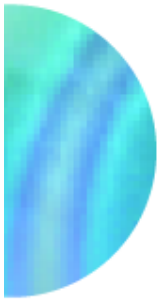
ELECTRICAL *and* COMPUTER ENGINEERING DEPARTMENT

Instruction Cache Locking inside a Binary Rewriter



Kapil Anand
Rajeev Barua

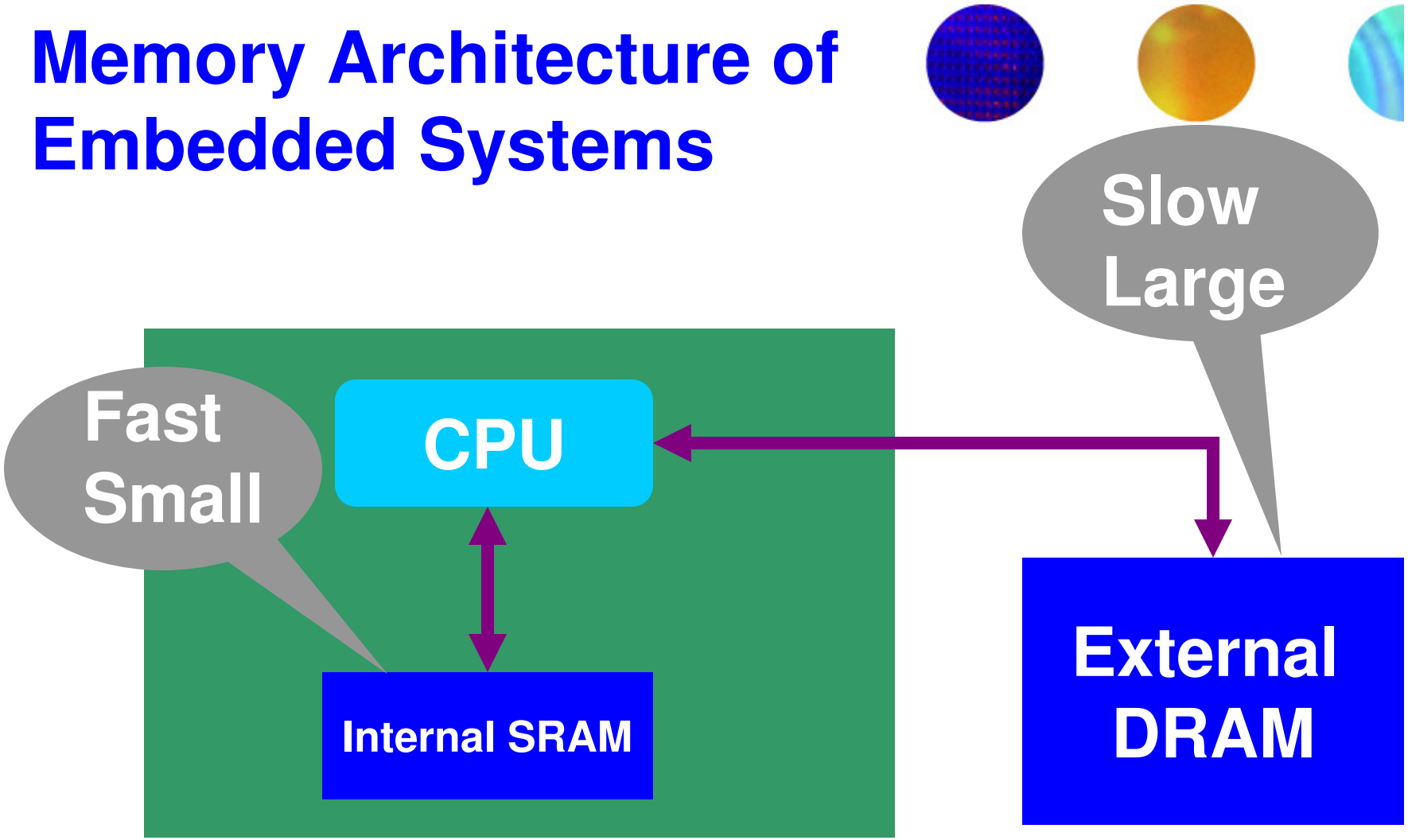
A. JAMES CLARK SCHOOL *of* ENGINEERING

UNIVERSITY OF MARYLAND

Outline

- Introduction
- Motivation
- Cache Locking Problem
- Algorithm
  - Binary Rewriter
- Experiment and Results
- Summary

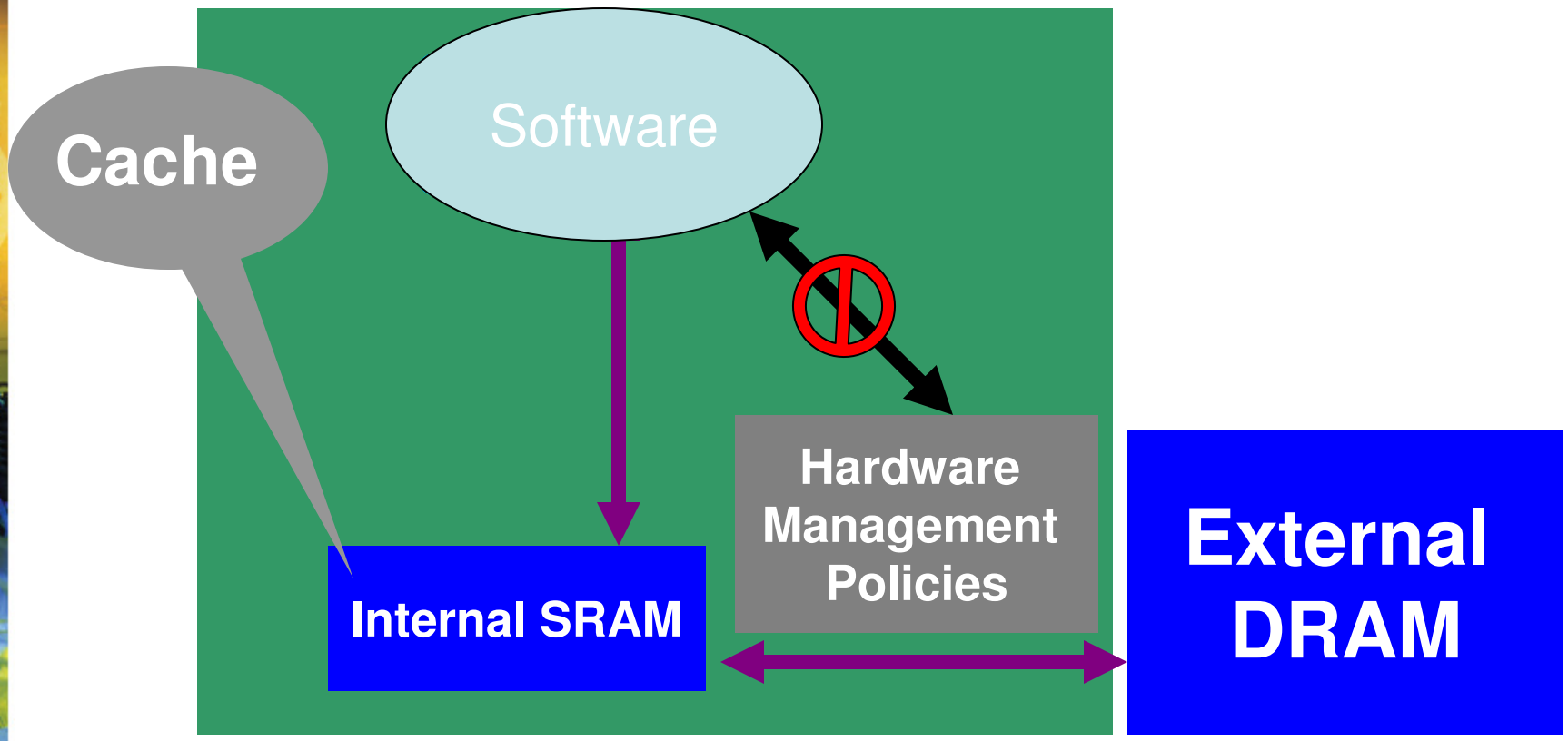
Memory Architecture of Embedded Systems



Internal SRAM should be managed intelligently

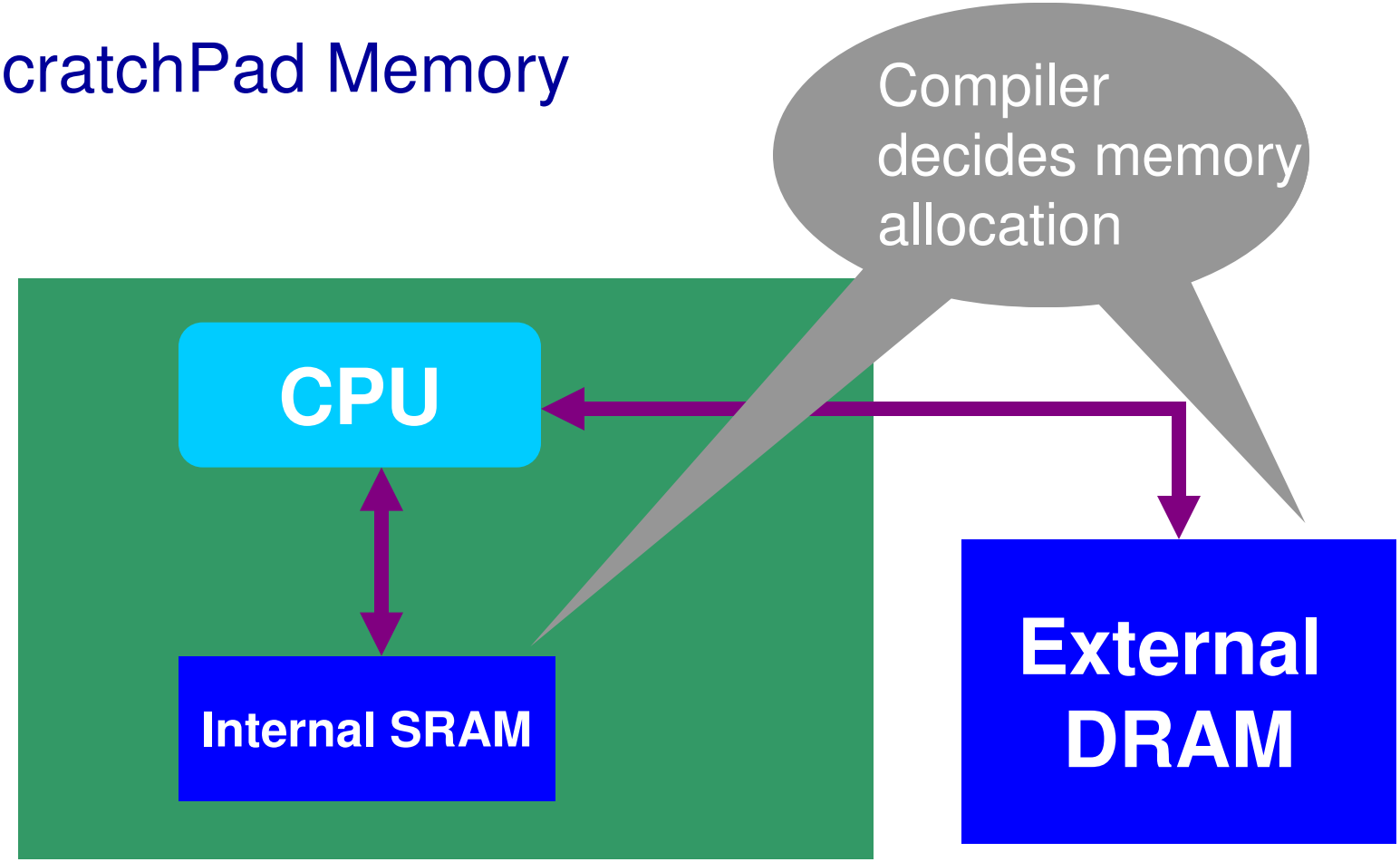
Internal SRAM Techniques

- Hardware Cache

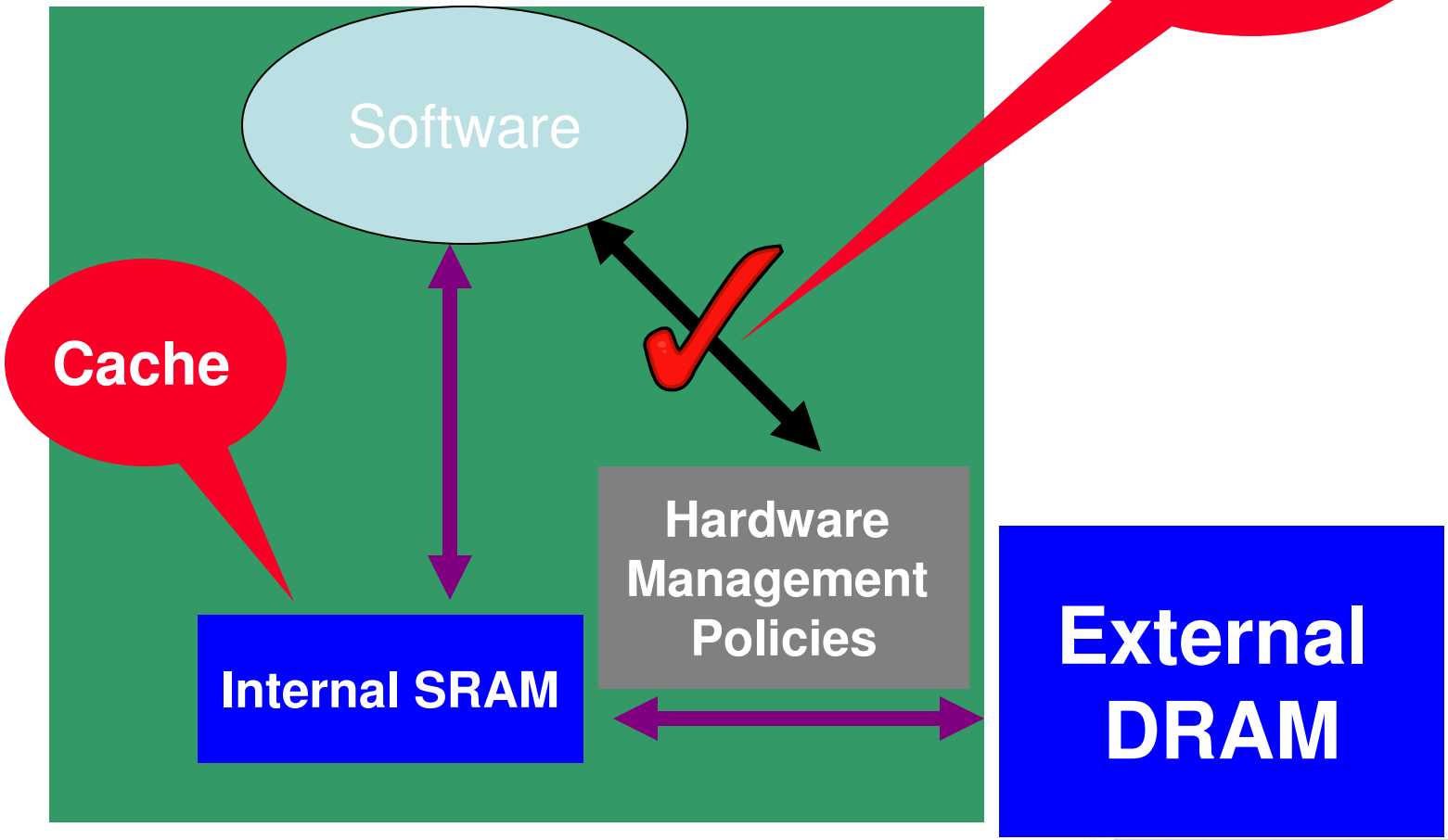


Software Involvement

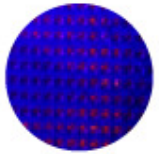
- ScratchPad Memory



Software Hints



Existing Cache Support



- Cache Hints in EPIC Architectures
  - Reuse distance based techniques suggested by Hollander et al
- Cache Locking
  - Intel Xscale, ARM Cortex, ARM9, ARM11
    - Coprocessor-based lock instructions for locking an address in the cache

Cache Locking

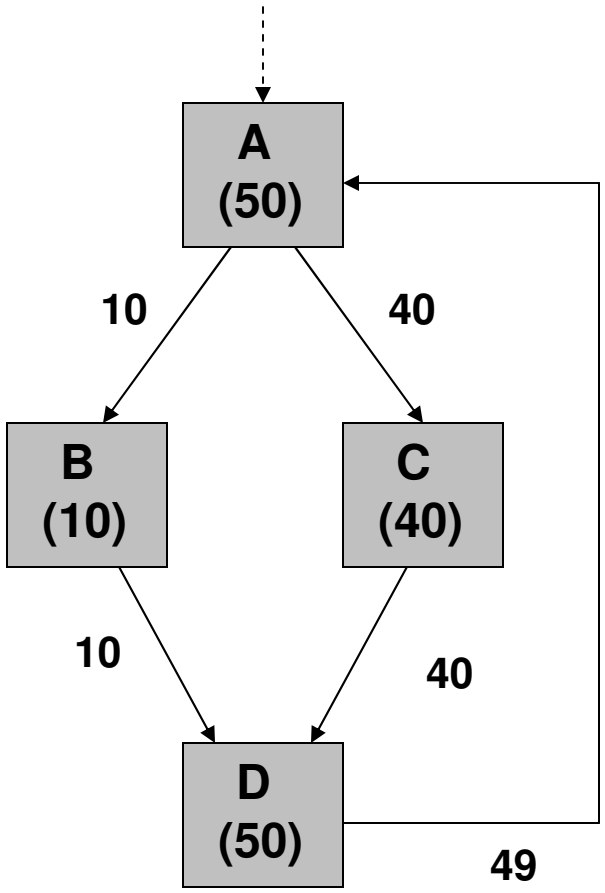
- What is Cache Locking?
  - The facility of locking one or more lines in the cache
  - An address, once locked in the cache, always results in a hit unless an unlocking operation is carried out

Software Influence on Cache Replacement!!

Cache Locking

- Current Uses
  - Adaptation of cache for multi-task real time systems
    - Campoy et al, 2001, Puaut et al, 2002, Arnaud and Puaut, 2006, Falk et al, 2007
  - Improves worst case estimation
- Our Objective
  - Improve average case run-time of embedded applications

Motivation

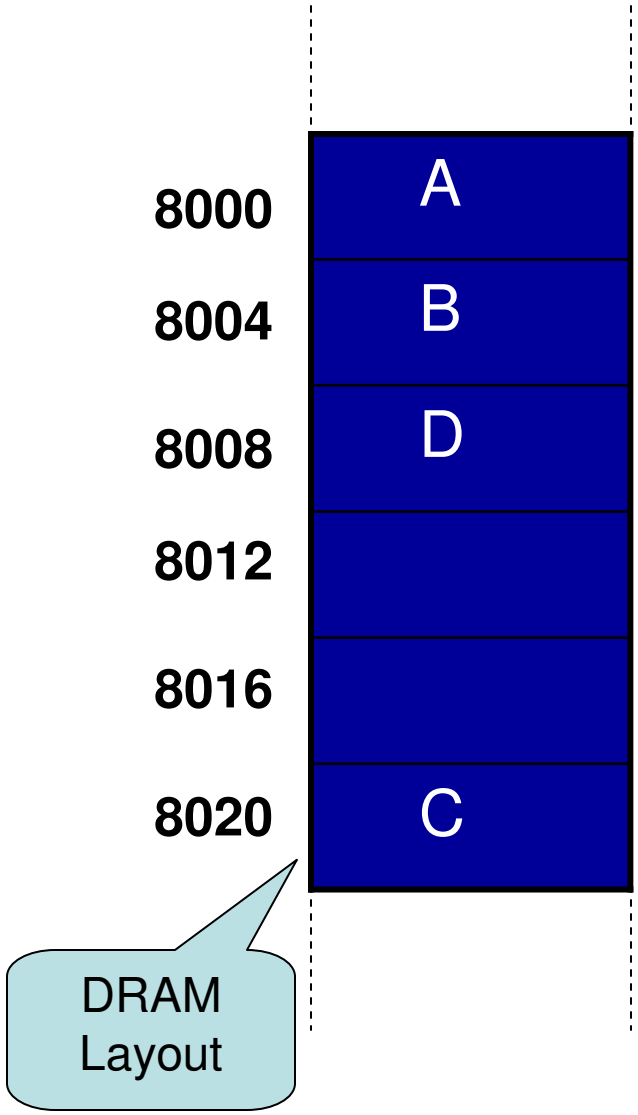


Control Flow Graph

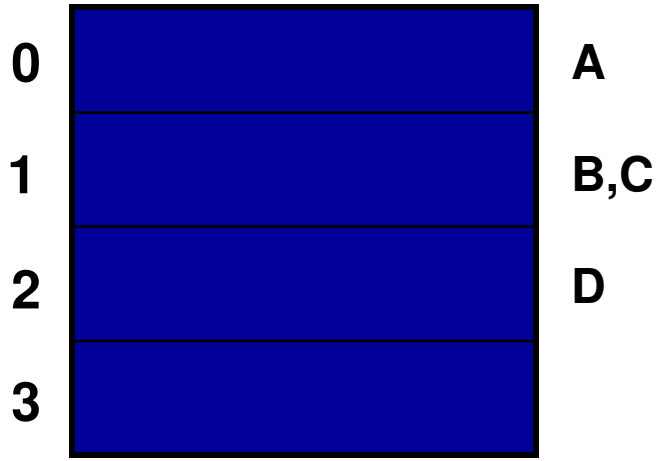
$$(ABD (ACD)^4)^{10}$$

Execution Trace

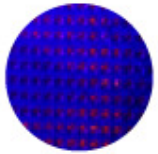
Motivation



4 Word Direct Mapped Cache



$(A\,B\,D\,(A\,C\,D)^{4})^{10}$

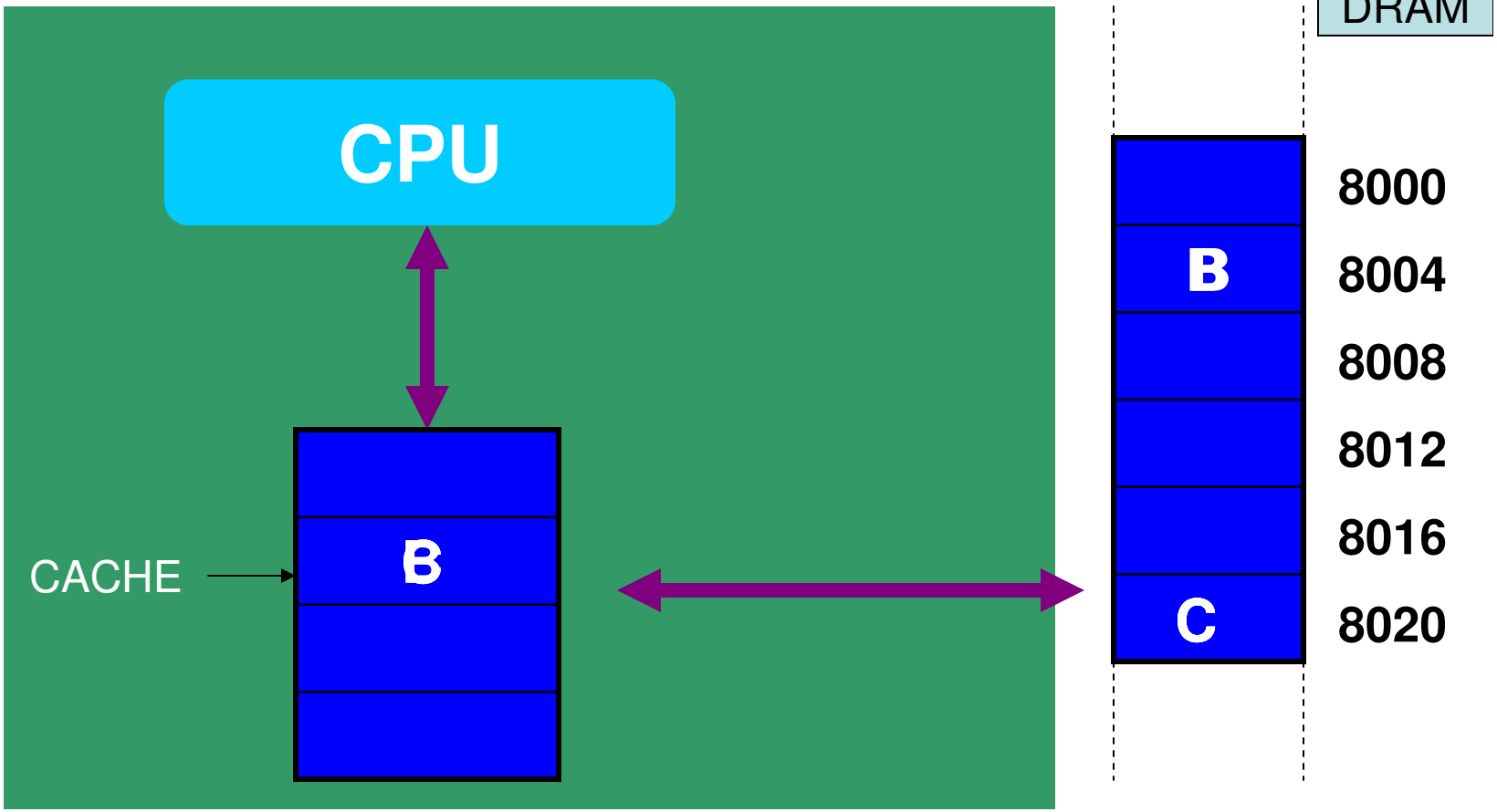


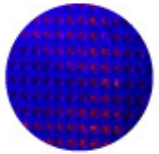
Without Cache Locking

Without Cache Locking

B C C ... B C

#Miss: 2



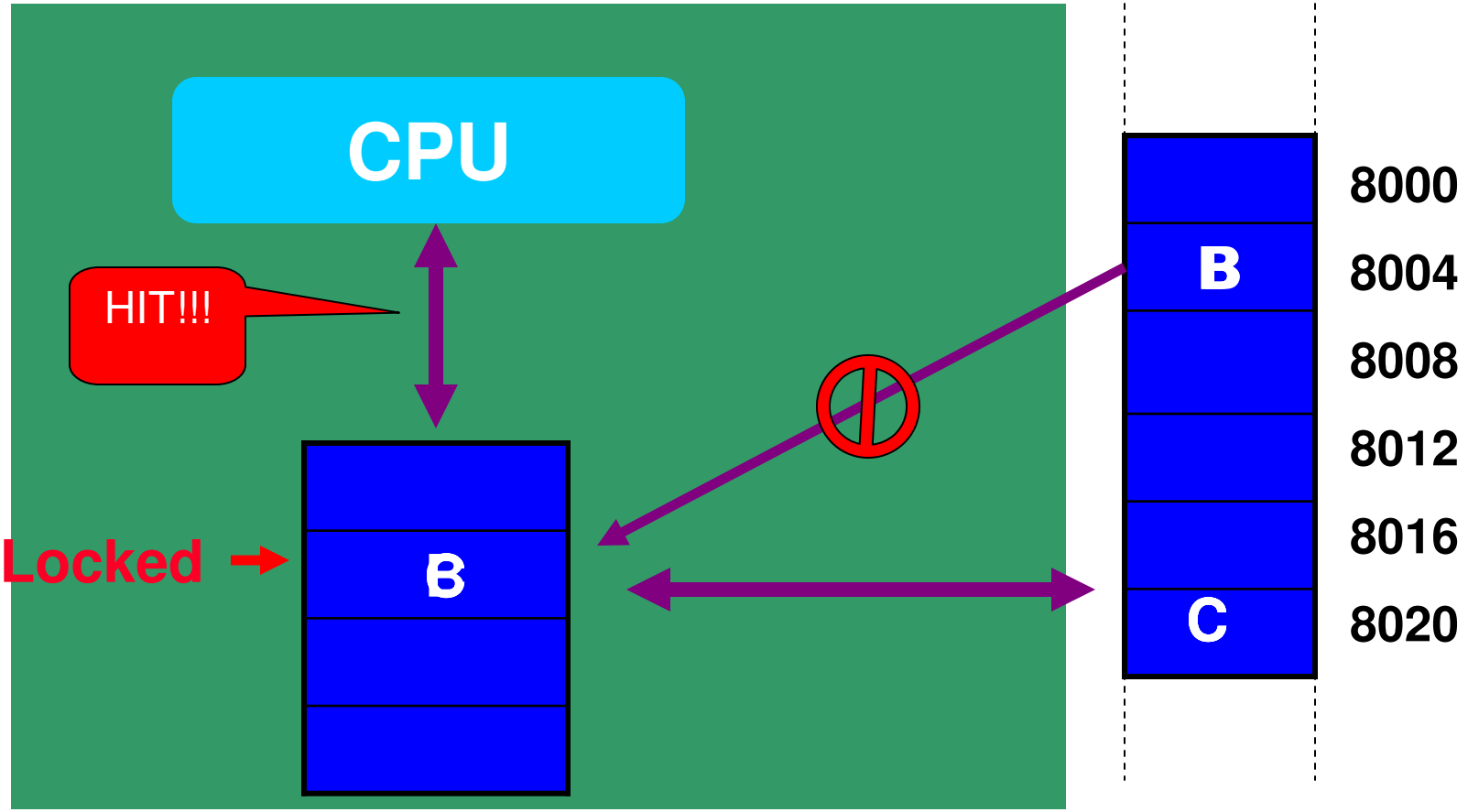


With Cache Locking

With Cache Locking

B C C ..... B C

#Miss: 2



Cache Misses

- **(ABD (ACD)4)10**

Node	# of Miss without locking	# of Miss With locking
A	1	1
B	10	10
C	10	1
D	1	1
Total	22	13

Cache Locking Problem

- Goal:

  “Select the memory addresses to be locked in the instruction cache to minimize the total number of instruction cache misses”

Problem Formulation

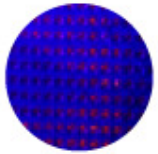
- Each Cache Set – INDEPENDENT
- Define:
  - N : Associativity of Cache
  - K : Maximum number of lines which can be locked in a set
  - L : Number of lines to be locked in the set

$$L \leq K \leq N$$

- For each cache set, Determine
  - L: The number of lines to be locked
  - LOCKLIST: The set of addresses to be locked

$$|\text{LOCKLIST}| = L$$

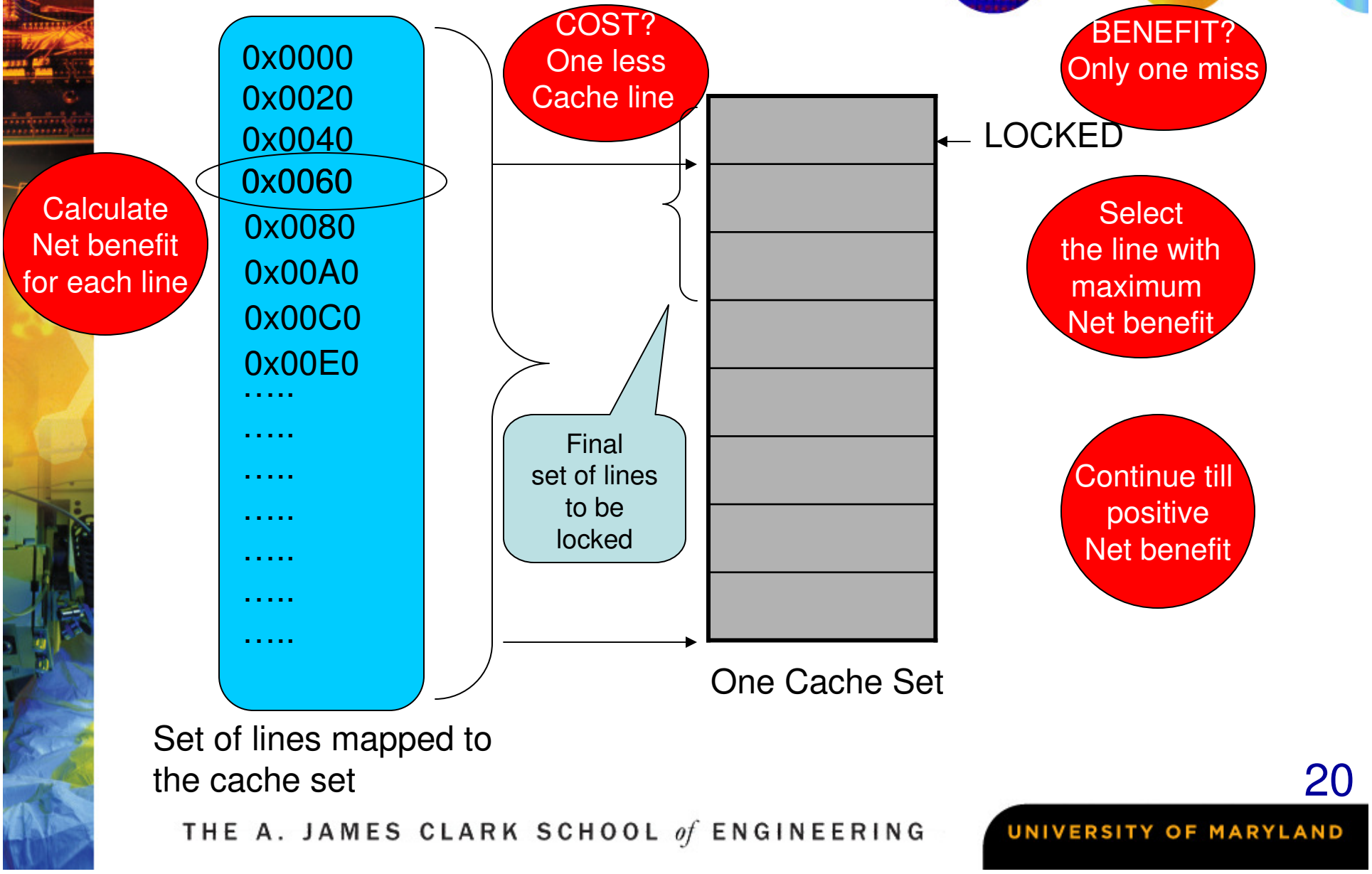
Complexity of Problem



- Exponential number of solutions

- Greedy and Iterative solution
  - Static

Solution Visualization



Time Model

- $Time(A/B)$
  - Total time to access A given the cache lines in set B have been locked

$$Time(x_i / LOCKLIST) = HIT_{LL}(x_i) * T_{HIT} + MISS_{LL}(x_i) * T_{MISS}$$

LOCKLIST – Set of lines locked in this set

LL – Number of lines locked in this set

Time Model

$$Time(x_i / LOCKLIST) = HIT_{LL}(x_i) * T_{HIT} + MISS_{LL}(x_i) * T_{MISS}$$

Suppose $F(x_i) = HIT_{LL}(x_i) + MISS_{LL}(x_i) \quad \forall\, LL$

$$Time(x_i / LOCKLIST) = HIT_{LL}(x_i) * T_{HIT} + (F(x_i) - HIT_{LL}(x_i)) * T_{MISS}$$

Benefit Model

- NoLockTime(xi)

$$Time(x_i / LOCKLIST) = HIT_{LL}(x_i) * T_{HIT} + (F(x_i) - HIT_{LL}(x_i)) * T_{MISS}$$

- LockTime(xi)

$$Time(x_i / (LOCKLIST \cup \{x_i\})) = T_{MISS} + (F(xi) - 1) * T_{HIT}$$

BenLock(xi) = NoLockTime(xi) – LockTime(xi)

Cost

- NoLockTime(xj)

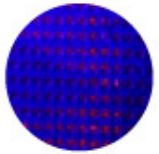
$$Time(x_j / LOCKLIST) = HIT_{LL}(x_j) * T_{HIT} + (F(x_j) - HIT_{LL}(x_j)) * T_{MISS}$$

- LockTime(xj/xi)

$$Time(x_j / (LOCKLIST \cup \{x_i\})) = HIT_{LL+1}(x_j) * T_{HIT} + (F(x_j) - HIT_{LL+1}(x_j)) * T_{MISS}$$

- CostLock(xj/xi) = LockTime(xj/xi) – NoLockTime(xj)
- Total Cost for locking xi = $\sum_j CostLock(xj / xi)$

Benefit Cost Model



- Net Benefit of locking a line xi

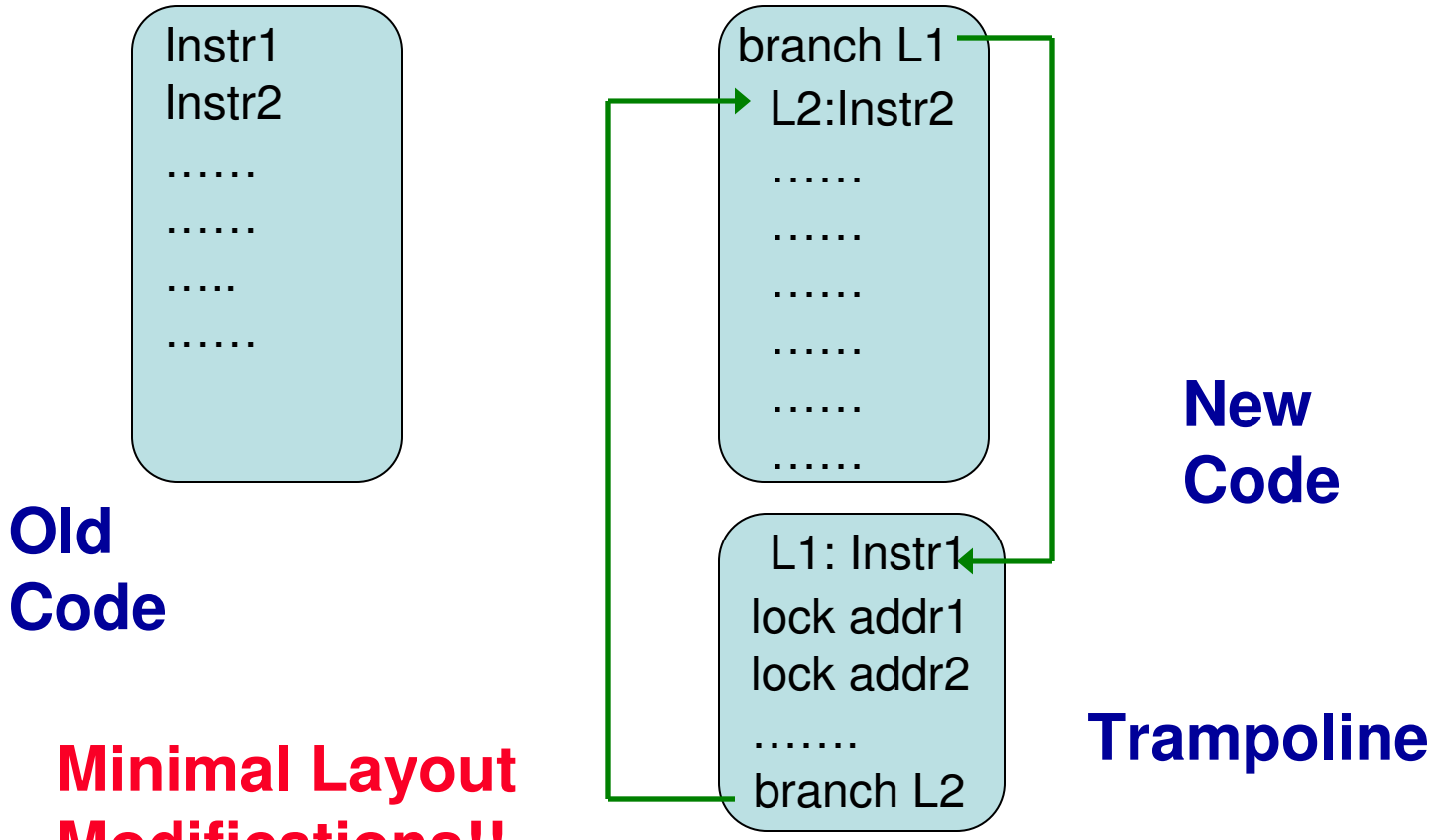
  $BenLock(x_i) - CostLock(x_i)$

Algorithm

- Beginning:
  - LOCKLIST is empty
- (LL+1)th iteration : LOCKLIST of LL lines
  - NetBenefit for each of the virtual cache line xi
  - Select virtual cache line with maximum net benefit
  - Add to LOCKLIST
- Iteration continued until
    - We reach the limit of maximum cache lines
    - The net benefit becomes zero

Binary Rewriting

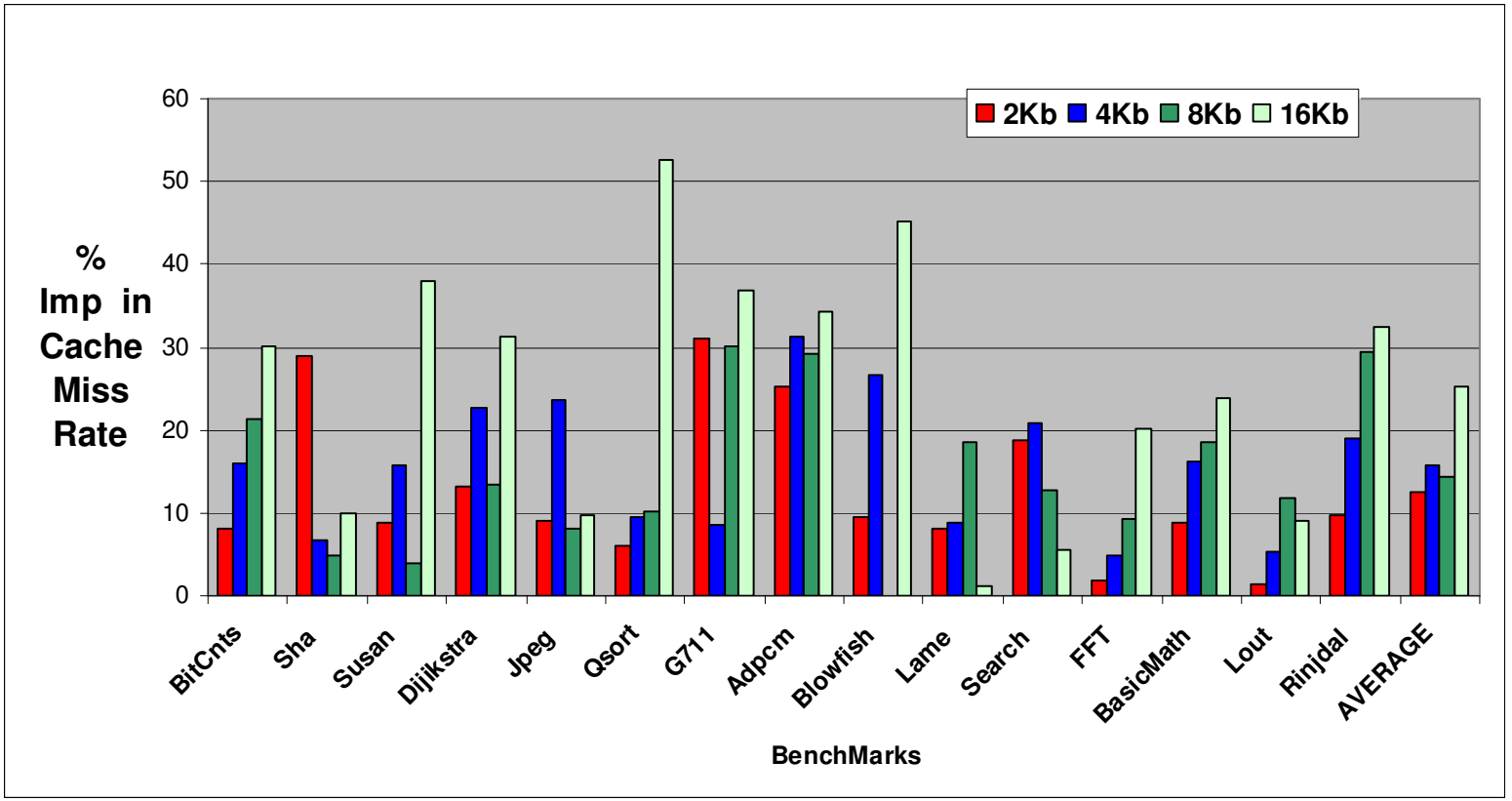
- Trampolines for inserting lock instructions



Bench Marks

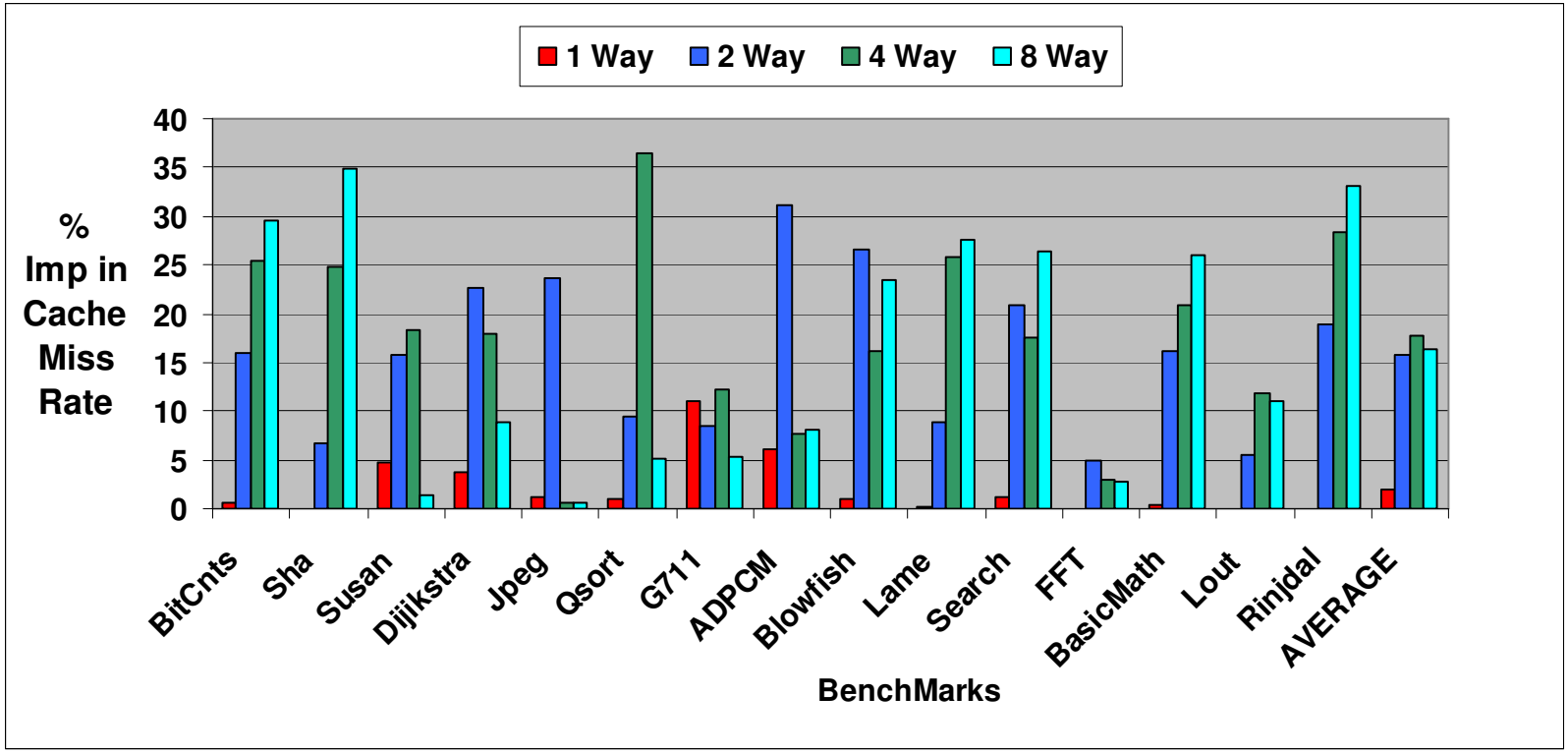
Application	*Source*	*Lines of Code*
BitCnts	MiBench	543
QuickSort	MiBench	79
Susan	MiBench	1456
Jpeg	MiBench	19804
Lame	MiBench	15959
Dijikstra	MiBench	268
StringSearch	MiBench	3072
Blowfish	MiBench	3260
Rinjdael	MiBench	1017
Sha	MiBench	207
BasicMath	MiBench	7367
FFT	MiBench	278
Lout	MiBench	30689
ADPCM	MediaBench	411
G711	MediaBench	1173

Results



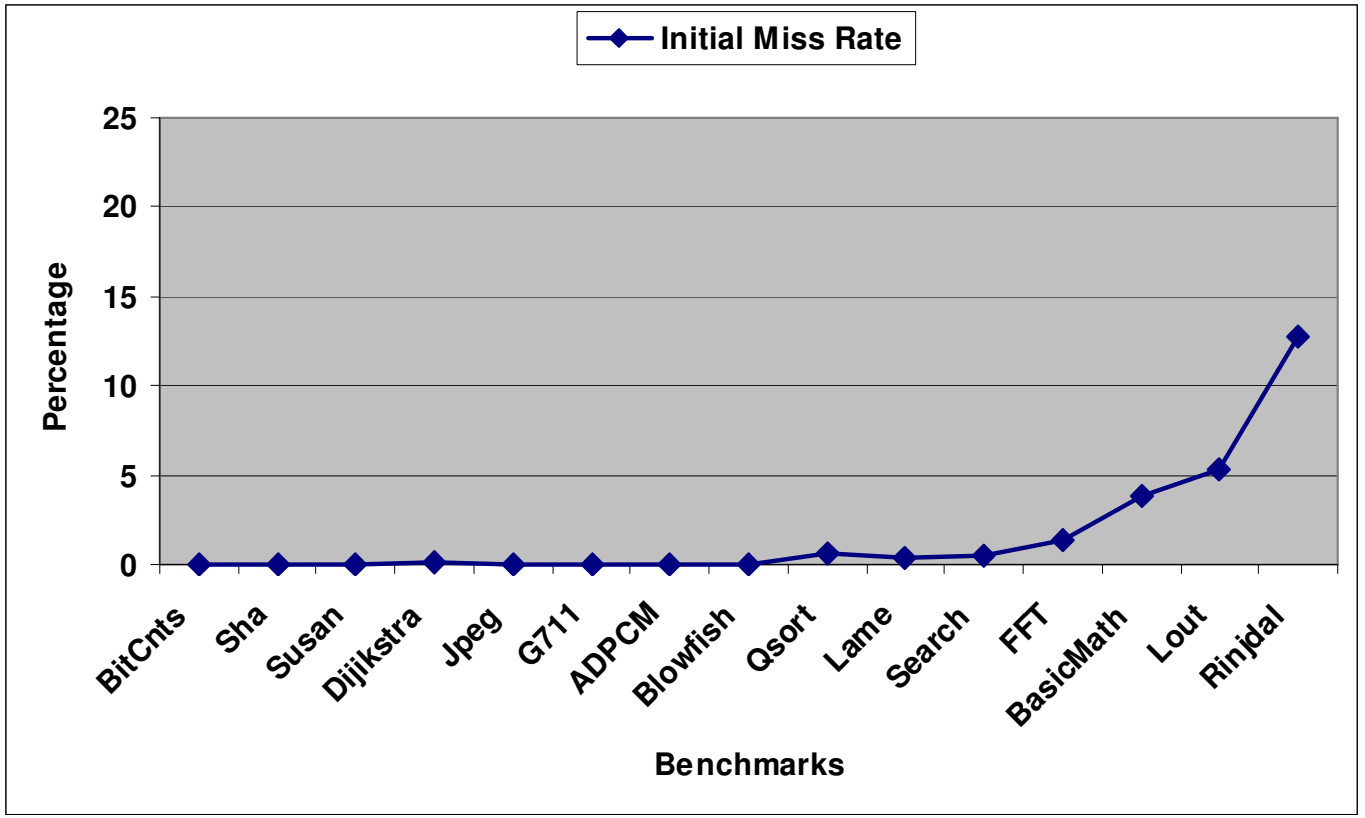
Improvement in Cache Miss Rate with variation in Cache size

Results

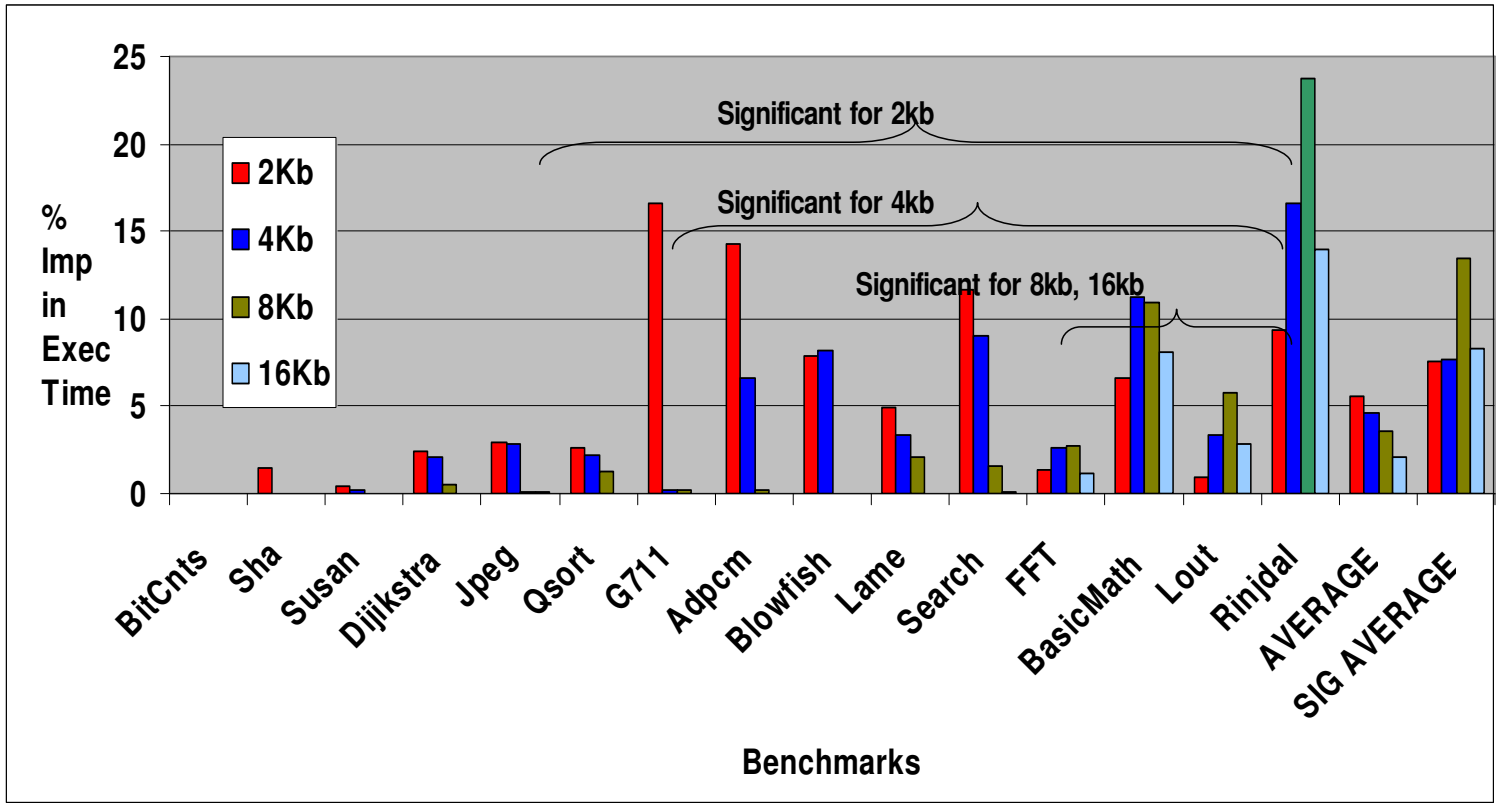


Improvement in Cache Miss rate with variation in associativity

Results

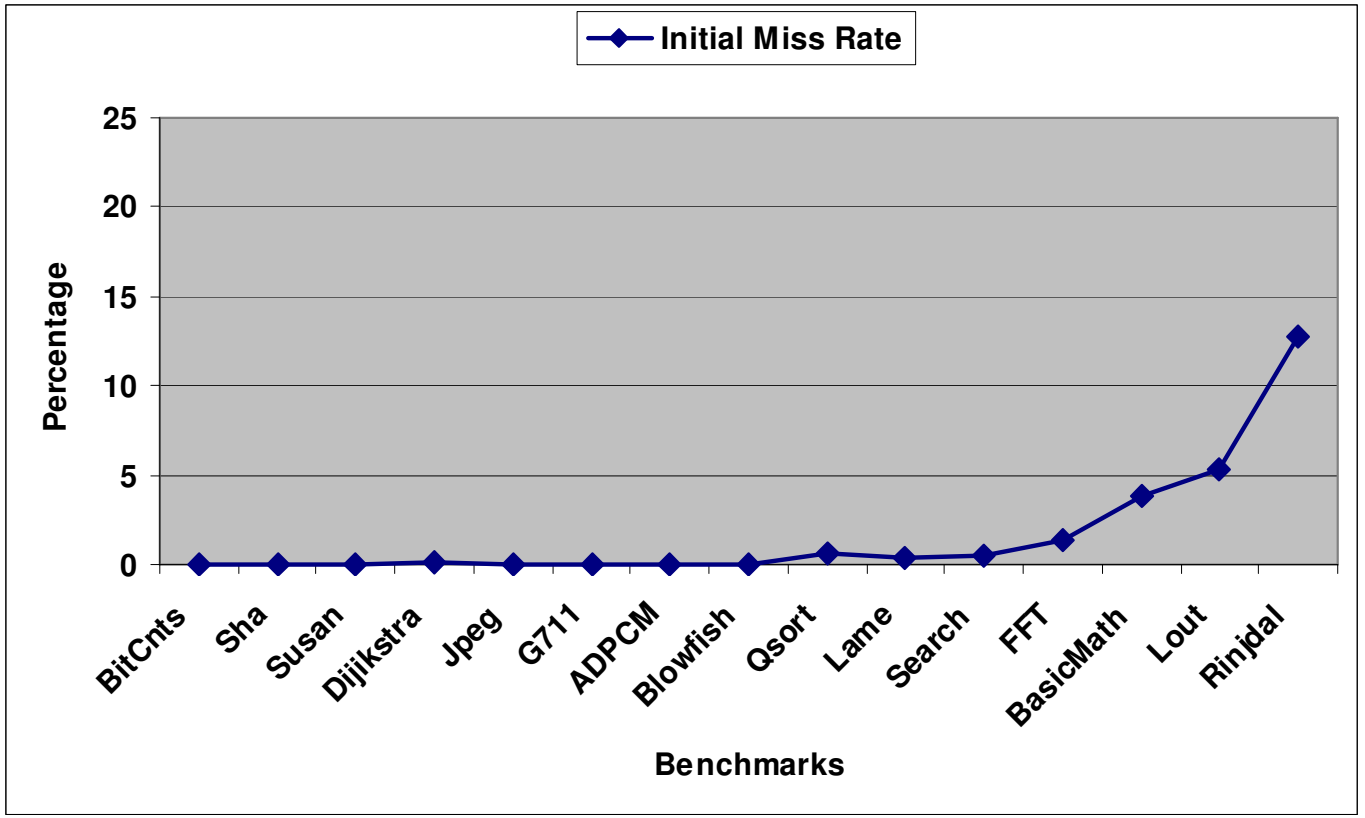

Initial Miss Rate
25
20
15
10
5
0
Percentage
BitCnts
Sha
Susan
Dijikstra
Jpeg
G711
ADPCM
Blowfish
Qsort
Lame
Search
FFT
BasicMath
Lout
Rinjdal
Benchmarks

Results

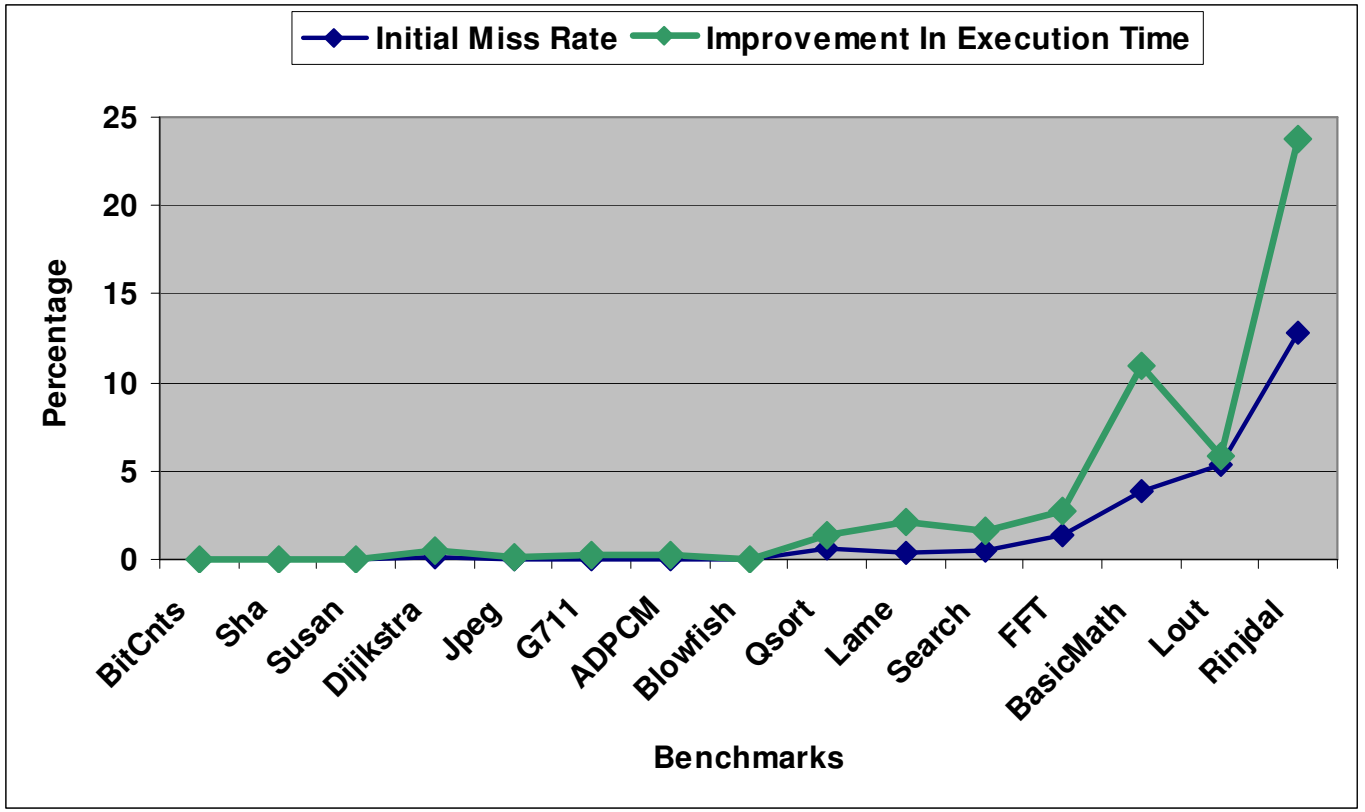


Improvement in Execution Time with variation in Cache Size

Results


Initial Miss Rate
25
20
15
10
5
0
Percentage
BitCnts
Sha
Susan
Dijikstra
Jpeg
G711
ADPCM
Blowfish
Qsort
Lame
Search
FFT
BasicMath
Lout
Rinjdal
Benchmarks

Results


Initial Miss Rate
Improvement In Execution Time
25
20
15
10
5
0
Percentage
BitCnts
Sha
Susan
Dijikstra
Jpeg
G711
ADPCM
Blowfish
Qsort
Lame
Search
FFT
BasicMath
Lout
Rinjdal
Benchmarks

Summary

- We present a new technique for cache locking
  - Uses Binary Rewriter
- First instruction cache locking method for average case run-time improvement
- Provides 13.5% improvement in execution time for memory access constrained benchmarks through a “pure software” method on existing commercial hardware

Q&A

- MERCI !!!!!

Need for Approximation

$$Time(x_j/(LOCKLIST \cup \{x_i\})) = HIT_{LL+1}(x_j) * T_{HIT} + (F(x_j) - HIT_{LL+1}(x_j)) * T_{MISS}$$

Can't be accurately determined!!!!

- Decreased hit rate can't be calculated accurately
- Approximate value by locking a dummy (unused) virtual cache line
- Always provides conservative estimates for future hit rate – GUARANTEED IMPROVEMENT

Problem Formulation

- Each set -- INDEPENDENT
- N way set associative cache
- K – maximum number of locked lines in a set
- M – number of cache lines getting mapped to this set

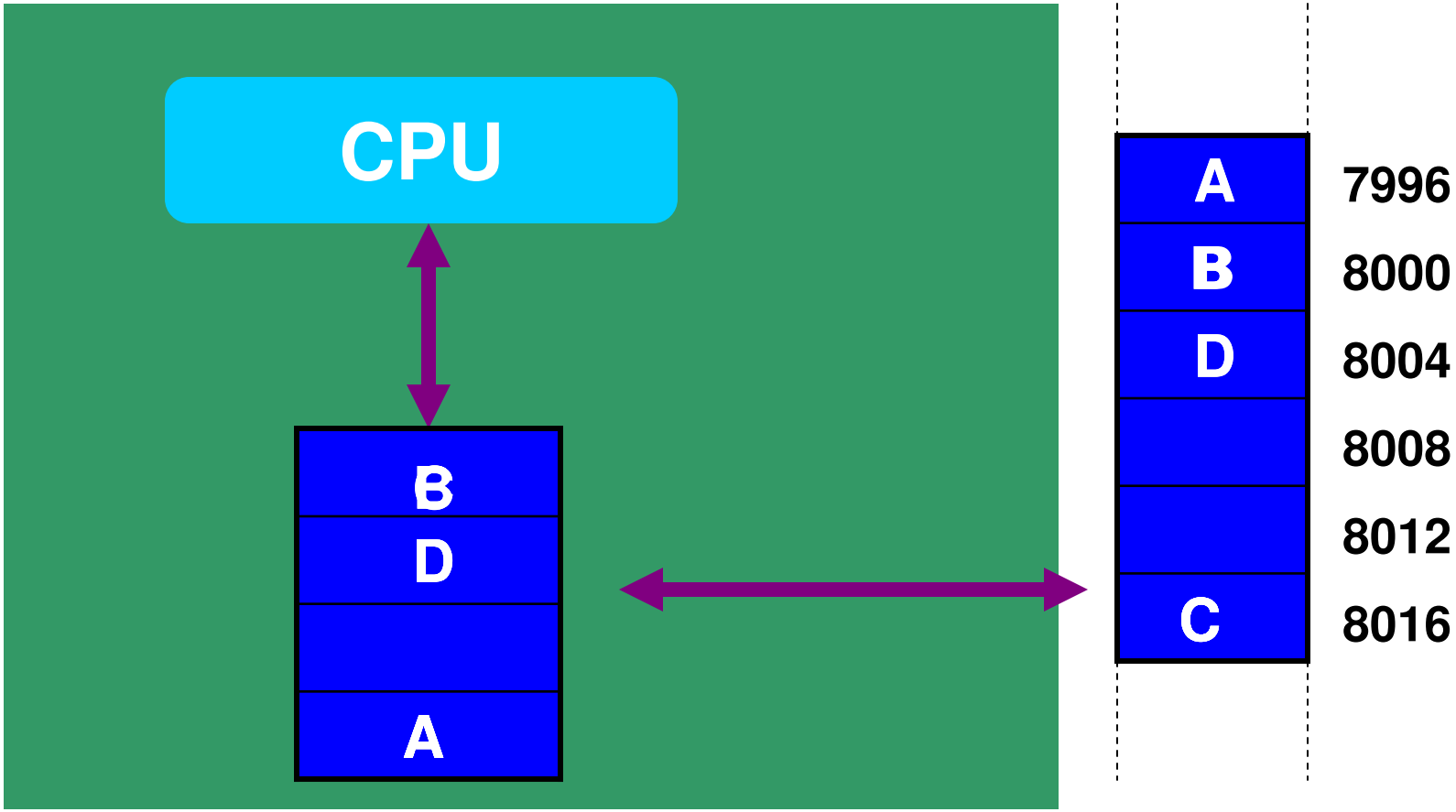
- Problem:
  - determining L: the number of lines to be locked
  - selecting L virtual cache lines out of M candidates

Objective

- Objective of previous Cache Locking
  - improve the worst case system behavior
- Our objective

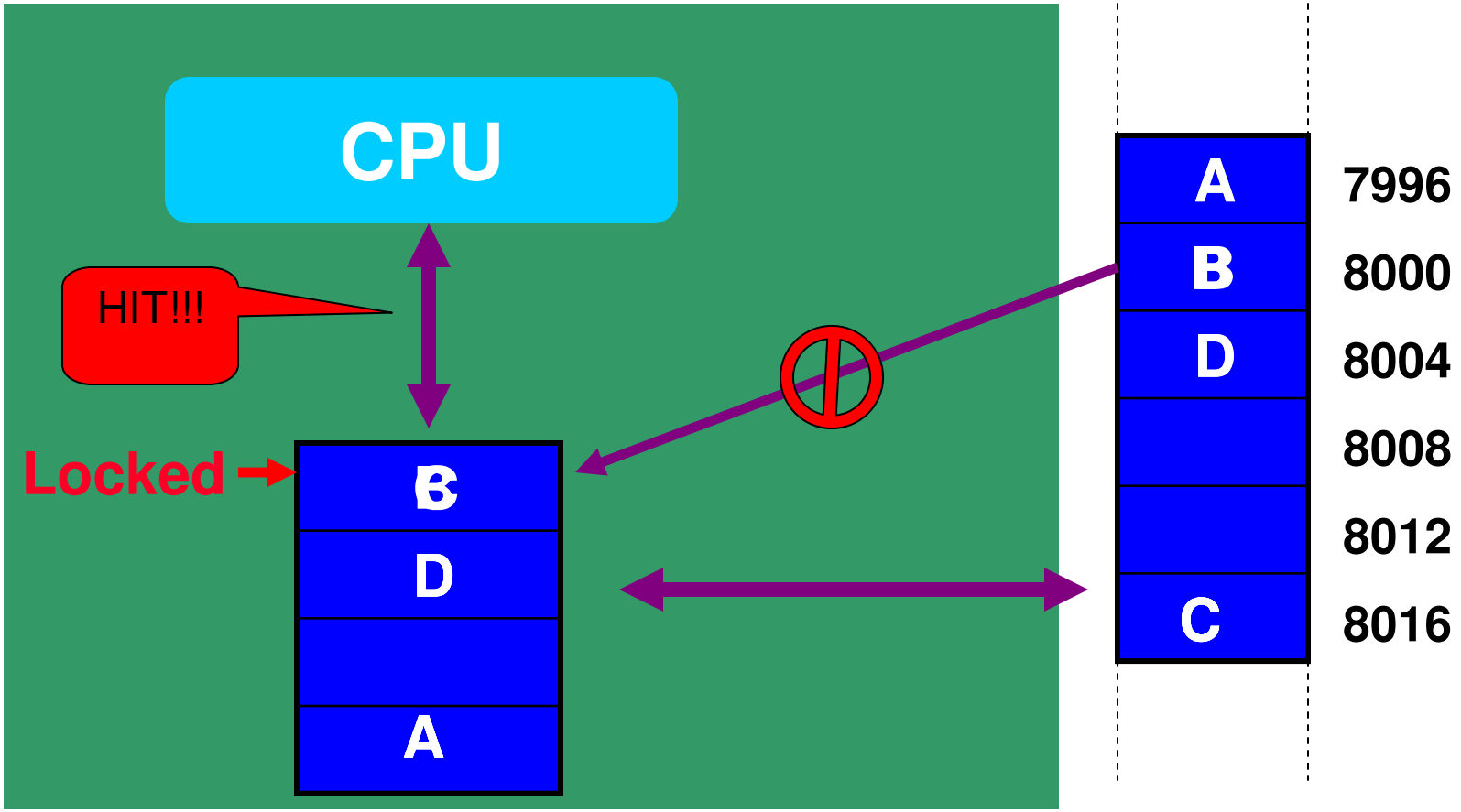
Example

A B D A C D ..... A B D A C D

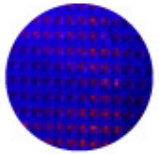


Example

A B D A C D ..... A B D A C D

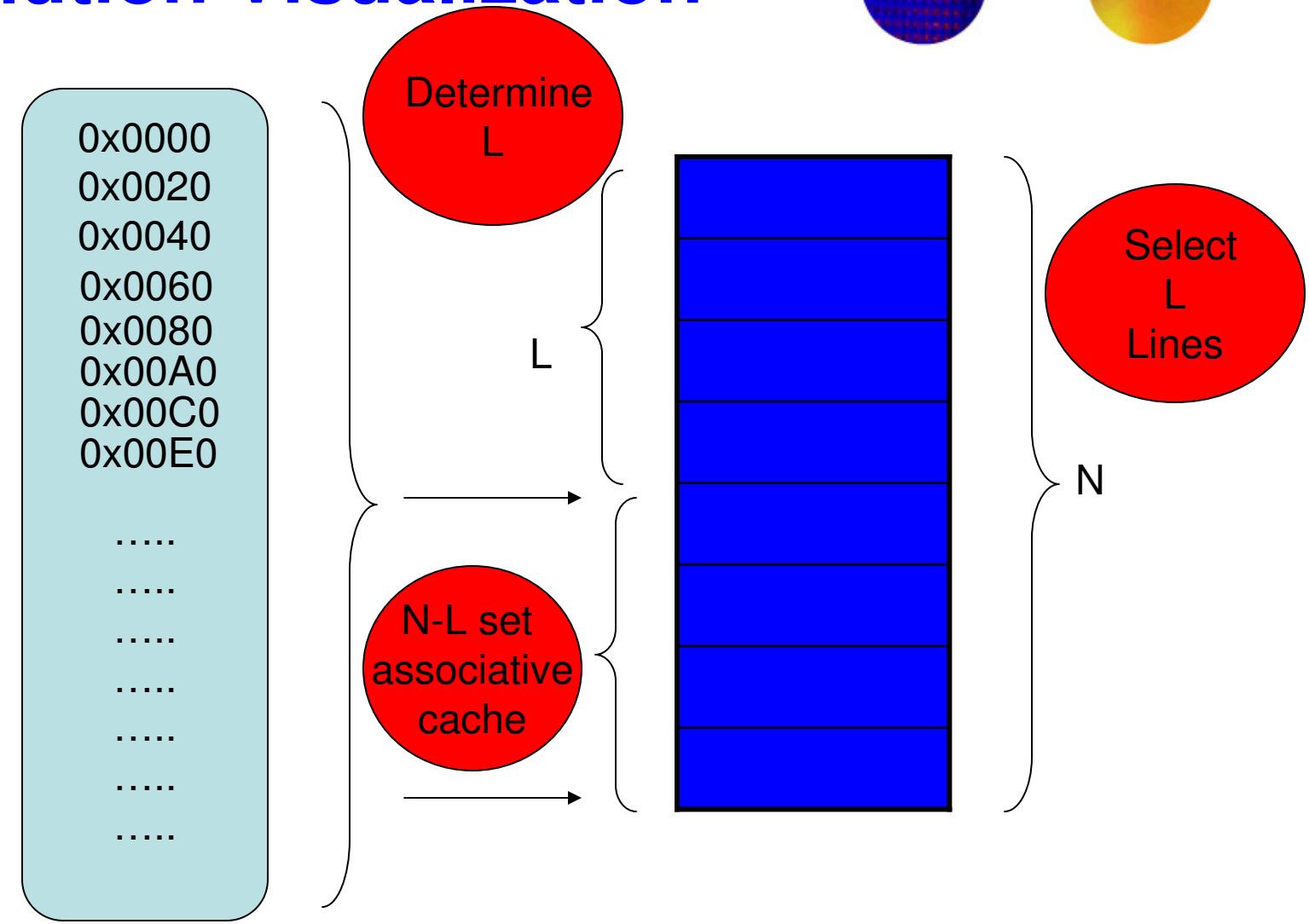


Cache Locking Interface



- Cache Locking Interface
  - Line Locking
    - Virtual Cache Line = Addr/Words-Per-Line

Solution Visualization



Cost-Benefit Model

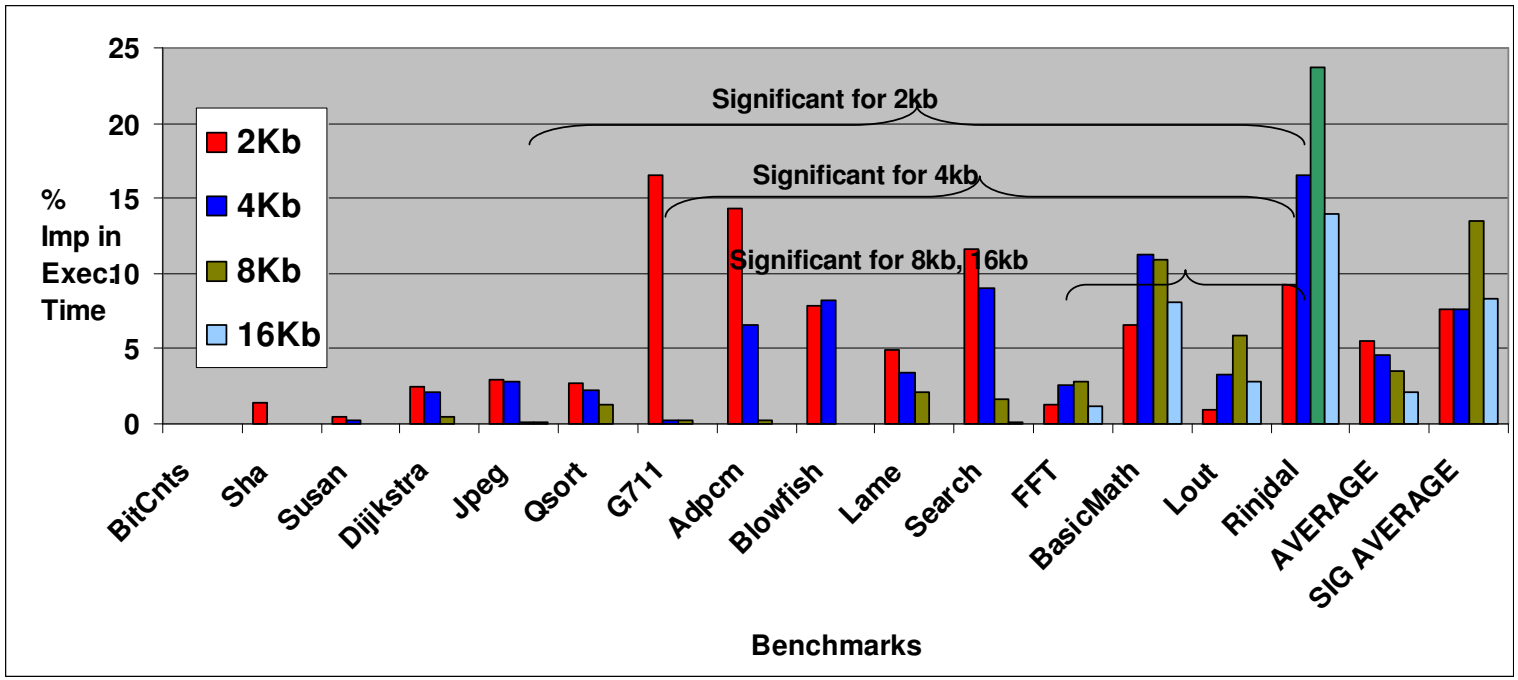
- If x_i is locked
  - x_i will observe only one compulsory miss
    BENEFIT!!!
  - One less cache line is available for remaining lines
    COST!!!

Greedy Formulation

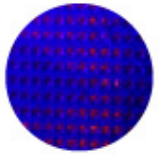
- Objective Function

$$Total\ Time = \sum_i Time(x_i / LOCKLIST)$$

- Find LOCKLIST to minimize total time
- Greedy and iterative solution for selecting one line at a time


25
20
15
10
5
0
% Imp in Exec Time
2Kb
4Kb
8Kb
16Kb
Significant for 2kb
Significant for 4kb
Significant for 8kb, 16kb
BitCnts
Sha
Susan
Dijkstra
Jpeg
Qsort
G711
Adpcm
Blowfish
Lame
Search
FFT
BasicMath
Lout
Rinjdal
AVERAGE
SIG AVERAGE
Benchmarks

Types of Software Involvement



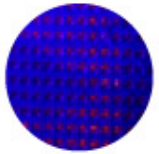
- Program Level Cache Control
  - Column Caching [Rudolph et al]
  - ReadMe/EvictMe Bits [Want et al]

Hardware schemes not present in current commercial architectures!!!

Why Different Problem?

- Didn't consider benefit, cost of locking
- Lock the line which has maximum number of accesses
- Number of lines to lock?
  - Might result in large number of wrong decisions

Example



for(){
access A
}// 50 iterations

for(){
access B
} // 20 iterations

for(){
access C
} // 30 iterations

- 2 way set associative cache
- Lock A and C
- All accesses to B – misses
- Right Solution ::: Don't Lock anything
- OUR MODEL FINDS THIS SOLUTION