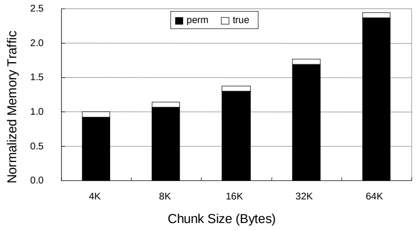
HIDE Cache: An Example



50% brought on-chip when C0 is permuted

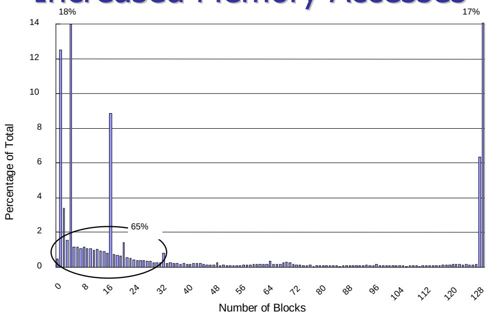
Block 1 relocated twice before it’s replaced

Increased Memory Accesses



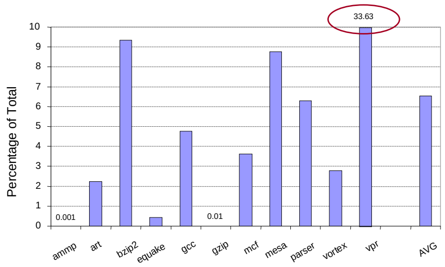
Breakdown of memory traffic for different chunk sizes

Increased Memory Accesses



Histogram of pages with 0-128 blocks accessed between permutations

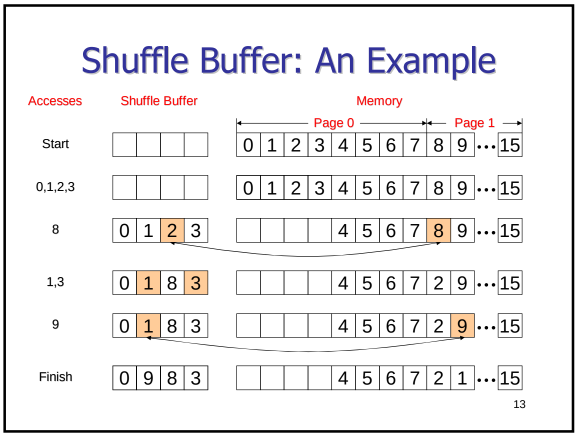
Redundant Permutations



Write Lock Induced Permutations

The Shuffle Buffer

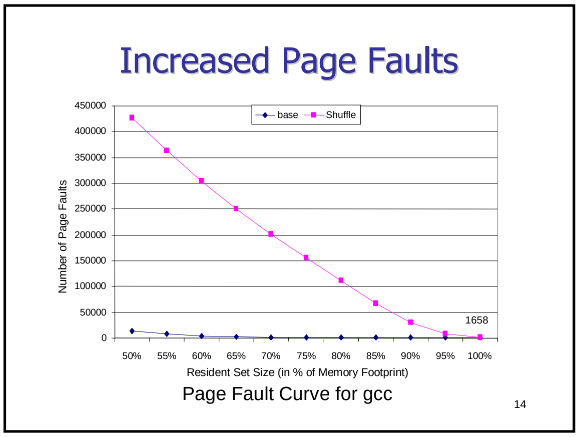
- The Idea: dynamic control flow obfuscation [X. Zhuang et al., CASES 2004]
  - Relocate a block if they appeared on the bus once
- How: Random Swap
  - Any newly read block is inserted into a shuffle buffer
  - A buffered block is written back to the address of the newly read block
  - Only read/write access pairs are observed on the address bus



Shuffle Buffer: An Example

Accesses	Shuffle Buffer	Memory (Page 0: 0–7, Page 1: 8–15)
Start	□ □ □ □	0 1 2 3 4 5 6 7 8 9 •••15
0,1,2,3	□ □ □ □	0 1 2 3 4 5 6 7 8 9 •••15
8	0 1 2 3	□ □ □ □ 4 5 6 7 8 9 •••15
1,3	0 1 8 3	□ □ □ □ 4 5 6 7 2 9 •••15
9	0 1 8 3	□ □ □ □ 4 5 6 7 2 9 •••15
Finish	0 9 8 3	□ □ □ □ 4 5 6 7 2 1 •••15

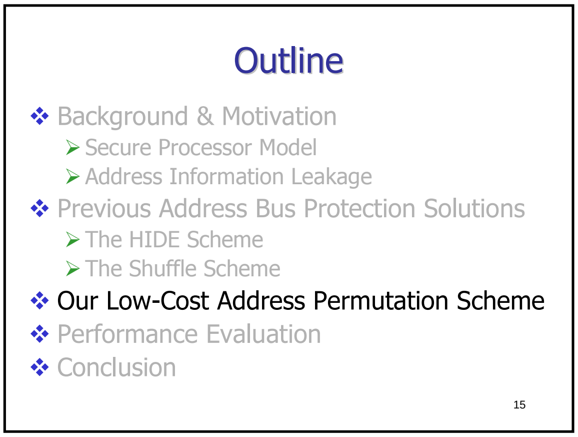
13



Increased Page Faults

Page Fault Curve for gcc

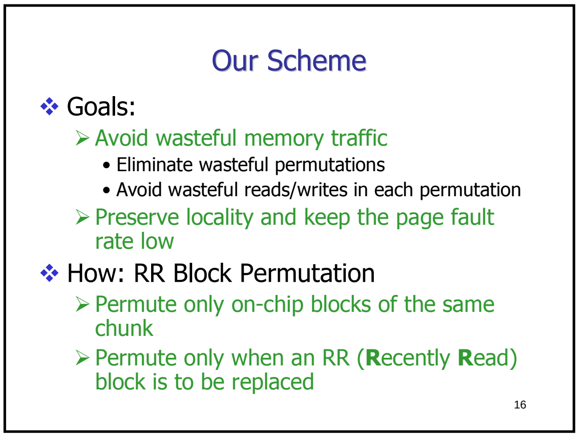
14



Outline

- Background & Motivation
  - Secure Processor Model
  - Address Information Leakage
- Previous Address Bus Protection Solutions
  - The HIDE Scheme
  - The Shuffle Scheme
- Our Low-Cost Address Permutation Scheme
- Performance Evaluation
- Conclusion

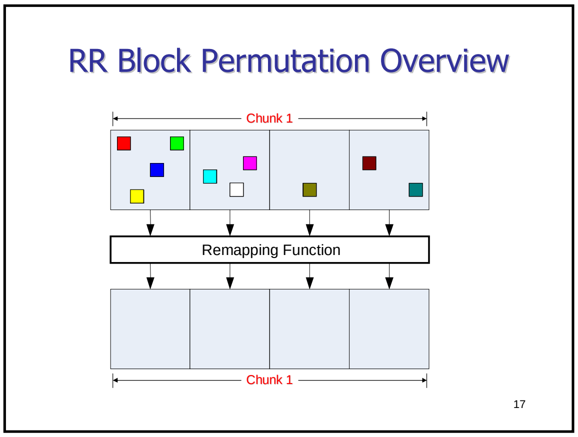
15



Our Scheme

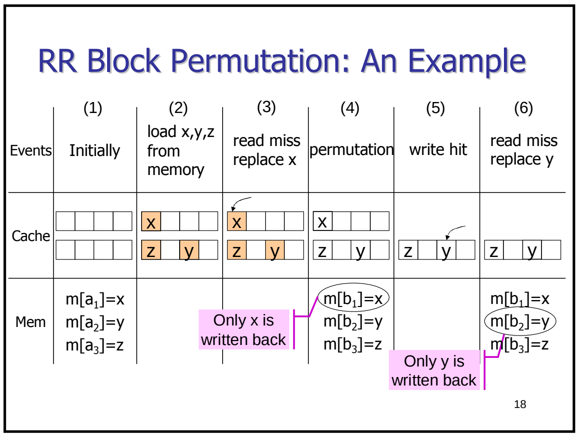
- Goals:
  - Avoid wasteful memory traffic
    - Eliminate wasteful permutations
    - Avoid wasteful reads/writes in each permutation
  - Preserve locality and keep the page fault rate low
- How: RR Block Permutation
  - Permute only on-chip blocks of the same chunk
  - Permute only when an RR (**R**ecently **R**ead) block is to be replaced

16



RR Block Permutation Overview

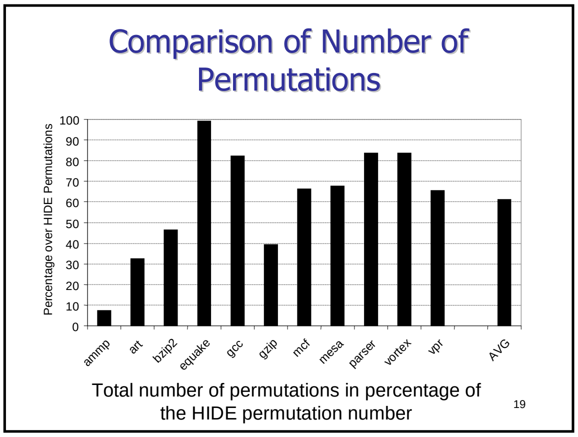
17



RR Block Permutation: An Example

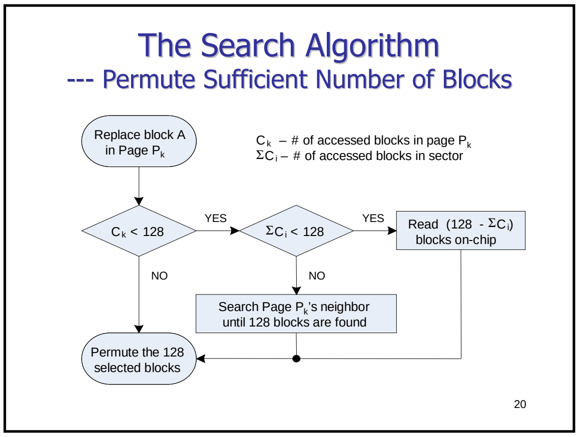
	(1)	(2)	(3)	(4)	(5)	(6)
Events	Initially	load x,y,z from memory	read miss replace x	permutation	write hit	read miss replace y
Cache	□ □ □ □ □ □ □ □	x □ □ □ z □ y □	x □ □ □ z □ y □	x □ □ □ z □ y □	z □ y □	z □ y □
Mem	$m[a_1]=x$ $m[a_2]=y$ $m[a_3]=z$		Only x is written back	$m[b_1]=x$ $m[b_2]=y$ $m[b_3]=z$	Only y is written back	$m[b_1]=x$ $m[b_2]=y$ $m[b_3]=z$

18



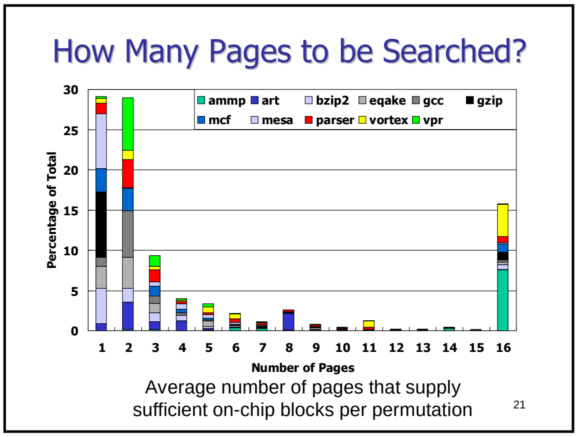
Comparison of Number of Permutations

Total number of permutations in percentage of the HIDE permutation number



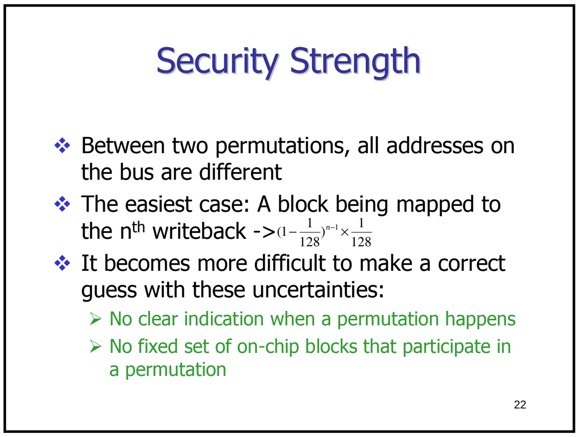
The Search Algorithm
--- Permute Sufficient Number of Blocks

Replace block A in Page P_k
C_k – # of accessed blocks in page P_k
ΣC_i – # of accessed blocks in sector
$C_k < 128$
YES
$\Sigma C_i < 128$
YES
Read $(128 - \Sigma C_i)$ blocks on-chip
NO
NO
Search Page P_k's neighbor until 128 blocks are found
Permute the 128 selected blocks



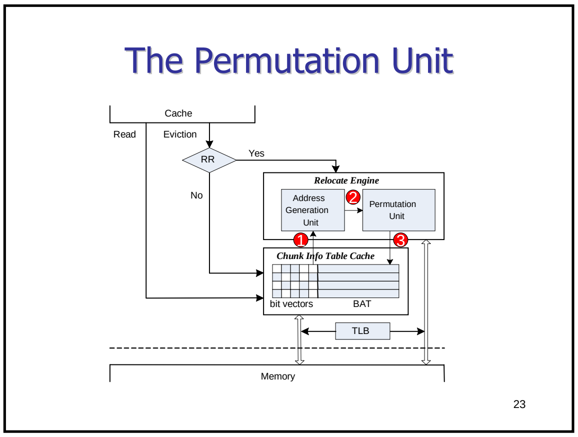
How Many Pages to be Searched?

Average number of pages that supply sufficient on-chip blocks per permutation

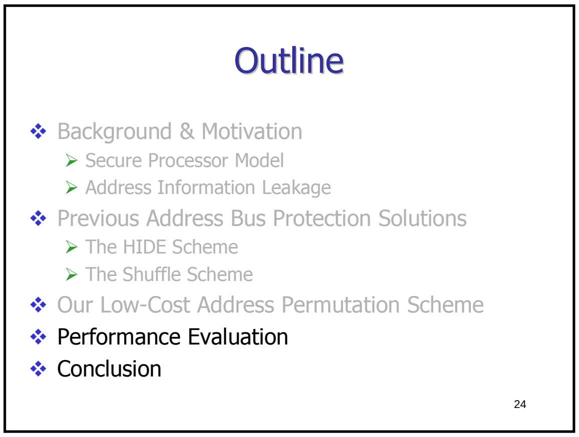


Security Strength

- Between two permutations, all addresses on the bus are different
- The easiest case: A block being mapped to the nth writeback ->$(1-\frac{1}{128})^{n-1}\times\frac{1}{128}$
- It becomes more difficult to make a correct guess with these uncertainties:
  - No clear indication when a permutation happens
  - No fixed set of on-chip blocks that participate in a permutation



The Permutation Unit



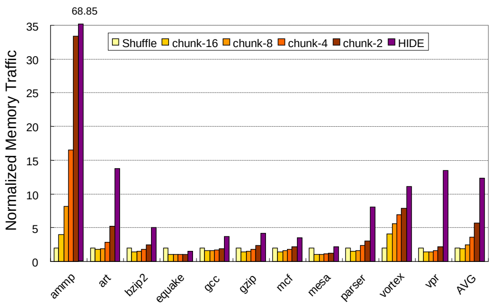
Outline

- Background & Motivation
  - Secure Processor Model
  - Address Information Leakage
- Previous Address Bus Protection Solutions
  - The HIDE Scheme
  - The Shuffle Scheme
- Our Low-Cost Address Permutation Scheme
- Performance Evaluation
- Conclusion

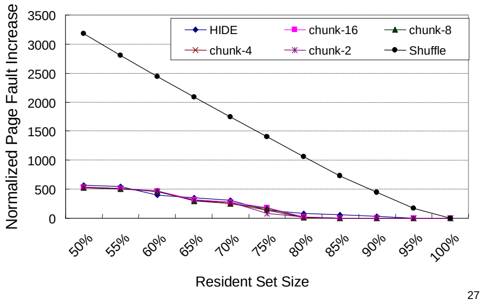
Experiment Environment

- Tools
  - Simplescalar Toolset 3.0
  - SPEC2K benchmarks
- Configuration
  - Cache
    - Separate L1 I- and D-cache: 8K, 32B line
    - Integrated L2 Cache: 1M, 32B line
    - Chunk Size: 8K, 16K, 32K, 64K
  - Other Settings
    - Page Settings: 4KB, perfect LRU repl policy
    - Perfect auxiliary on-chip storage for all schemes

Memory Traffic Comparison



Page Faults Comparison



Conclusion

- Proposed an efficient address permutation scheme to combat the information leakage on the address bus
- Tackled two main problems of the previous schemes:
  - The excessive memory traffic in the HIDE scheme
  - The increased page faults in the Shuffle scheme
- Preliminary experiments:
  - Reduce the memory traffic in HIDE from 12X to 1.88X
  - Keep the page fault rate as low as the base settings

Questions ?