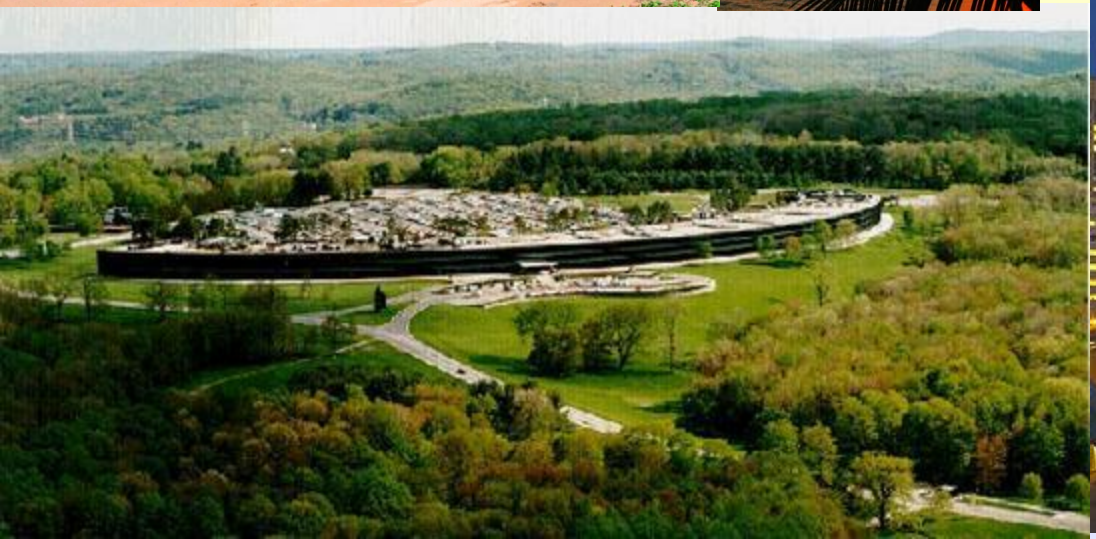
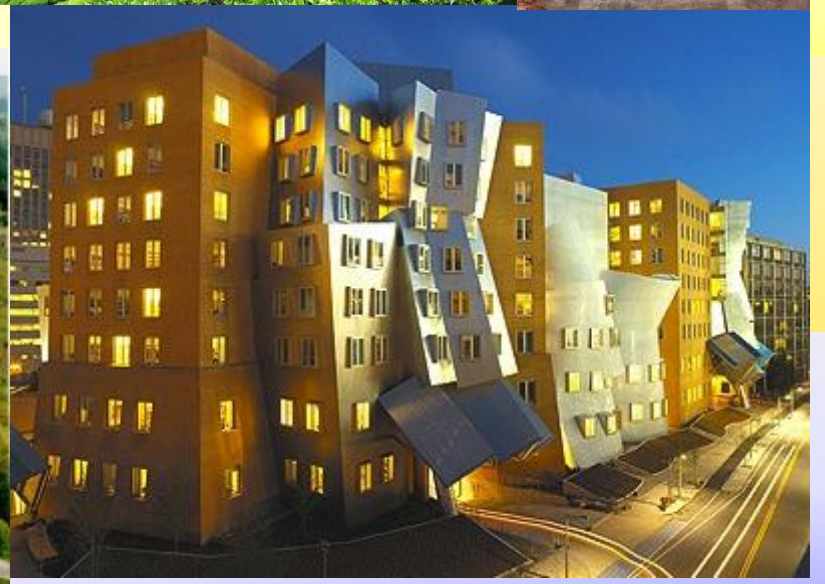
Detection of False Sharing Using Machine Learning





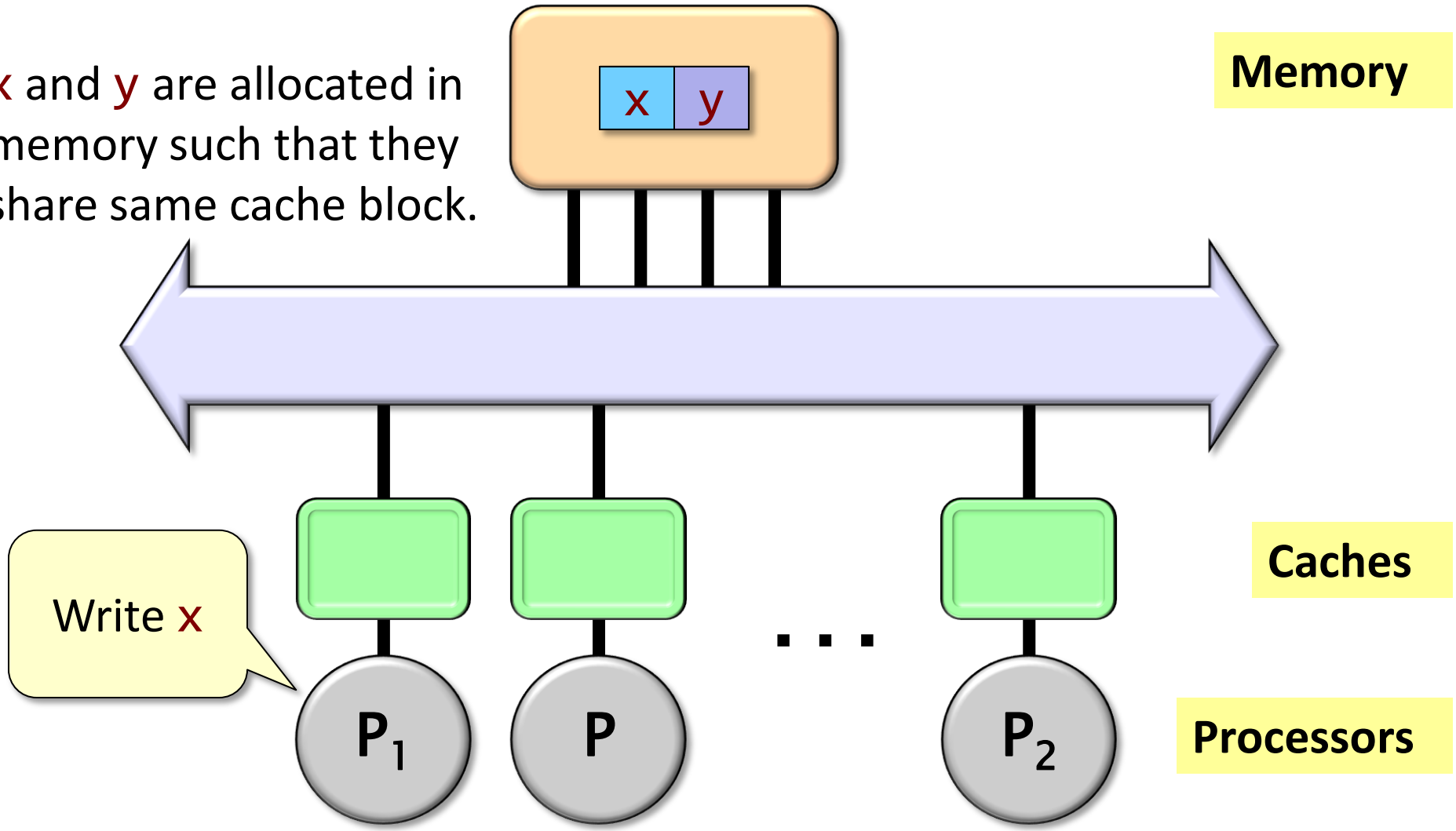



Perils of Parallel Programming

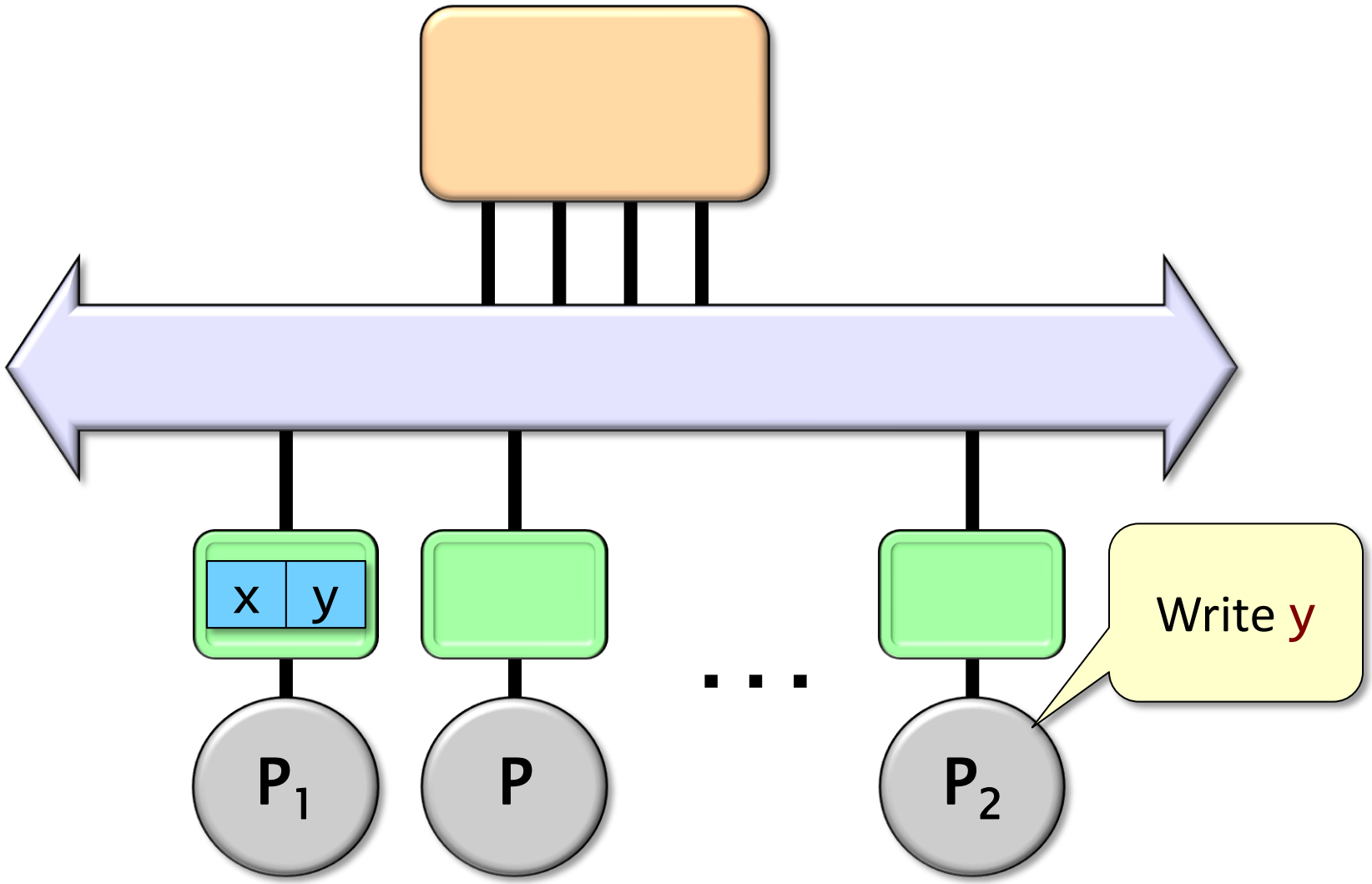
- Parallel programming is unavoidable in the era of the multicore
- Use of multiple threads on shared memory introduces new classes of correctness and performance bugs
- Some of these bugs are hard to detect and fix
- **False Sharing** is one such performance bug

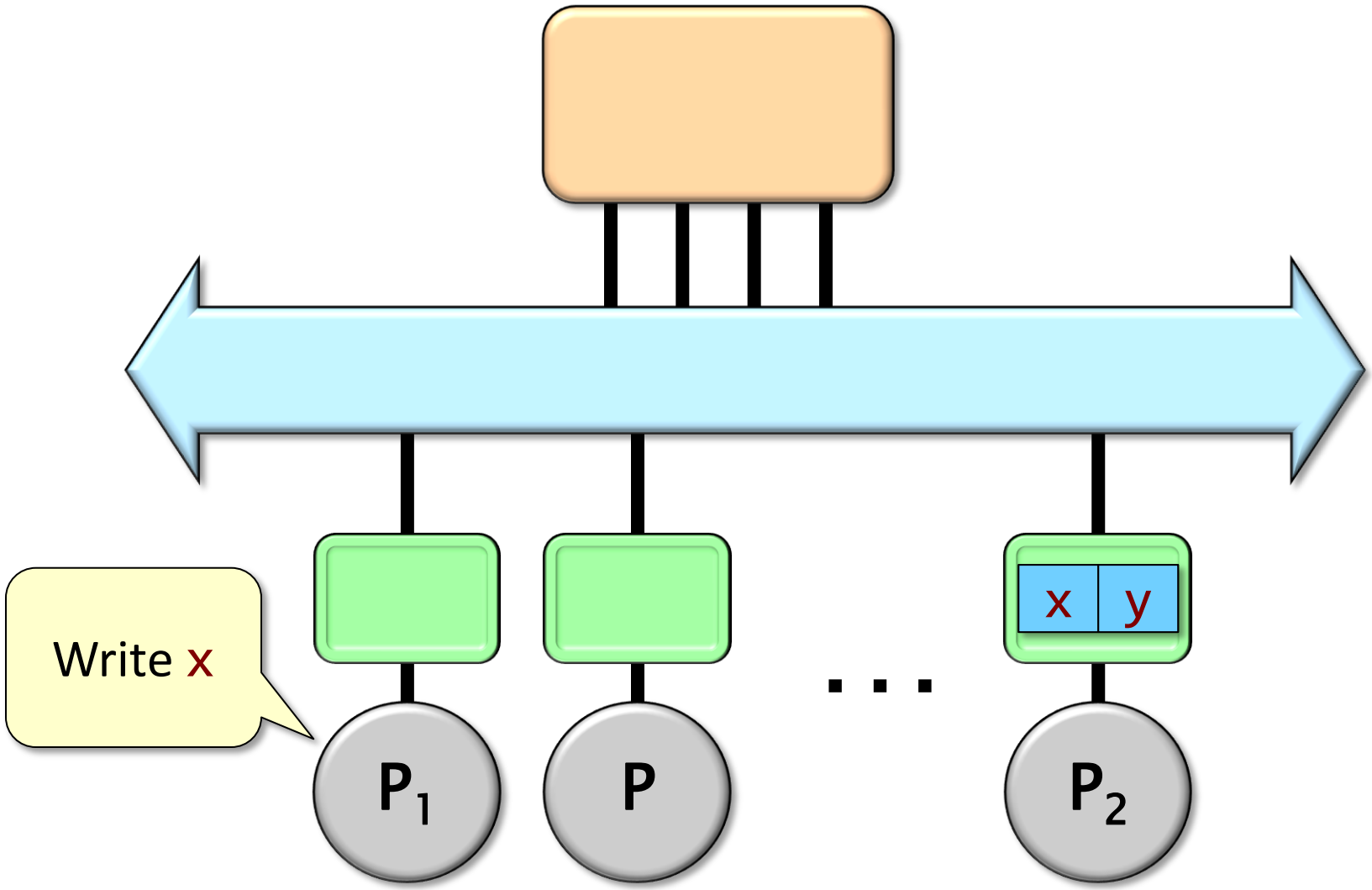
What is False Sharing?

x and y are allocated in memory such that they share same cache block.



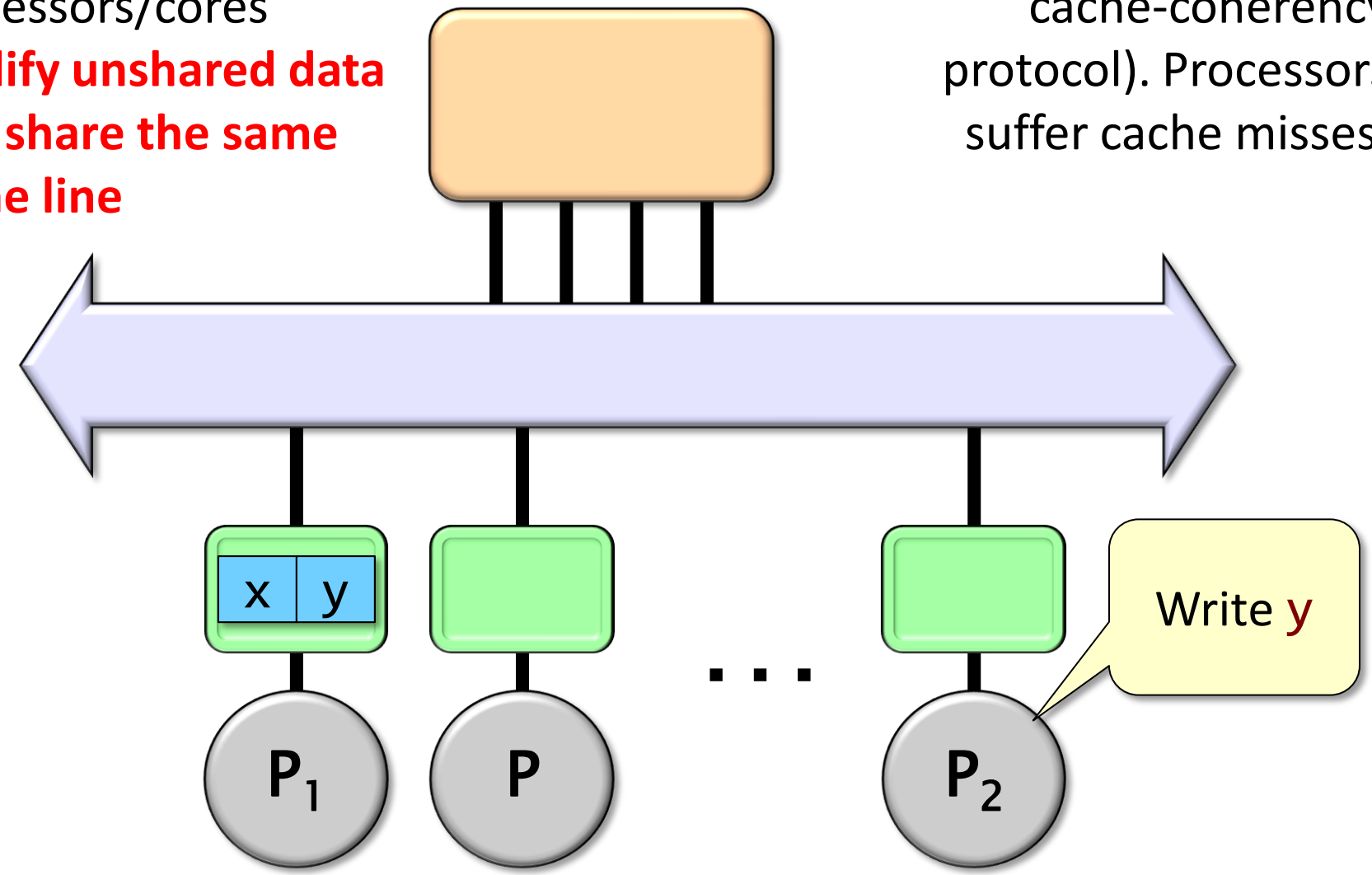
Source: [Leiserson & Angelina Lee, 2012]


x
y
Write y
P1
P
P2


Write x
x
y
P1
P
P2

False sharing: threads running on different processors/cores **modify unshared data** that **share the same cache line**

Ping-pong effect on cache-line (due to cache-coherency protocol). Processors suffer cache misses.



False Sharing: Program Example

Computing the dot-product of two vectors v1[N], v2[N]

```
int psum[MAXTHREADS];
int V1[N], V2[N];
```

```
void pdot_1(…) {
    int mysum = 0;
    for(int i=myid*BLKSZ; i < min((myid+1)*BLKSZ, N); i++)
        mysum += V1[i] * V2[i];
    psum[myid] = mysum;
}
```

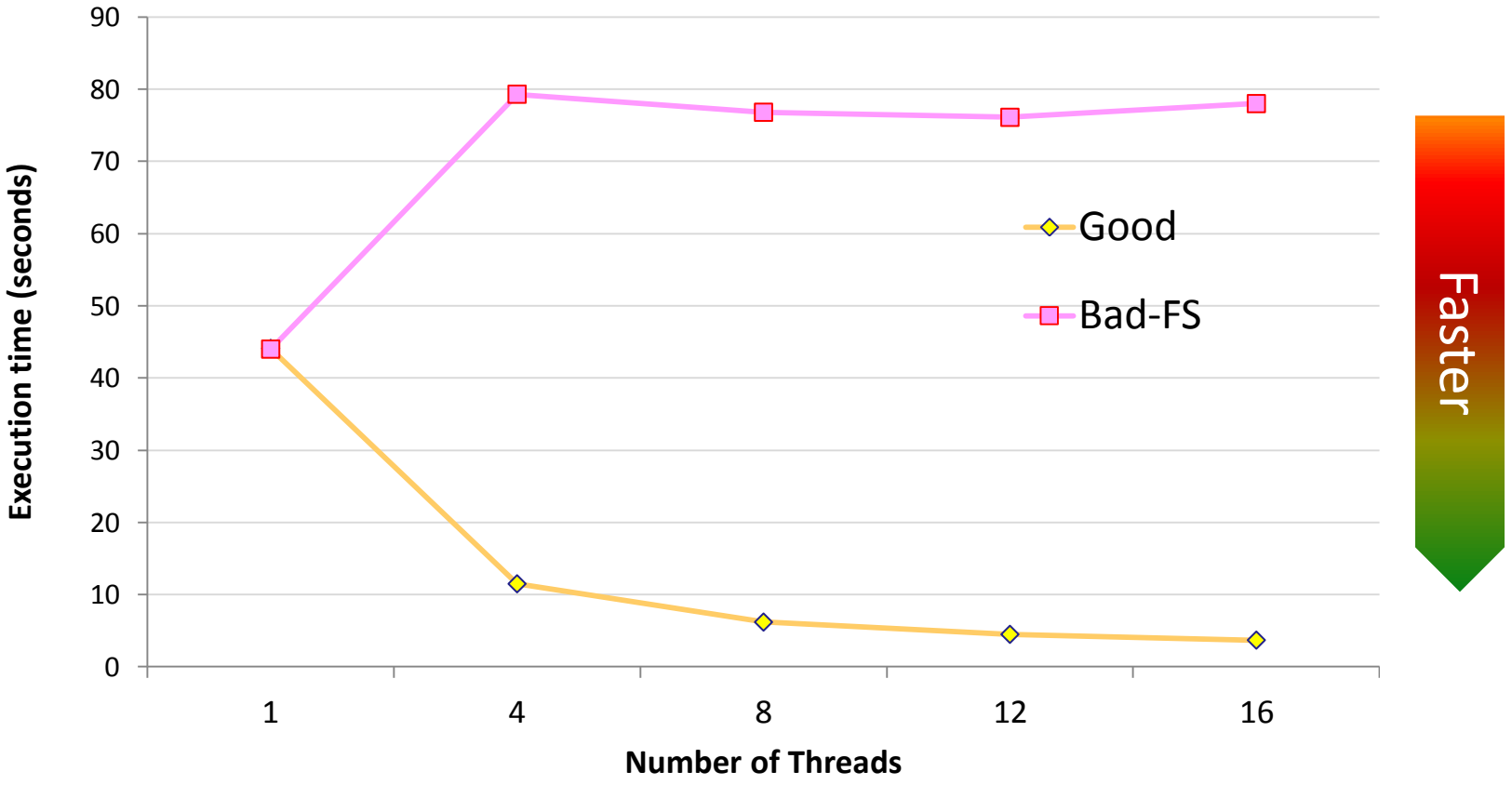
GOOD

```
void pdot_2(…) {
    for(int i=myid*BLKSZ; i < min((myid+1)*BLKSZ, N); i++)
        psum[myid] += V1[i] * V2[i];
}
```

BAD-FS

False Sharing: Impact

Elapsed times (seconds) for the parallel dot-product on a 32-core Intel Xeon X7550 Nehalem system, Vector size N=10^8



Detecting False Sharing is Hard

- The program is functionally correct
  - Only running much slower than possible
  - Major class of bugs
- There is no sharing at the program level
  - Two interfering variables that share a cache line are independent with no visible relationship
  - Program analysis will not find it
- Happens due to interaction among cores
  - Looking within a single core does not reveal the problem

Recent Work

- [Zhao et al, 2011, VEE]
  - Dynamic instrumentation, memory shadowing
  - Excessive run-time overhead (5x slowdown), limited to 8 threads
  - Some cache misses identified as false sharing
- [Liu & Berger, 2011, OOPSLA]
  - 'SHERIFF' framework replaces pthreads, breaks threads into processes
    - Big change to the execution model
  - 20% run-time overhead

Our Approach

- We use machine learning to analyze hardware performance event data
- Basic idea: train a classifier with data from *problem-specific* mini-programs
- Develop a set of mini-programs, with 3 possible modes of execution
  - Good (no false sharing, no bad memory access)
  - Bad-FS (with false sharing)
  - Bad-MA (with bad memory access)

Overall Bad Memory Accesses

Introduced a 3rd class of memory references.
Bad memory accesses are due to other types of cache misses.

Differentiate between other cache misses vs. false sharing

```
int psum[MAXTHREADS];
int V1[N], V2[N];

void pdot_3(…) {
    int mysum = 0;
    for(int i=myid*BLKSZ; i < min((myid+1)*BLKSZ, N); i++)
        mysum += V1[permute(i)] * V2[permute(i)];
    psum[myid] = mysum;
}
```

BAD-MA

Performance Events

- "Performance events" can be counted using Performance Monitoring Units (PMU)

- But performance event data
  - can be confusing
  - too much for human processing when large amounts are collected

Example

Sample event counts for fast and slow versions of one program

	Fast Version	Slow Version
Execution Time	1.57 seconds	3.01 seconds
Sample Events	**Event Counts**	**Event Counts**
Resource Stalls (r01a2)	3,947,728,352	8,627,478,887
L3 References (r4f2e)	97,594,129	128,009,158
L3 Misses (412e)	31,202,292	117,648,528
L1D Modif. Evicted (r0451)	108,399,271	109,767,458
DTLB Load Misses (r1008)	1,561,291	610,899
DTLB Store Misses (r1049)	1,207,394	601,354

Machine learning may recognize patterns in such data

Our Methodology

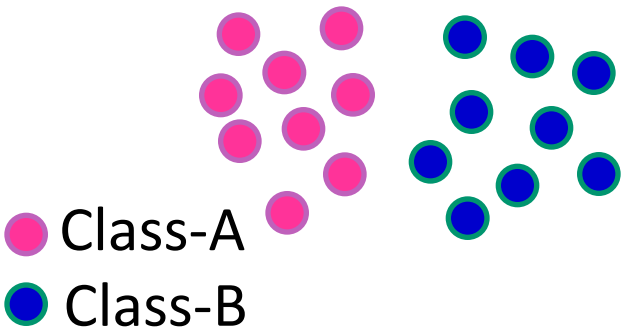
1. Identify a set of performance events
2. Collect performance event counts from problem-specific mini-programs
3. Label data instances as “good”, “bad-fs”, “bad-ma”
4. Train a classifier using these training data

---

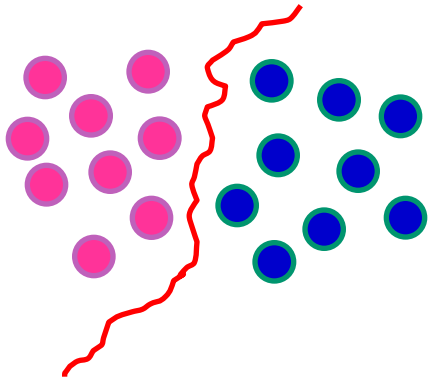
5. Use the trained classifier to classify data from unseen programs

Classification: Training & Testing

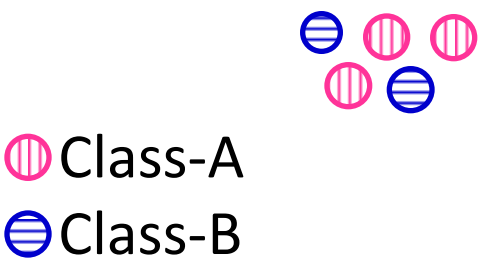
Training data (manually classified)

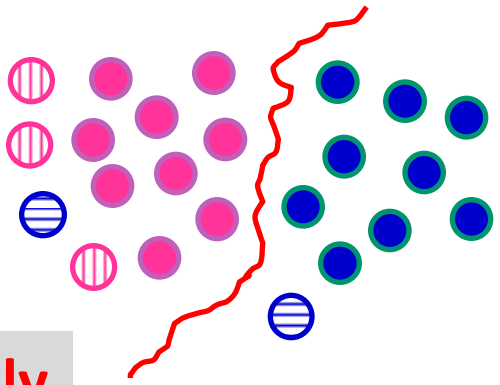


Classified by the Classifier



Testing data (manually classified)



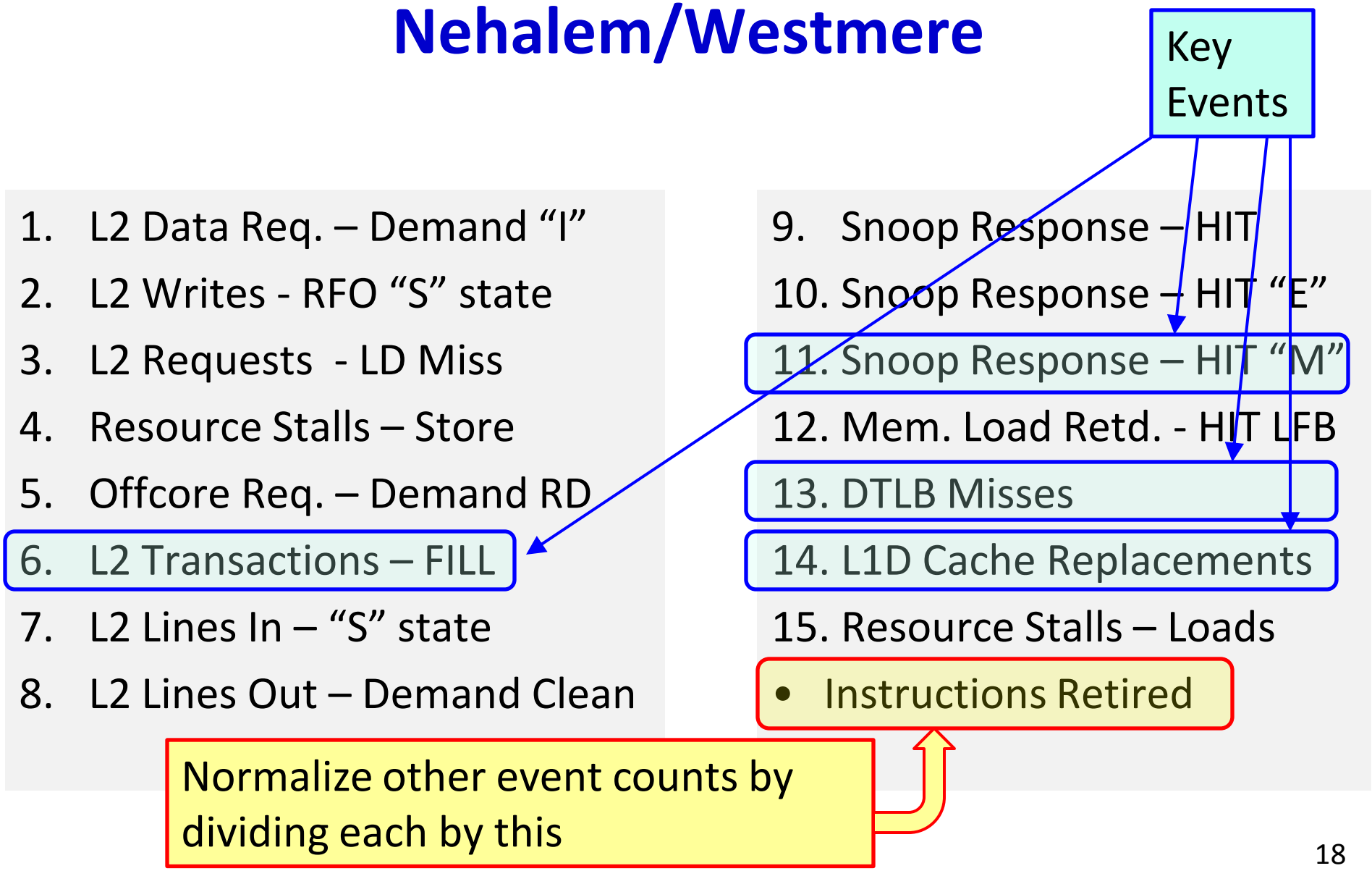


4/5 classified correctly
Correctness = 80%

Problem-Specific Mini-Programs

- Multi-threaded parallel programs
  - 3 scalar programs, 3 vector programs, matrix-multiplication, matrix-compare
  - Parameters: mode, problem size (N), number of threads (T)
- Sequential (single-threaded) programs
  - array access for: read, read-modify-write, write; dot product, matrix multiplication
  - Parameters: mode, problem size (N)

Selected Performance Events for Intel Nehalem/Westmere



Training Data

	good	bad-fs	bad-ma	Total
Part A (Multi-threaded)	324	216	113	653
Part B (Single-threaded)	130	-	97	227
Training Data Set	**454**	**216**	**210**	**880**

In each data instance, each of the 15 event counts is normalized as a (scaled up)
ratio = (event count/# instructions) x 10^9

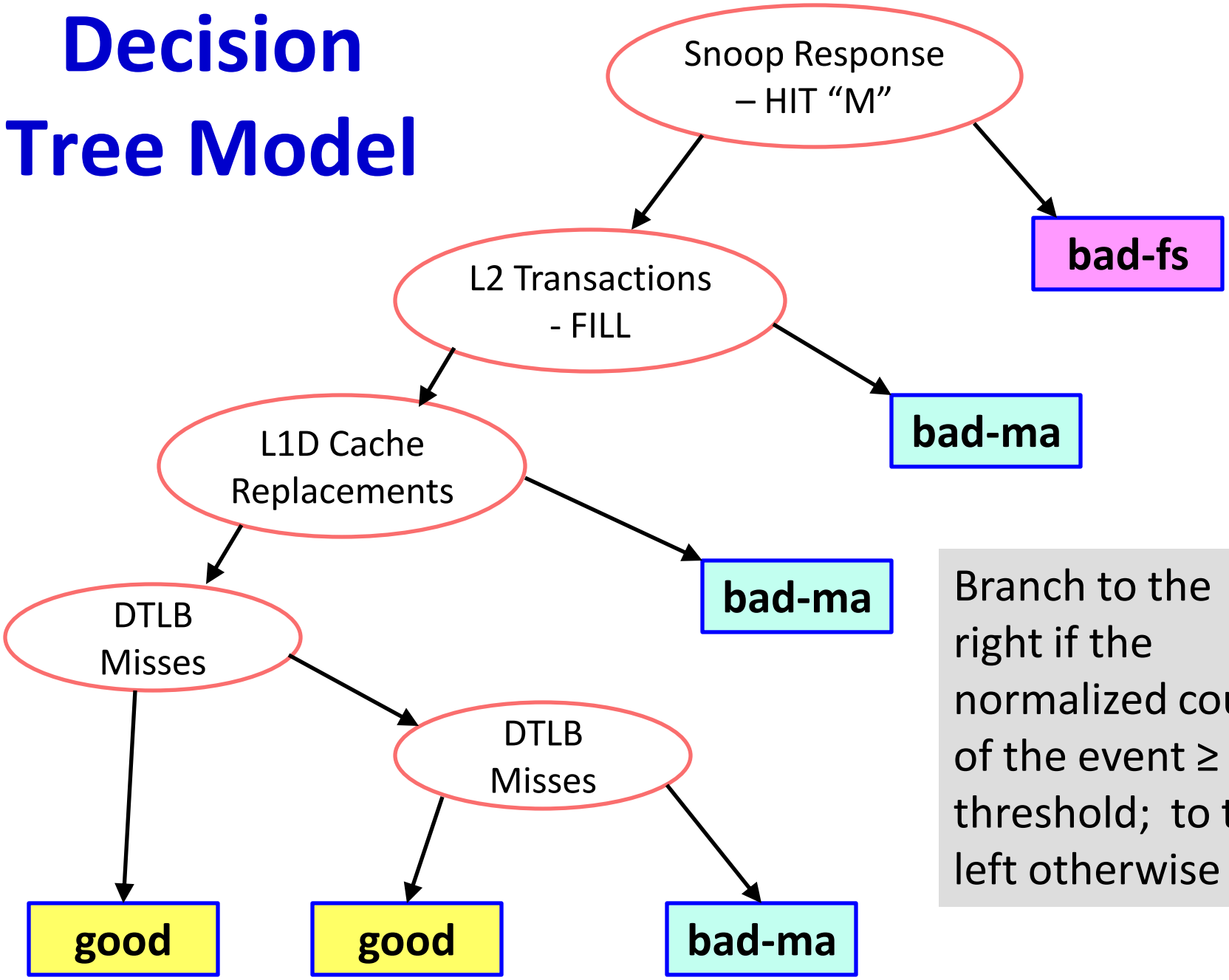
Training & Model Validation

Training Data Set	Decision Tree Model	10-fold stratified Cross-validation: Correct
880 instances (454, 216, 210)	6 leaves 11 nodes	875 99.4%

Using ***J48*** classifier that implements the ***C4.5*** decision-tree algorithm

		Predicted Class		
		good	**bad-fs**	**bad-ma**
Actual Class	**good**	453	1	0
	bad-fs	0	216	0
	bad-ma	4	0	206

Decision Tree Model



Branch to the right if the normalized count of the event ≥ a threshold; to the left otherwise

Results: Detection of False Sharing in Phoenix and PARSEC Benchmarks

Experimental setup:

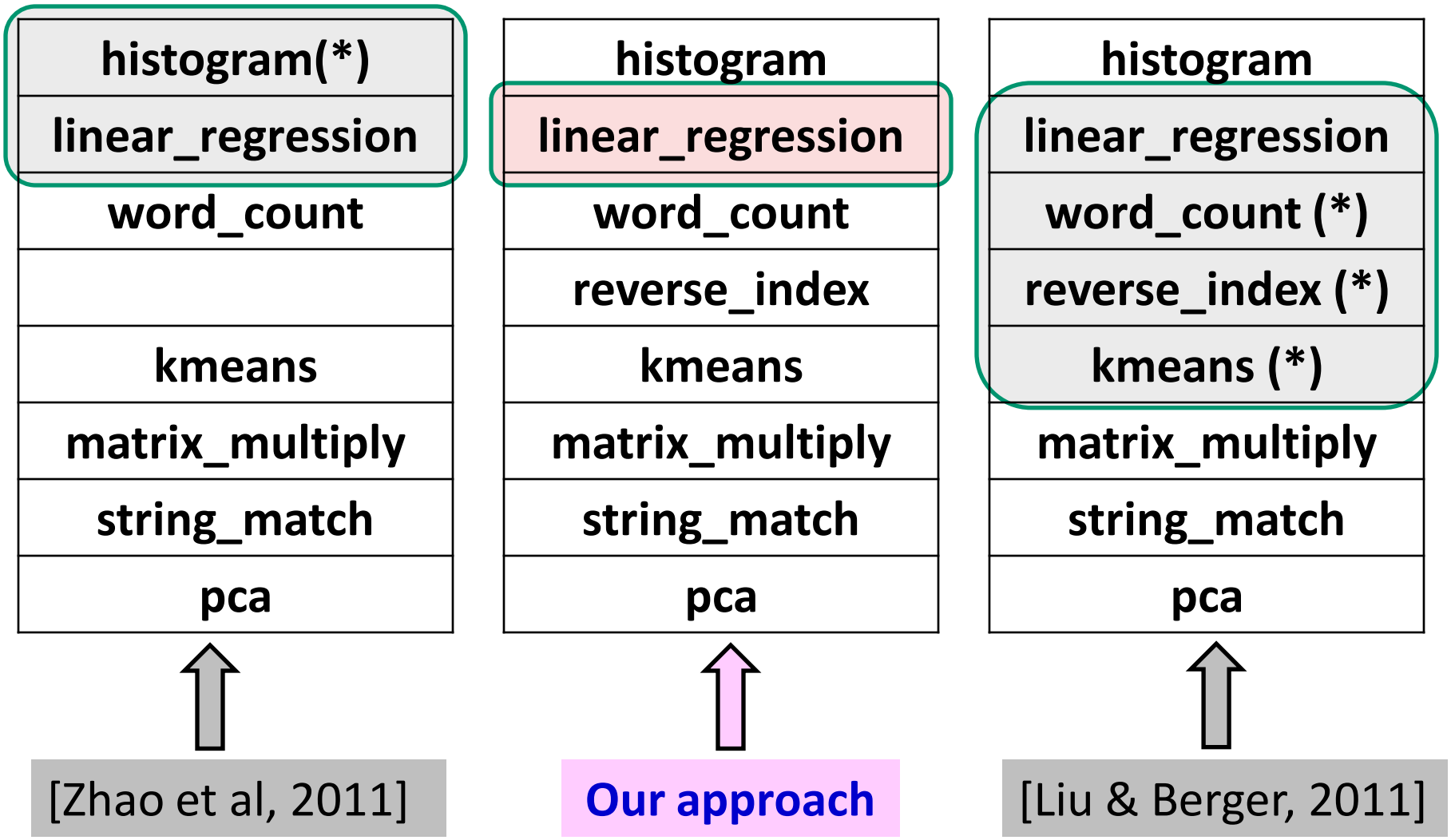
2x 6-core (total 12-core) Intel Xeon X5690 @3.47GHz, 192 GB RAM, Linux x86_64

Our Detection of False Sharing in Phoenix and PARSEC Benchmarks

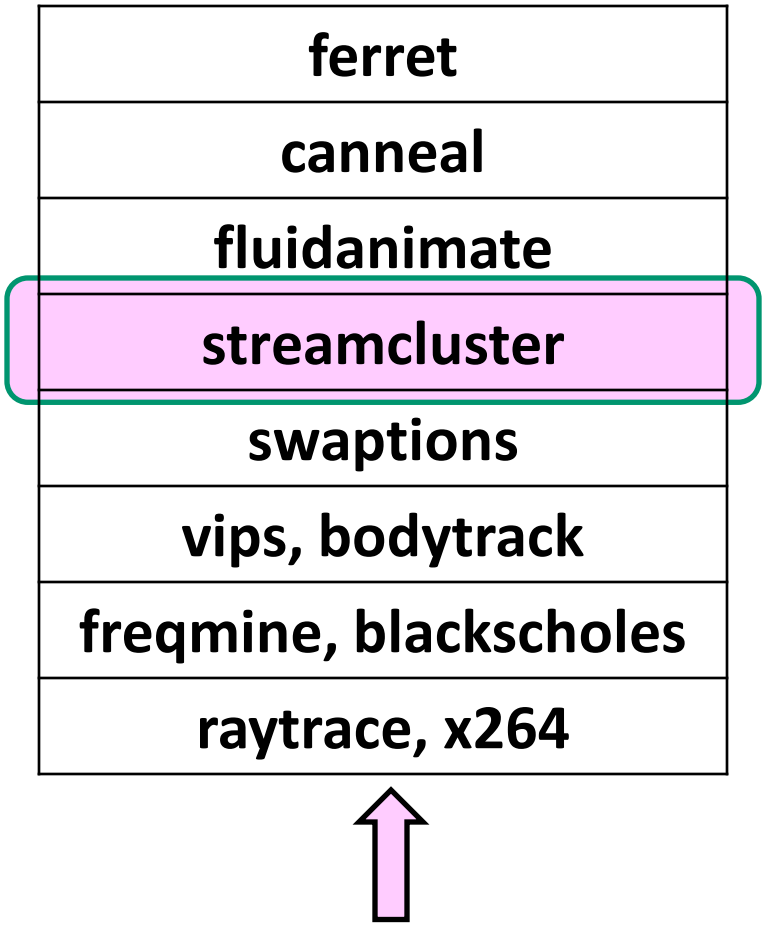
Phoenix		PARSEC	
histogram	No	ferret	No
linear_regression	**Yes**	canneal	No
word_count	No	fluidanimate	No
reverse_index	No	**streamcluster**	**Yes**
kmeans	No	swaptions	No
matrix_multiply	No	vips	No
string_match	No	bodytrack	No
pca	No	freqmine	No
		blackscholes	No
		raytrace	No
		x264	No

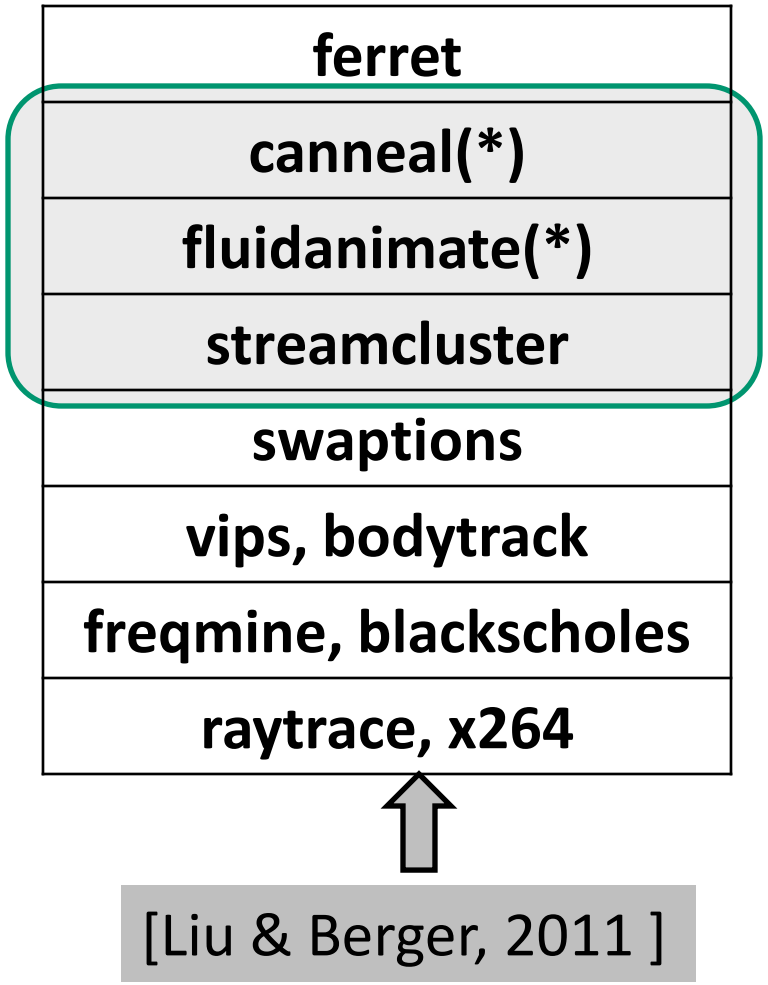
Each program had multiple cases (by varying inputs, # of threads, compiler optimization); the above is based on the majority result.

Phoenix: Comparison With Other Work



PARSEC : Comparison With Other Work





* Indicates false sharing would not have a significant impact

Verification of Our Detection of False Sharing: Phoenix Benchmarks

Benchmark	# cases	Actual		Detected	
		FS	No FS	FS	No FS
histogram	18	0	18	0	18
linear_regression	**18**	**18**	**0**	**12**	**06**
word_count	18	0	18	0	18
reverse_index	06	0	06	0	06
kmeans	12	0	12	0	12
matrix_multiply	18	0	18	0	18
string_match	18	0	18	0	18
pca	18	0	18	0	18
Subtotal	**126**	**18**	**108**	**12**	**114**

Verification is by the approach of [Zhao et al, 2011], on which the "Actual" columns are based

Verification of Our Detection of False Sharing: PARSEC Benchmarks

Benchmark	# cases	Actual		Detected	
		FS	No FS	FS	No FS
ferret	18	0	18	0	18
canneal	18	0	18	0	18
fluidanimate	18	0	18	0	18
streamcluster	**18**	**11**	**07**	**10**	**08**
swaptions	18	0	18	0	18
vips	18	0	18	0	18
bodytrack	18	0	18	0	18
freqmine	16	0	16	0	16
blackscholes	18	0	18	0	18
raytrace	18	0	18	0	18
x264	18	0	18	0	18
Total (Overall)	**322**	**29**	**293**	**22**	**300**

Summary: Verification of Our Detection of False Sharing

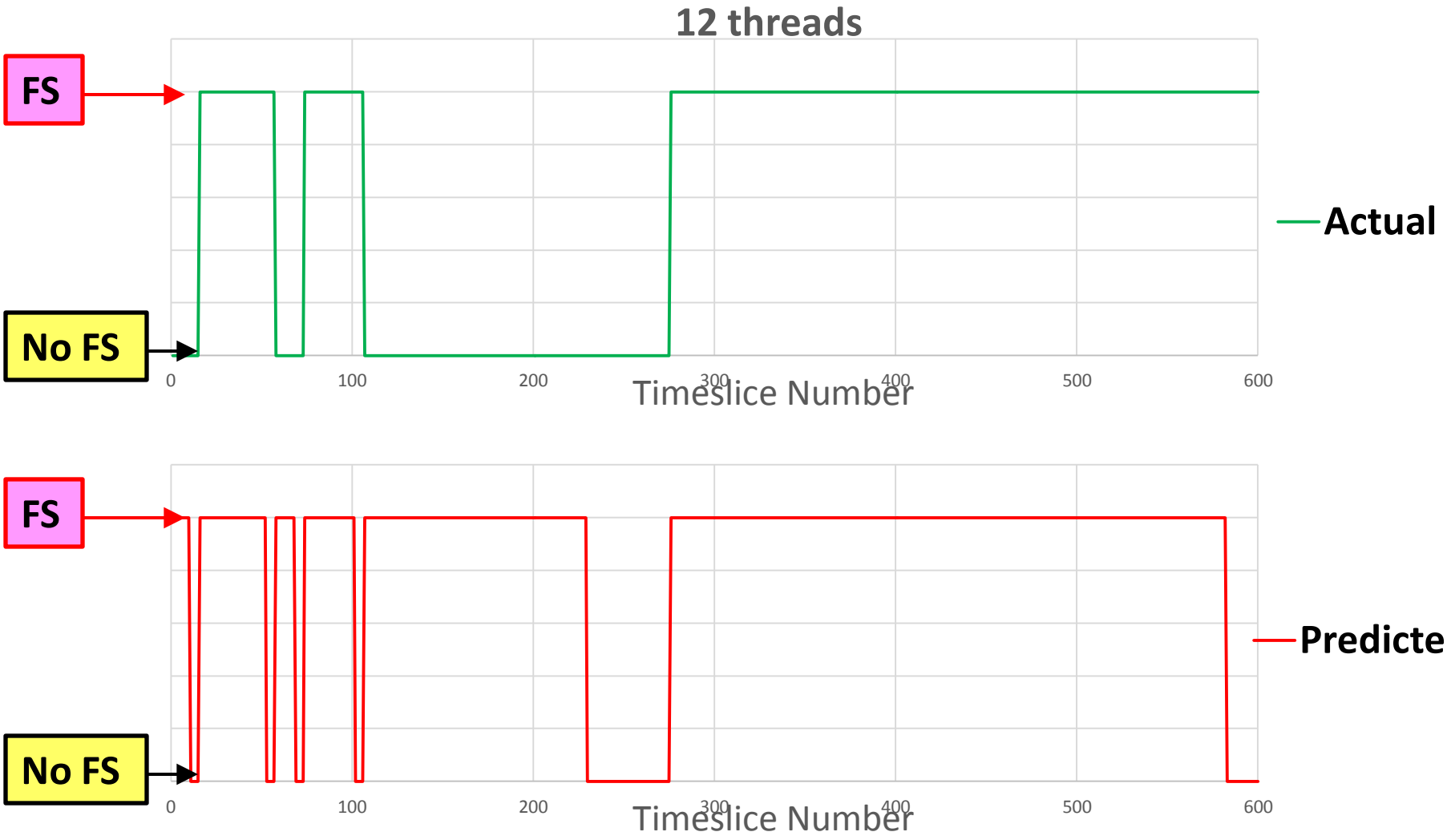
		Detection (Our Classification)	
		FS	**No FS**
Actual	**FS**	22	7
	No FS	0	293
Correctness		(22+293)/(22+7+0+293) = 97.8%	
False Positive Rate		0/(293+0) = 0%	

Verification is by the approach of [Zhao et al, 2011], on which the "Actual" values are based

Ongoing/Future Work

- Finer granularity in false sharing detection for long running programs
  - Time slicing
  - Subroutine-wise
- Extending to IBM Power platform
- Other forms of performance problems related to memory access (locks, true sharing)

Preliminary Results: Detection in Time-Slices



Conclusions

- False sharing can seriously degrade performance yet it is difficult to detect
- We presented an efficient and effective approach
  - Minimal performance overhead (< 2%), and easy to apply
- In PARSEC and Phoenix benchmarks, we detect all programs where false sharing exists with 0 false positives

Acknowledgements

- Anonymous reviewers
- Part of this work was done when the first author was spending sabbatical at the Massachusetts Institute of Technology
- This work was partially supported by:
  - DOE award DE-SC0005288
  - DOD DARPA award HR0011-10-9-0009
  - NSF awards CCF-0632997, CCF-0811724
  - Open Collaborative Research (OCR) program from IBM

Q & A

Extra Slides

Selection of Performance Events

- Identify a candidate set **C** of events
  - Related to: memory access, data caches, TLBs, interaction among CPU cores, resource stalls…
  - 60-70 events for Intel Nehalem, Westmere
- Use the mini-programs to select a relevant subset **R** from **C**
  - Select events that can distinguish good and bad modes for most mini-programs
  - 15 events for Intel Nehalem/Westmere

Training data: Part A (Multi-threaded)

Program	N count	Threads	Good	Bad-FS	Bad-MA	Total
pdot	7	1,3,6,9, 12	42	28	35	105
psums	7		42	28	-	70
psumv	7		42	28	35	105
count	7		42	28	35	105
padding	7		42	28	-	70
false1	7		42	28	-	70
pmatcom	6		36	24	30	90
pmatmul	6		36	24	-	60
Total	**Part A**		**324**	**216**	**113**	**653**

Training data: Part B (single-threaded)

Program	N count	Threads	Good	Bad-FS	Bad-MA	Total
read	9	1	9	-	18	27
write	9		9	-	18	27
read-mod-write	9		9	-	18	27
dotp	9		9	-	18	27
matmult	14		135	-	28	163
Total	**Part B**		**130**	**-**	**97**	**227**

- Intel docs say to look for the high count of "MEM_UNCORE_RETIRED.OTHER_CORE_L2_HITM" event as an indication of possible false sharing
- Ref:
  - *Avoiding and Identifying False Sharing Among Threads*, http://software.intel.com/en-us/articles/avoiding-and-identifying-false-sharing-among-threads

Decision Tree for IBM Power 7