Accuracy Bugs:
A New Class of Concurrency Bugs to Exploit Algorithmic Noise Tolerance

Ismail Akturk[1], Riad Akram[2], Mohammad M. Islam[2], Abdullah Muzahid[2], Ulya R. Karpuzcu[1]

1 University of Minnesota, Twin Cities

2 University of Texas at San Antonio

01/25/2017

Outline

- Background
- Accuracy Bugs
- Methodology
- Evaluation

Background

- Parallel programming is hard
  - Correctness: error-free execution
  - Concurrency bugs
- Applications having algorithmic error tolerance (Recognition, Mining and Synthesis)
  - Massive data (noisy, redundant)
  - Iterative/probabilistic algorithm
  - More than one valid output set (no single golden/precise result)
- Can we exploit the error tolerance?
  - Ease programming
  - Utilize approximate hardware

Accuracy Bugs

- Concurrency Bugs
  - data races
  - ordering violations
  - atomicity violations
  - deadlocks
- A new class of Concurrency Bug: Accuracy bugs
  - do not lead to program failures (Lu et al. 2008)
  - manifest themselves as inaccuracy in outputs (do not give up correctness!)
  - comprise the subset of concurrency bugs that mainly affect the dataflow

Accuracy Bugs: How they manifest themselves?

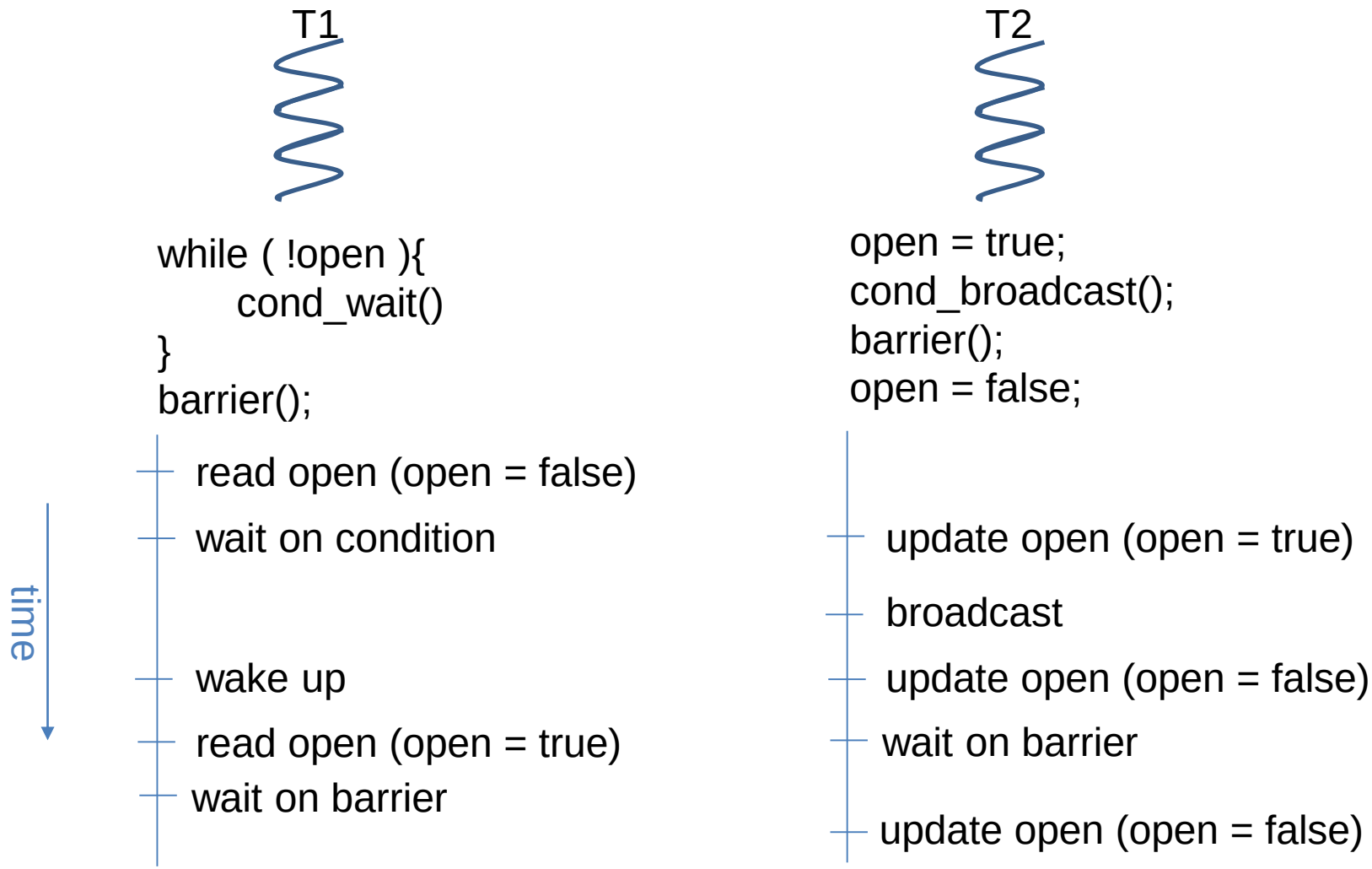
T1

```
while ( !open ){
      cond_wait()
}
barrier();
```

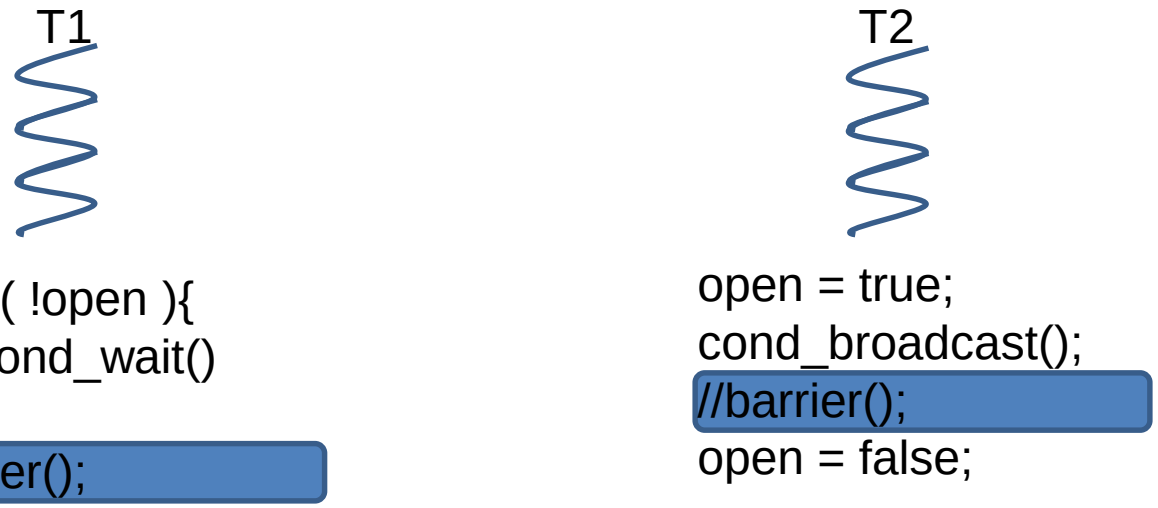
T2

```
open = true;
cond_broadcast();
barrier();
open = false;
```

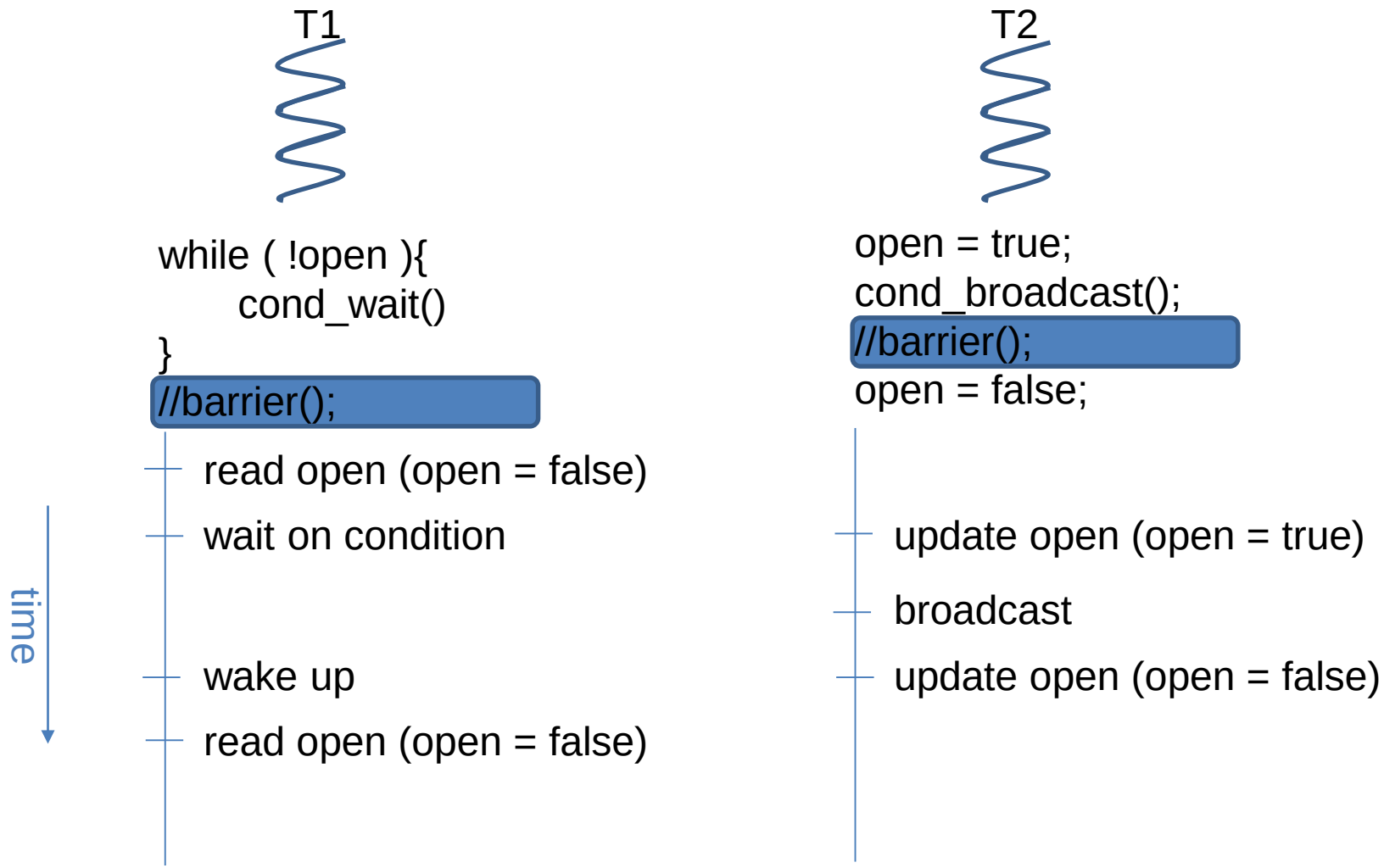
Accuracy Bugs: How they manifest themselves?



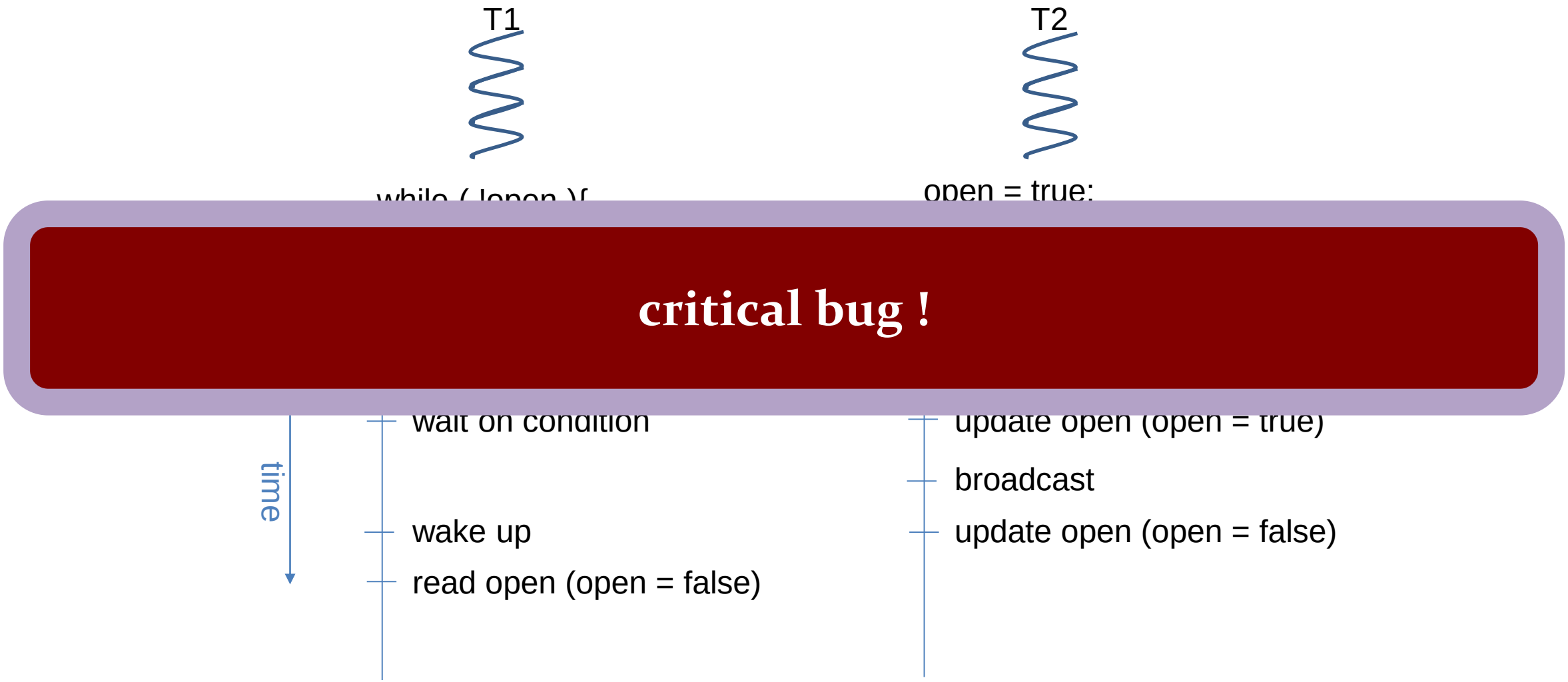
Accuracy Bugs: How they manifest themselves?



Accuracy Bugs: How they manifest themselves?



Accuracy Bugs: How they manifest themselves?

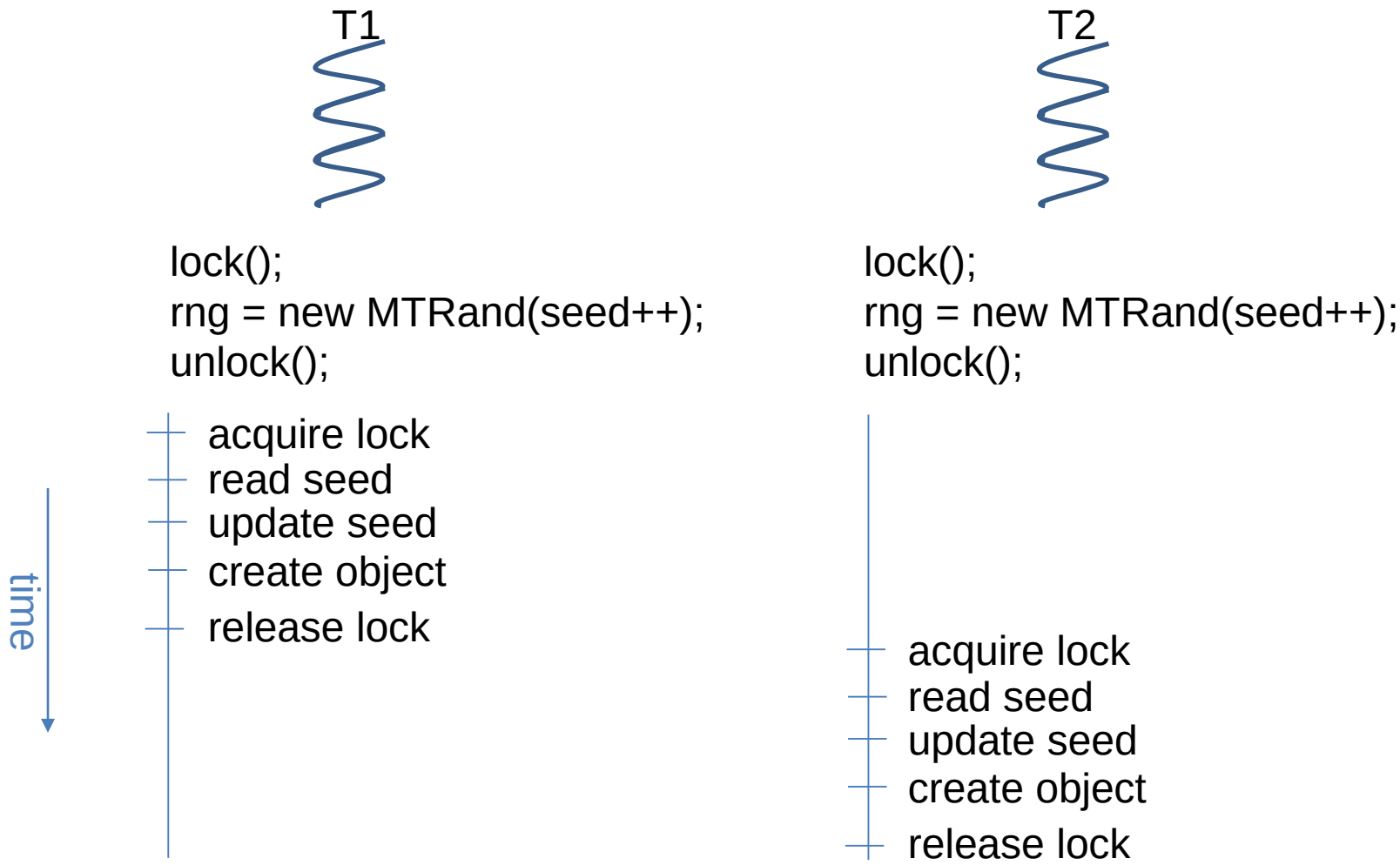


Accuracy Bugs: How they manifest themselves?

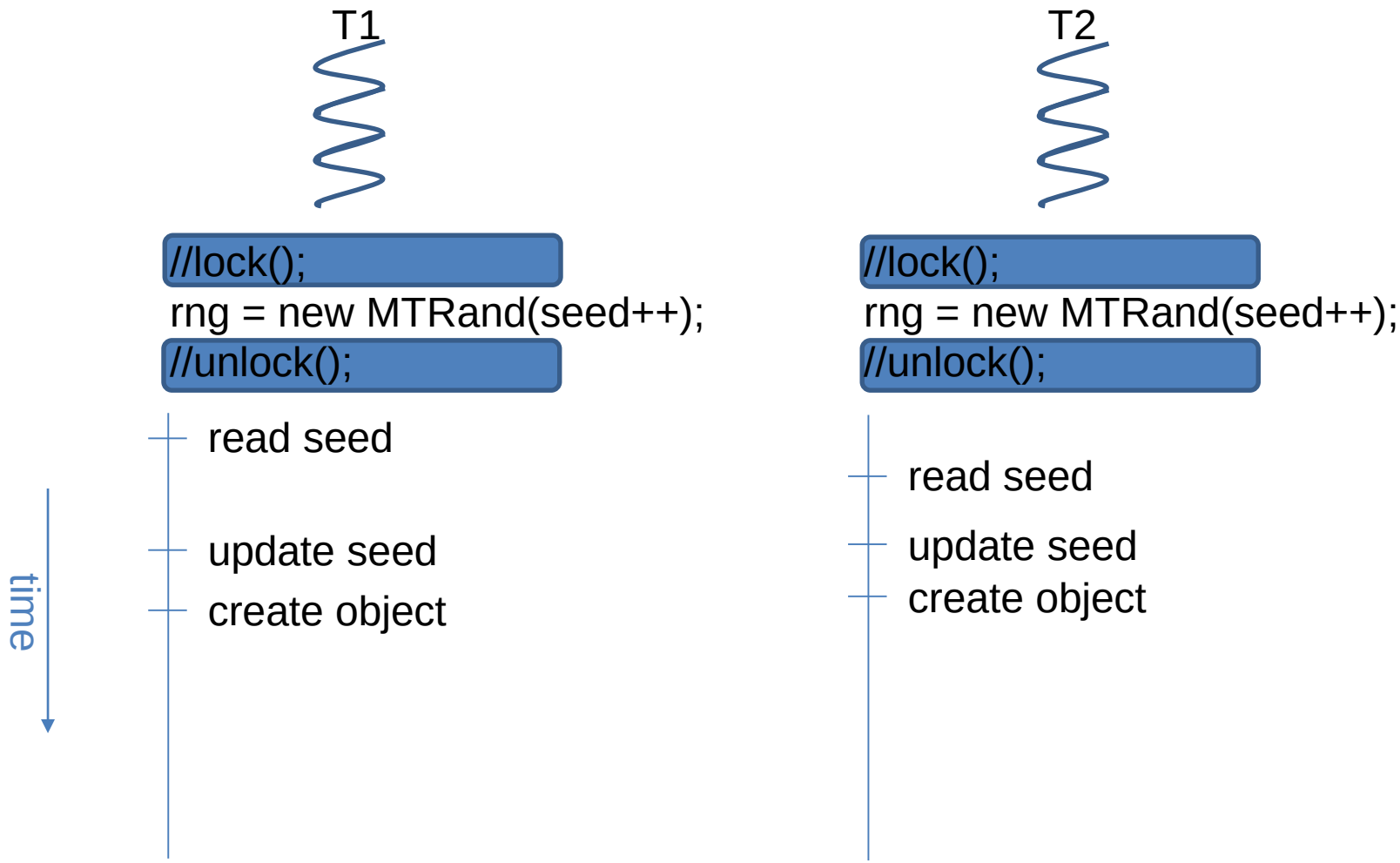

```
lock();
rng = new MTRand(seed++);
unlock();
```


```
lock();
rng = new MTRand(seed++);
unlock();
```

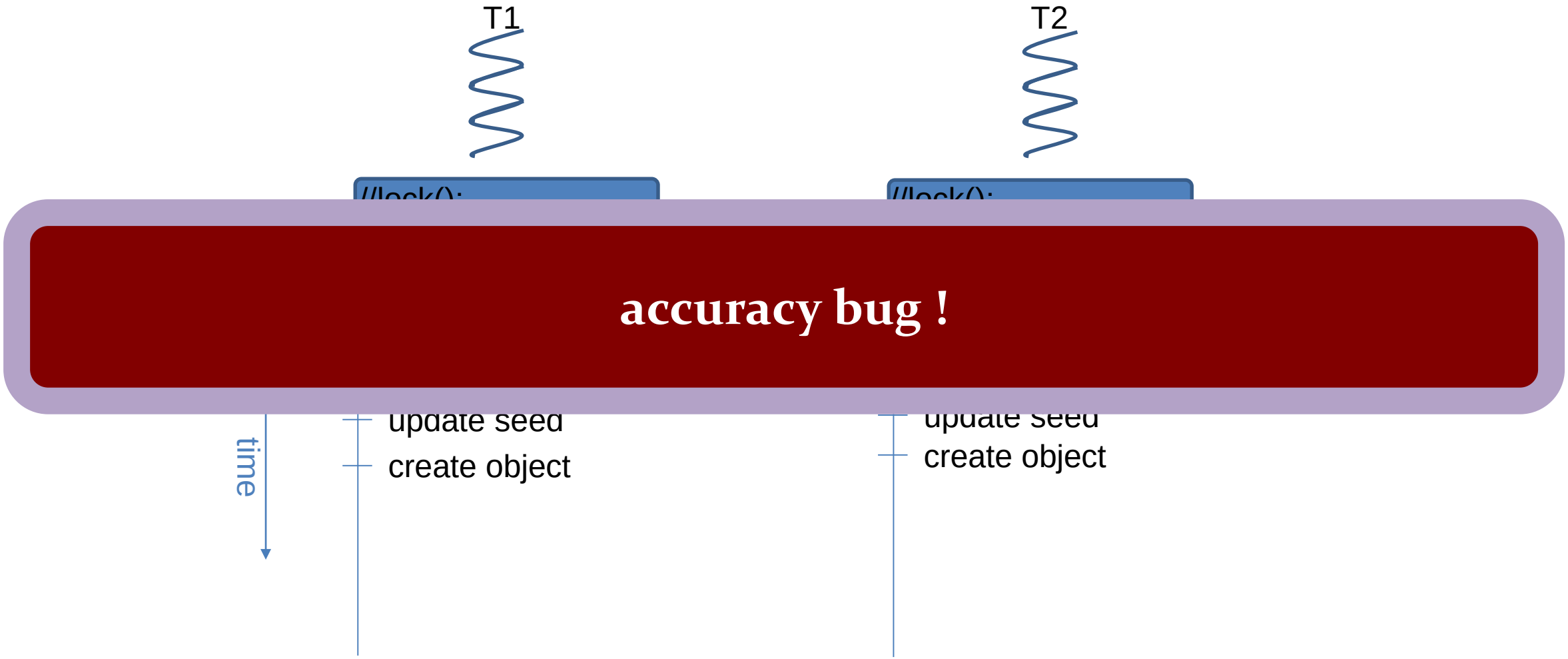
Accuracy Bugs: How they manifest themselves?



Accuracy Bugs: How they manifest themselves?



Accuracy Bugs: How they manifest themselves?



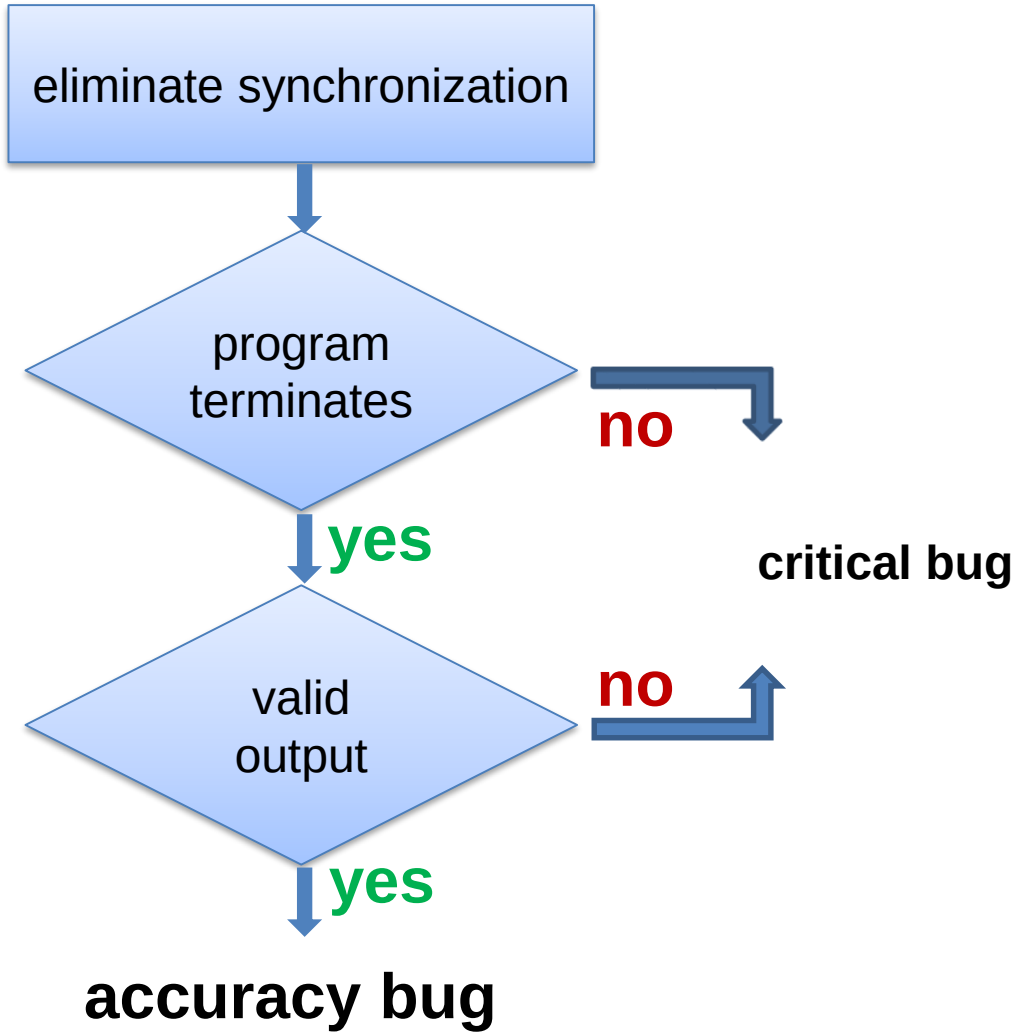
Accuracy Bugs: Can we accept all of them?

- The amount of degradation (i.e. inaccuracy) in output determines acceptability
  - It is domain specific
  - The same amount of degradation may be acceptable in one domain, but not in another

Accuracy Bugs: Is it only the accuracy?

- Performance may increase or decrease
  - Convergence criteria in iterative refinement algorithms
- System complexity
  - Coherence and consistency

Methodology



Evaluation: Setup

- Parsec and Splash benchmarks
- Microarchitectural simulation backed up with real system experimentation
  - Snipersim
  - Intel Xeon E5-4620 v2 processors (4 sockets, 8 cores/socket)
- Quality metrics
  - Scalar: relative change
  - Vector: sum of square difference (SSD)
  - Image: peak signal-to-noise ratio (PSNR) / structural similarity (SSIM)
  - Clustering/Similarity: difference of common elements

Evaluation: Synchronization points

Benchmark	Total # of synchronizations	Quality degradation bin					
		0%	< 1%	< 50%	< 100%	fault	invalid
barnes	12	3	4	1	1	3	0
canneal	11	0	7	3	0	0	1
dedup	9	3	1	0	0	1	4
fluidanimate	16	9	1	0	0	4	2
streamcluster	27	5	0	1	5	16	0
ferret	3	1	0	0	0	2	0
bodytrack	31	22	0	0	0	9	0
vips	15	11	0	0	0	4	0
raytrace	8	5	0	0	0	3	0
x264	2	0	1	0	0	1	0

Evaluation: Synchronization points

Benchmark	Total # of synchronizations	Quality degradation bin					
		0%	< 1%	< 50%	< 100%	fault	invalid
barnes	12	3	4	1	1	3	0
canneal	11	0	7	3	0	0	1
dedup	9	3	1	0	0	1	4
fluidanimate	16	9	1	0	0	4	2
streamcluster	27	5	0	1	5	16	0
ferret	3	1	0	0	0	2	0
bodytrack	31	22	0	0	0	9	0
vips	15	11	0	0	0	4	0
raytrace	8	5	0	0	0	3	0
x264	2	0	1	0	0	1	0

accuracy bugs

Evaluation: Synchronization points

Benchmark	Total # of synchronizations	Quality degradation bin					
		0%	< 1%	< 50%	< 100%	fault	invalid
barnes	12	3	4	1	1	3	0
canneal	11	0	7	3	0	0	1
dedup	9	3	1	0	0	1	4
fluidanimate	16	9	1	0	0	4	2
streamcluster	27	5	0	1	5	16	0
ferret	3	1	0	0	0	2	0
bodytrack	31	22	0	0	0	9	0
vips	15	11	0	0	0	4	0
raytrace	8	5	0	0	0	3	0
x264	2	0	1	0	0	1	0

accuracy bugs

critical bugs

Evaluation: Synchronization points

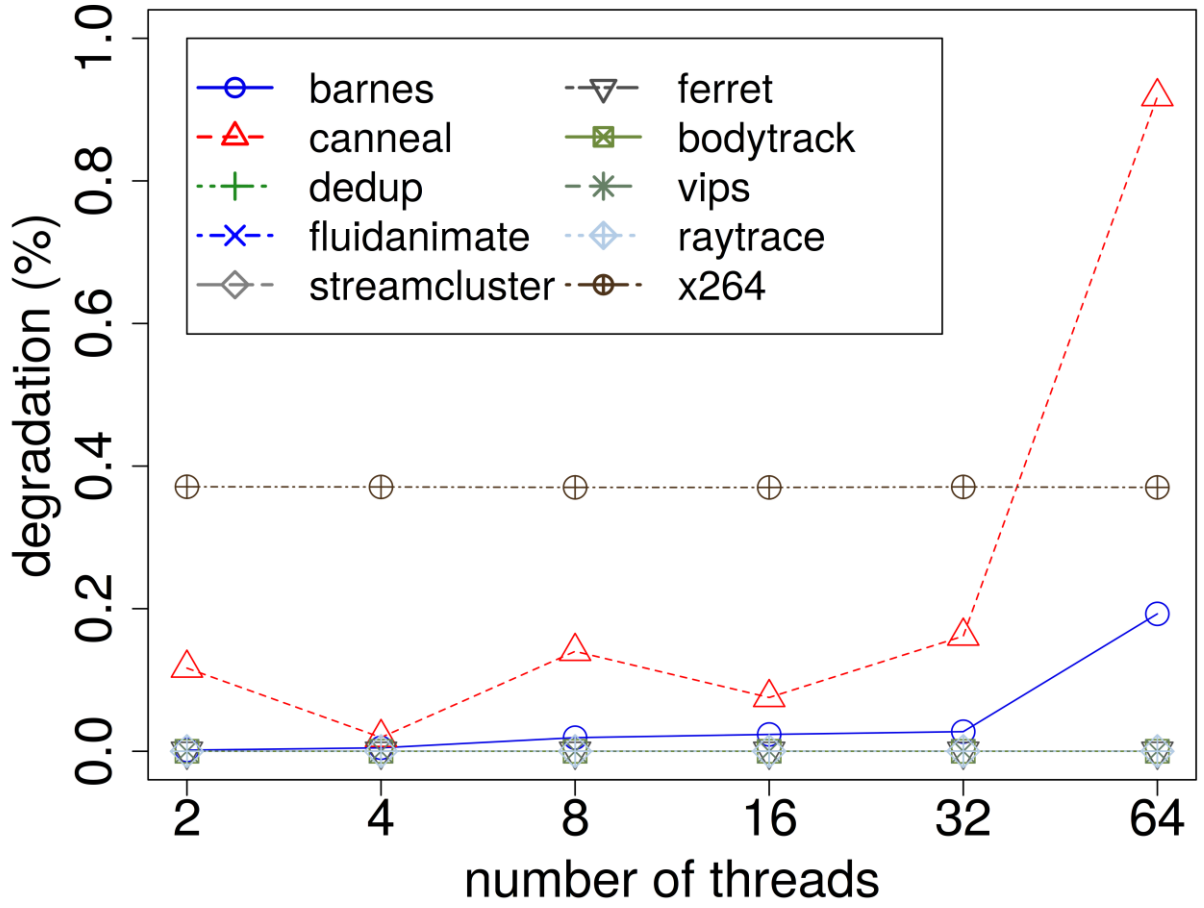
Benchmark	Total # of synchronizations	Quality degradation bin					
		0%	< 1%	< 50%	< 100%	fault	invalid
barnes	12	3	4	1	1	3	0
canneal	11	0	7	3	0	0	1
ferret	3	1	0	0	0	2	0
bodytrack	31	22	0	0	0	9	0
vips	15	11	0	0	0	4	0
raytrace	8	5	0	0	0	3	0
x264	2	0	1	0	0	1	0

84 out of 134 (62%) synchronizations lead to accuracy bugs

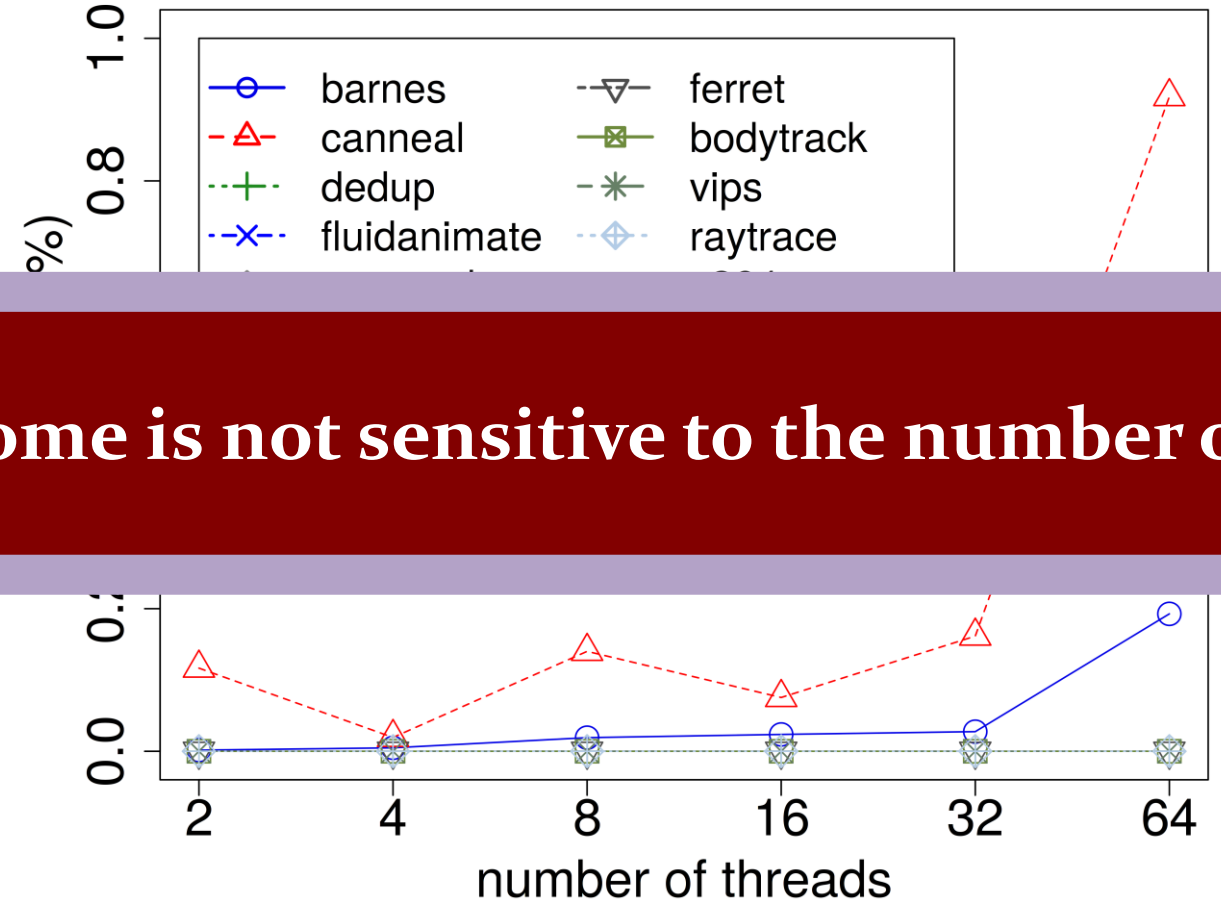
accuracy bugs

critical bugs

Evaluation: Output quality vs. Thread count

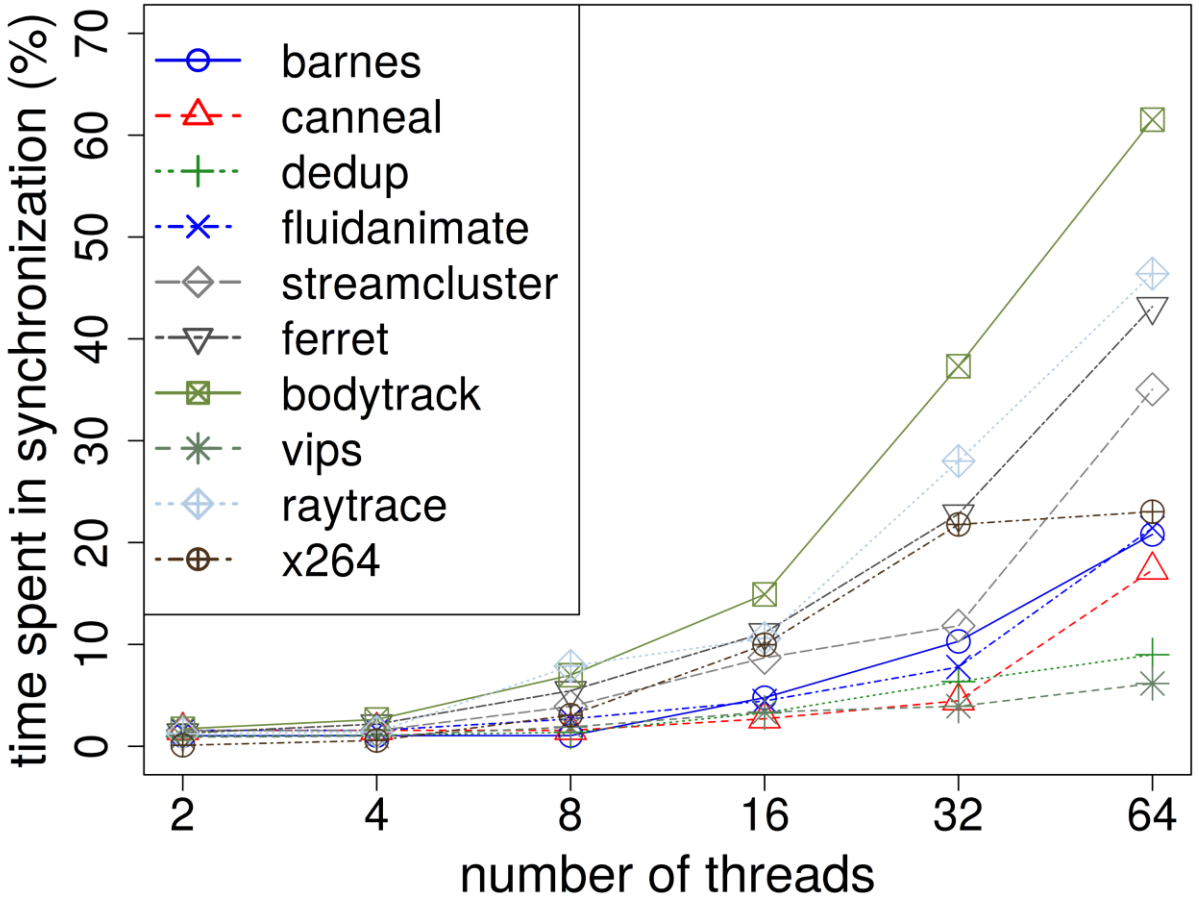


Evaluation: Output quality vs. Thread count

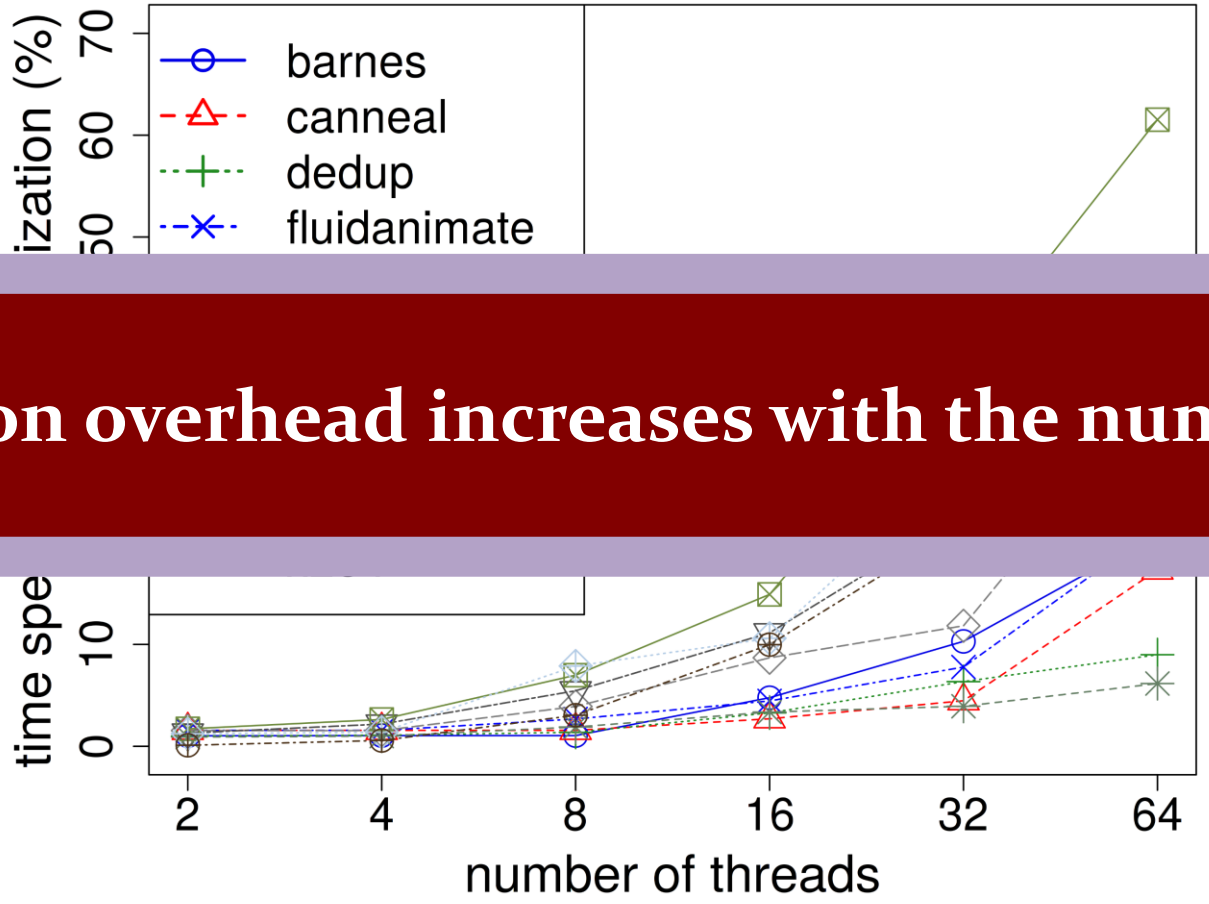


the outcome is not sensitive to the number of threads

Evaluation: Time spent in synchronization

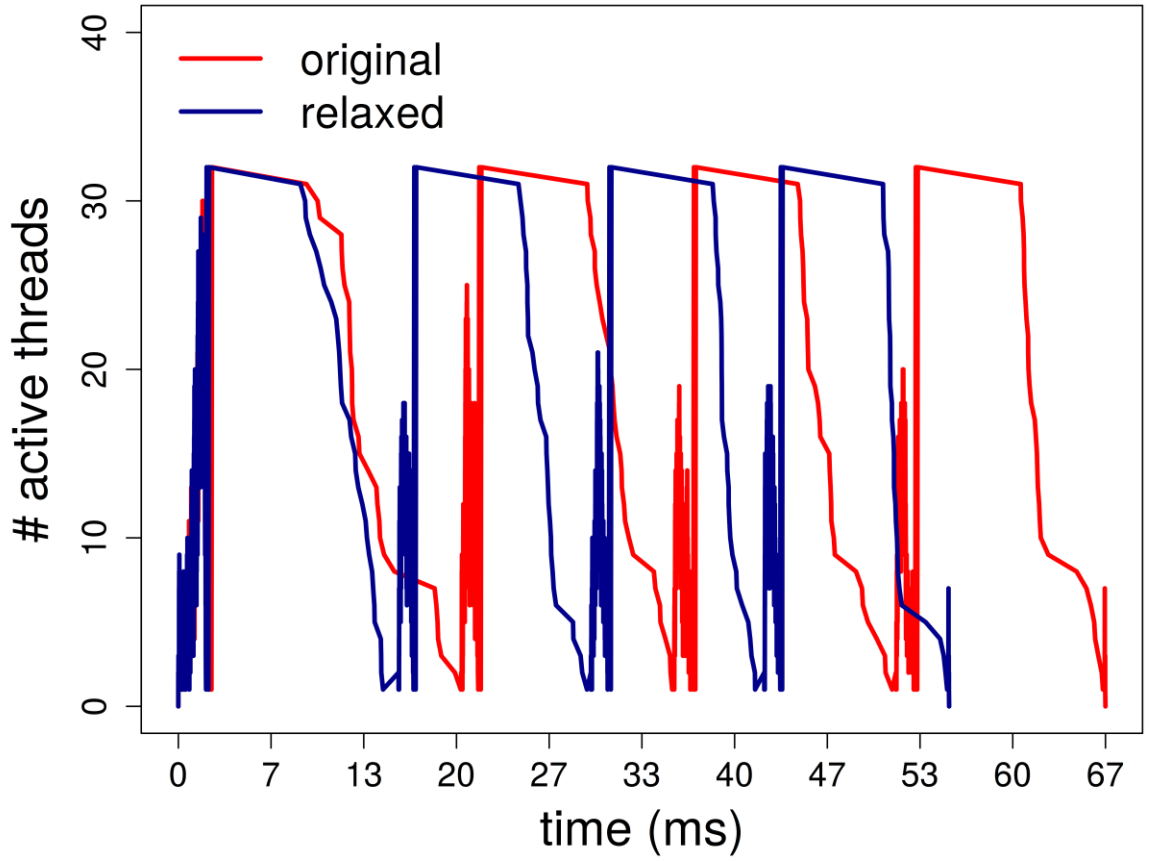


Evaluation: Time spent in synchronization

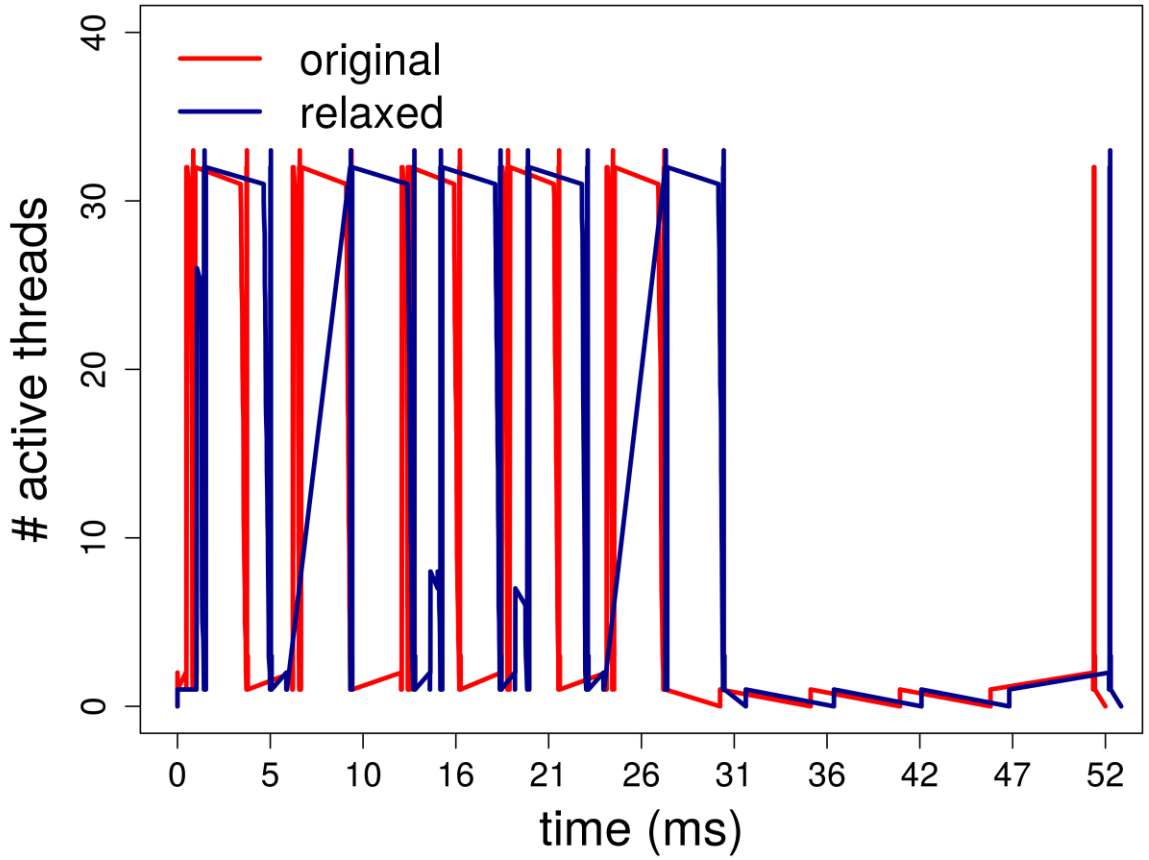


synchronization overhead increases with the number of threads

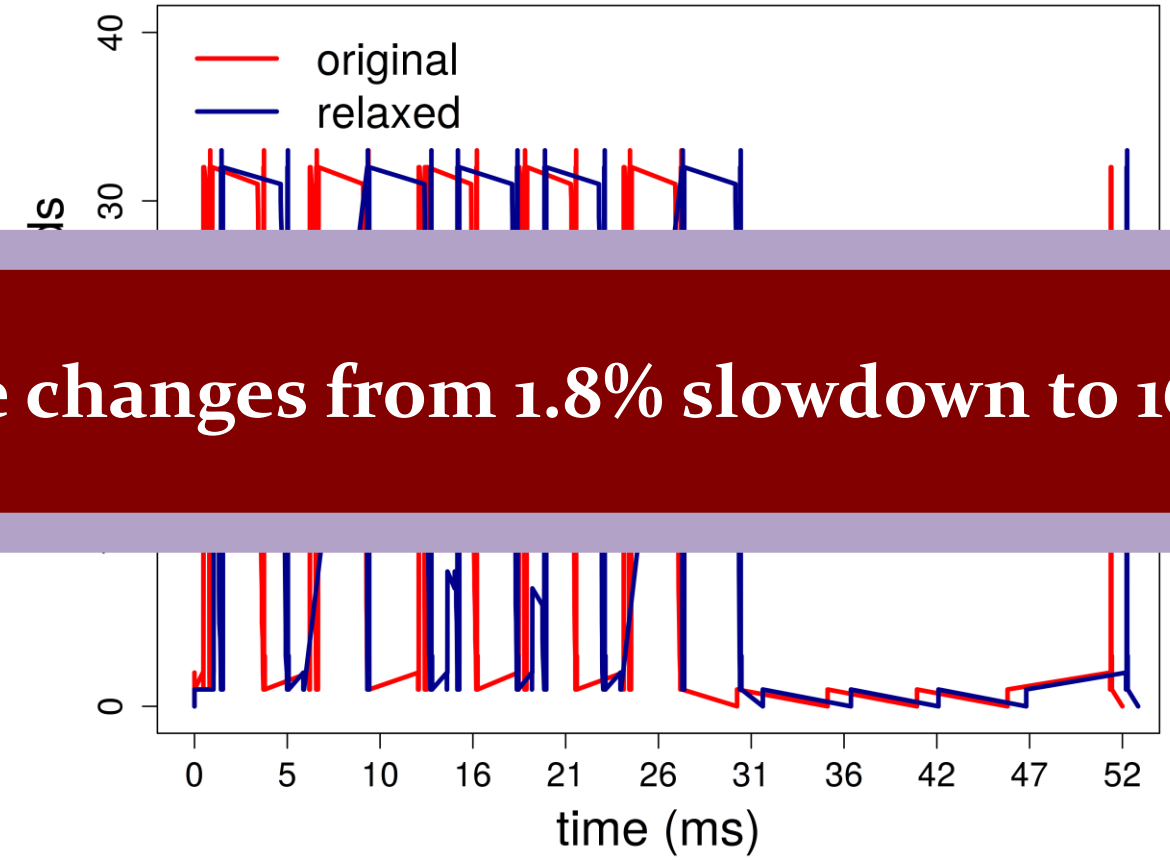
Evaluation: Impact on performance - barnes



Evaluation: Impact on performance - bodytrack



Evaluation: Impact on performance - bodytrack



Performance changes from 1.8% slowdown to 16.9% speedup

Evaluation: Bug categorization

Bug Injection	Data race		Atomicity Violation		Ordering Violation		Other	
	critical	accuracy	critical	accuracy	critical	accuracy	critical	accuracy
Lock Elimination	12	93	6	41	0	0	2	23
Barrier Elimination	30	52	4	19	0	0	3	15
Condition Elimination	0	0	0	0	7	7	4	2
Lock Splitting	0	0	60	144	0	0	0	1
Atomicity Elimination	0	0	0	0	0	0	1	7
Total	**42**	**145**	**70**	**204**	**7**	**7**	**10**	**48**

Evaluation: Bug categorization

Bug Injection	Data race		Atomicity Violation		Ordering Violation		Other	
	critical	accuracy	critical	accuracy	critical	accuracy	critical	accuracy
Lock Elimination	12	93	6	41	0	0	2	23
F								
Total	**42**	**145**	**70**	**204**	**7**	**7**	**10**	**48**

404 out of 533 (76%) concurrency bugs are accuracy bugs

Summary

Summary

- Most of the synchronizations are noncritical
  - their relaxation introduces less than 10% inaccuracy
  - most of the injected bugs are accuracy related

Summary

- Most of the synchronizations are noncritical
  - their relaxation introduces less than 10% inaccuracy
  - most of the injected bugs are accuracy related
- Performance gain is not significant at a lower thread count
  - potential gain increases significantly at a higher thread count: Amdahl's Law

Summary

- Most of the synchronizations are noncritical
  - their relaxation introduces less than 10% inaccuracy
  - most of the injected bugs are accuracy related
- Performance gain is not significant at a lower thread count
  - potential gain increases significantly at a higher thread count: Amdahl's Law
- Findings should be considered with set of applications (and alike) and the evaluation methodology presented in this study

Accuracy Bugs:
A New Class of Concurrency Bugs to Exploit Algorithmic Noise Tolerance

Ismail Akturk[1], Riad Akram[2], Mohammad M. Islam[2], Abdullah Muzahid[2], Ulya R. Karpuzcu[1]

1 University of Minnesota, Twin Cities

2 University of Texas at San Antonio

01/25/2017

Case Study: streamcluster – critical bug

```
1  if(pid == 0){
2    work_mem = (double *) malloc (...);
3    ...
4  }
5  pthread_barrier_wait(barrier);
6  ...
7  work_mem[pid*stride] = count;
8  ...
```

Case Study: streamcluster – critical bug

```
1  if(pid == 0){
2    work_mem = (double *) malloc (...);
3    ...
4  }
5  pthread_barrier_wait(barrier);
6  ...
7  work_mem[pid*stride] = count;
8  ...
```

Case Study: streamcluster – critical bug

```
1  if(pid == 0){
2    work_mem = (double *) malloc (...);
3    ...
4  }
5  pthread_barrier_wait(barrier);
6  ...
7  work_mem[pid*stride] = count;
8  ...
```

T0

Case Study: streamcluster – critical bug

```
1   if(pid == 0){                                  <-- T0
2     work_mem = (double *) malloc (...);
3     ...
4   }
5   pthread_barrier_wait(barrier);
6   ...
7   work_mem[pid*stride] = count;                  <-- T1
8   ...
```

Case Study: streamcluster – critical bug

```
1   if(pid == 0){                                  <-- T0
2     work_mem = (double *) malloc (...);
3     ...
4   }
5   pthread_barrier_wait(barrier);
6   ...
7   work_mem[pid*stride] = count;                  <-- T1   segmentation fault
8   ...
```

Case Study: streamcluster – accuracy bug

```
1   while( change/cost > 1.0*e){
2     …
3     if(pid == 0){
4       intshuffle(feasible, numfeasible);
5     }
6     pthread_barrier_wait(barrier);
7     …
8     change += pgain(feasible[x], ...);
9     …
10    cost -= change;
11    …
12  }
```

Case Study: streamcluster – accuracy bug

```
1   while( change/cost > 1.0*e){
2     …
3     if(pid == 0){
4       intshuffle(feasible, numfeasible);
5     }
6     pthread_barrier_wait(barrier);
7     …
8     change += pgain(feasible[x], ...);
9     …
10    cost -= change;
11    …
12  }
```

Case Study: streamcluster – accuracy bug

```
1   while( change/cost > 1.0*e){
2     …
3     if(pid == 0){
4       intshuffle(feasible, numfeasible);
5     }
6     pthread_barrier_wait(barrier);
7     …
8     change += pgain(feasible[x], ...);
9     …
10    cost -= change;
11    …
12  }
```

T0

Case Study: streamcluster – accuracy bug

```
1   while( change/cost > 1.0*e){
2     …
3     if(pid == 0){
4       intshuffle(feasible, numfeasible);      <-- T0
5     }
6     pthread_barrier_wait(barrier);
7     …
8     change += pgain(feasible[x], …);          <-- T1
9     …
10    cost -= change;
11    …
12  }
```

Case Study: streamcluster – accuracy bug

```
1   while( change/cost > 1.0*e){
2     …
3     if(pid == 0){
4       intshuffle(feasible, numfeasible);      <- T0
5     }
6     pthread_barrier_wait(barrier);
7     …
8     change += pgain(feasible[x], …);
9     …
10    cost -= change;                           <- T1
11    …
12  }
```

Case Study: streamcluster – accuracy bug

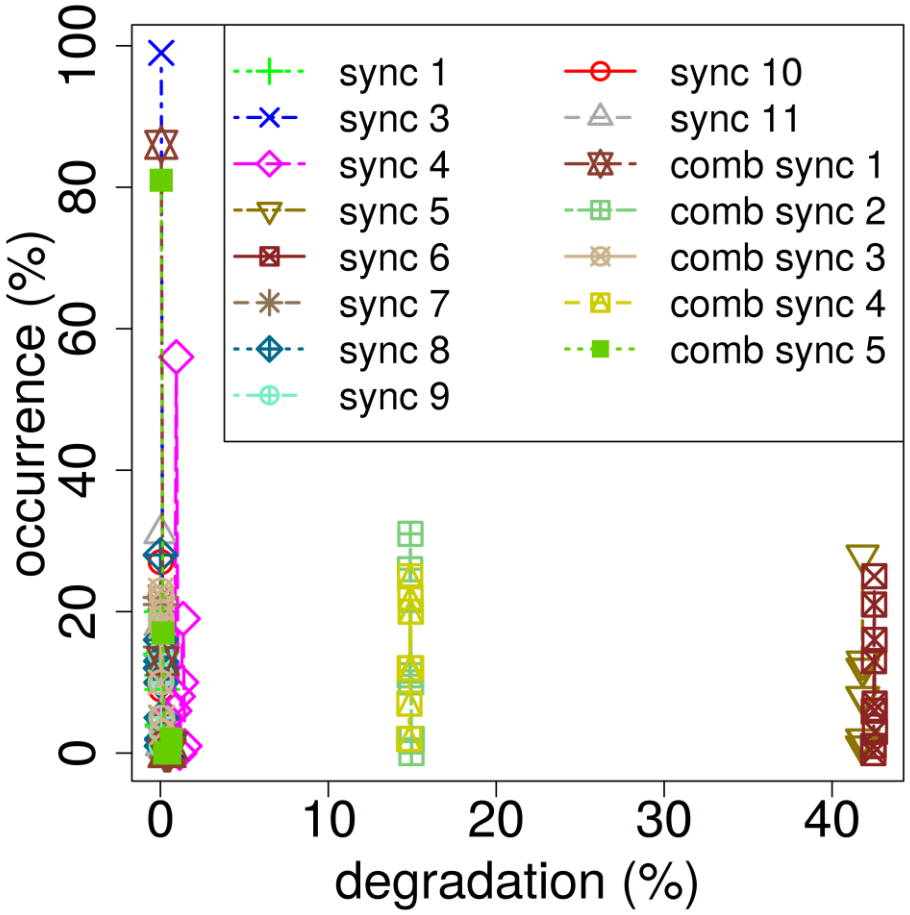
```
1    while( change/cost > 1.0*e){
2      …
3      if(pid == 0){
4        intshuffle(feasible, numfeasible);
5      }
6      pthread_barrier_wait(barrier);
7      …
8      change += pgain(feasible[x], …);
9      …
10     cost -= change;
11     …
12   }
```

T1

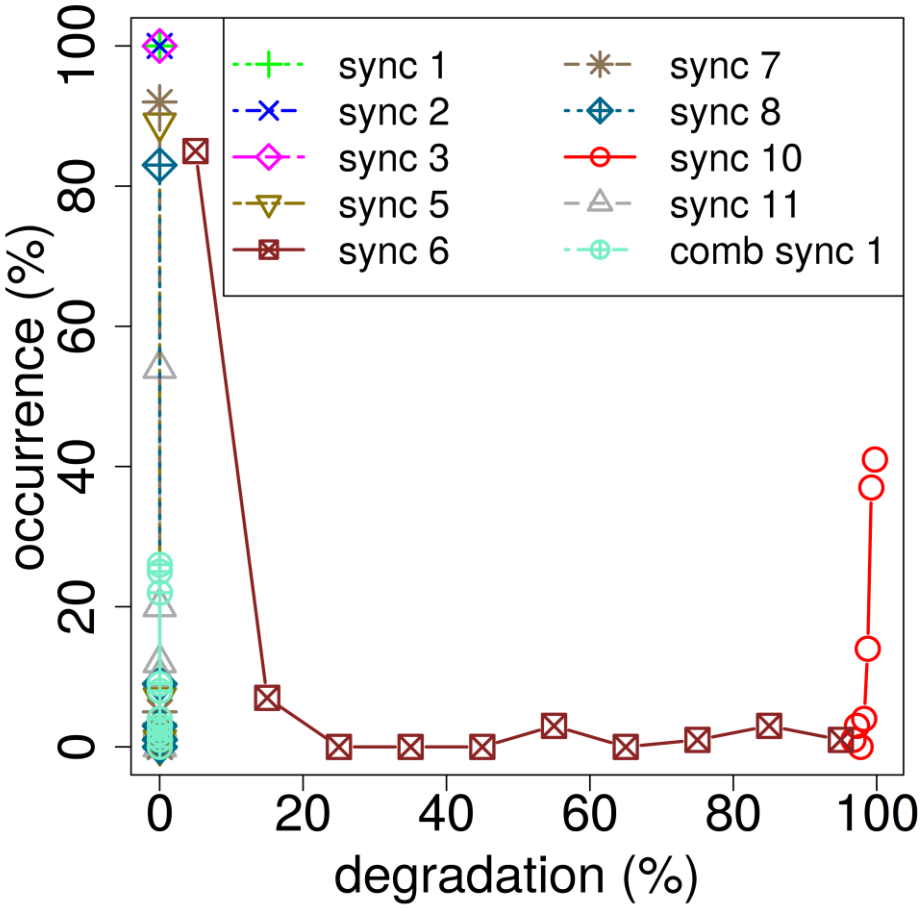
Evaluation: Quality metrics - backup

Benchmark	Domain	Quality Metric
barnes	N-body simulation	Difference in body positions
canneal	Optimization	Relative routing cost
dedup	Compression	Relative compression rate
fluidanimate	N-body simulation	Difference in particle positions
streamcluster	Clustering	Number of common clusters
ferret	Similarity search	Number of common images
bodytrack	Computer vision	SSD of output vectors
vips	Image processing	Peak-signal-to-noise ratio (PSNR)
raytrace	Real time animation	PSNR
x264	Video encoding	Structural similarity

Evaluation: Impact on output quality (canneal) - backup



Evaluation: Impact on output quality (barnes) - backup



Case Study: streamcluster – critical bug

```
1   // streamcluster.cpp: barrier @ 991 in pgain()
2   ...
3   if(pid == 0){
4     work_mem = (double *) malloc (...);
5     ...
6   }
7   pthread_barrier_wait(barrier);
8   for(i = k1; i < k2 ; i++){
9     if(is_center[i]){
10      center_table[i] = count++;
11    }
12  }
13  work_mem[pid*stride] = count;
14  ...
```

Case Study: streamcluster – accuracy bug

```
1   // streamcluster.cpp: barrier @ 1231 in pFL()
2   ...
3   while( change/cost > 1.0*e){
4     change = 0;
5     numberOfPoints = points->num;
6     if(pid == 0){
7       intshuffle(feasible, numfeasible);
8     }
9     pthread_barrier_wait(barrier);
10    for(i = 0; i < iter; i++){
11      x = i % numfeasible;
12      change += pgain(feasible[x], ...);
13    }
14    cost -= change;
15    pthread_barrier_wait(barrier);
16  }
17  return cost;
```