

# Accelerating Multiprocessor Simulation with a Memory Timestamp Record

Kenneth Barr

Heidi Pan

Michael Zhang

Krste Asanovic



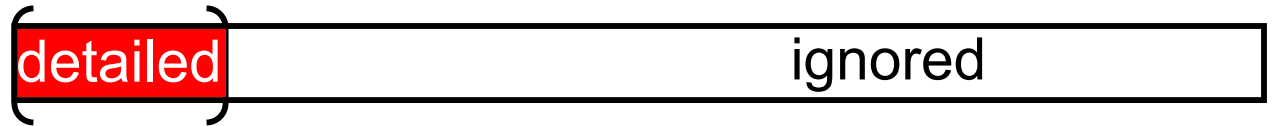
Massachusetts  
Institute of  
Technology



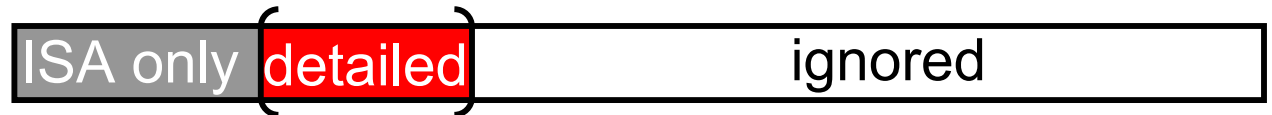
March 21, 2005

# Intelligent sampling gives best speed-accuracy tradeoff for uniprocessors (Yi, HPCA '05)

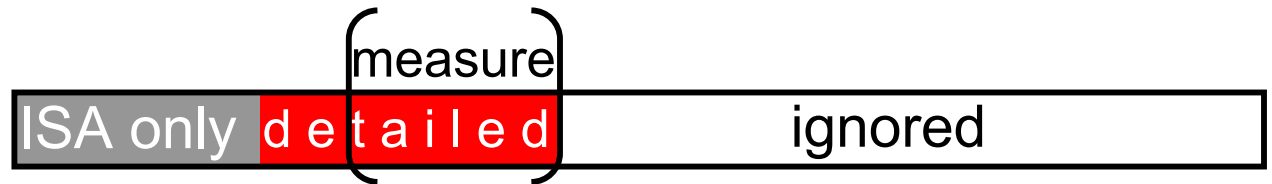
- Single sample



- Fastforward + single sample



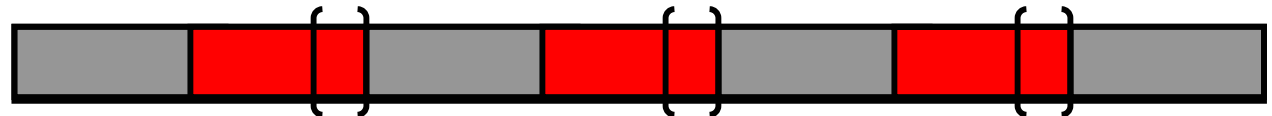
- Fastforward + Warmup + sample



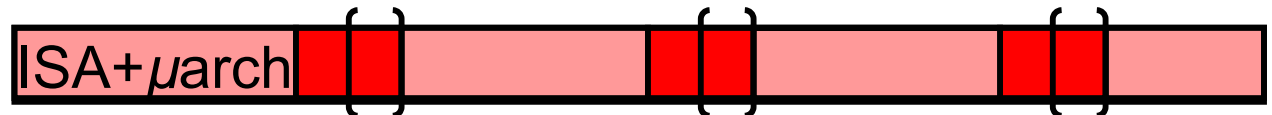
- Selective Sampling (SimPoints)



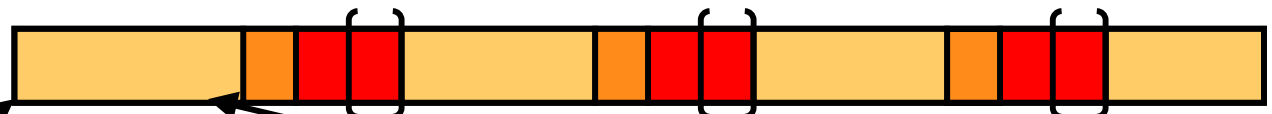
- Statistical Sampling



- Statistical sampling w/ Fast Functional Warming (SMARTS, FFW)



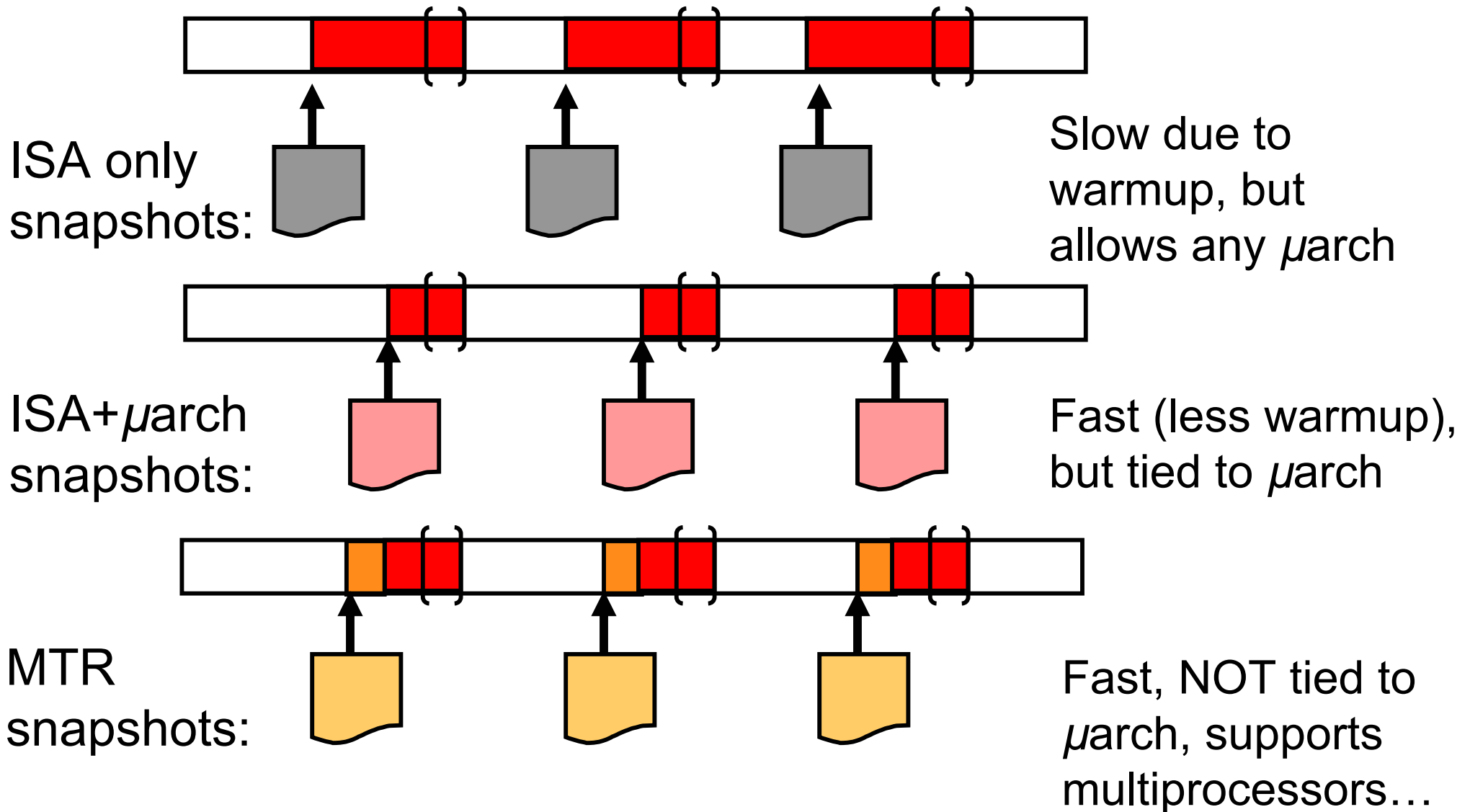
- Memory Timestamp Record



ISA+MTR Update

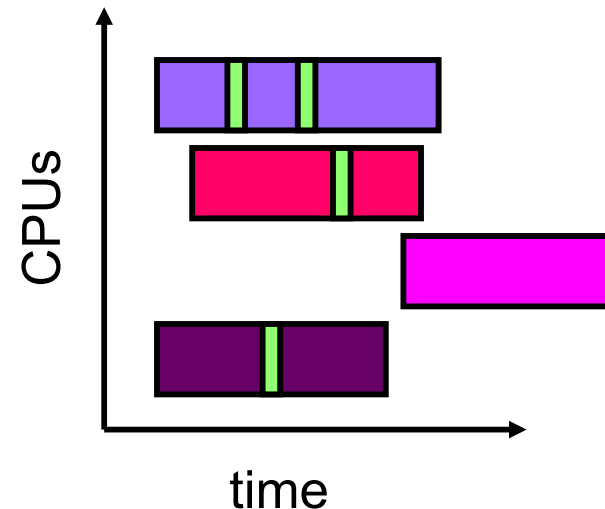
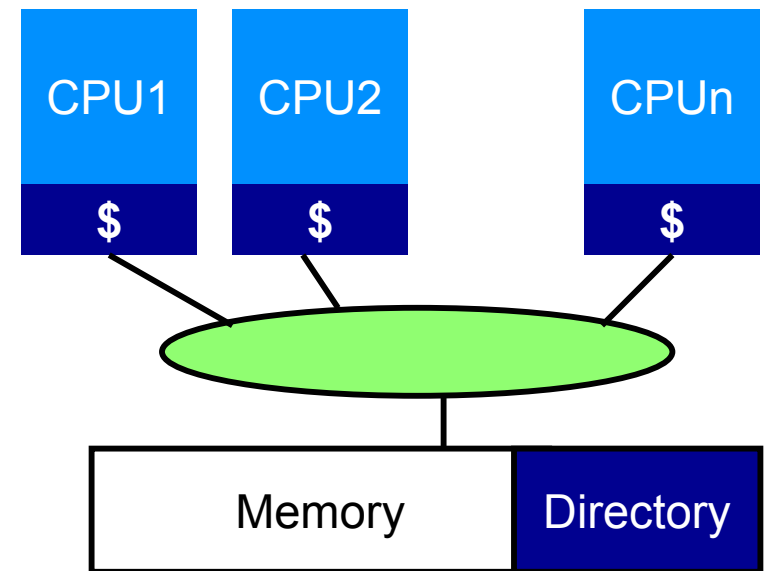
Reconstruct caches

# Snapshots amortize fast-forwarding, but require slow warming or bind to a particular $\mu$ arch



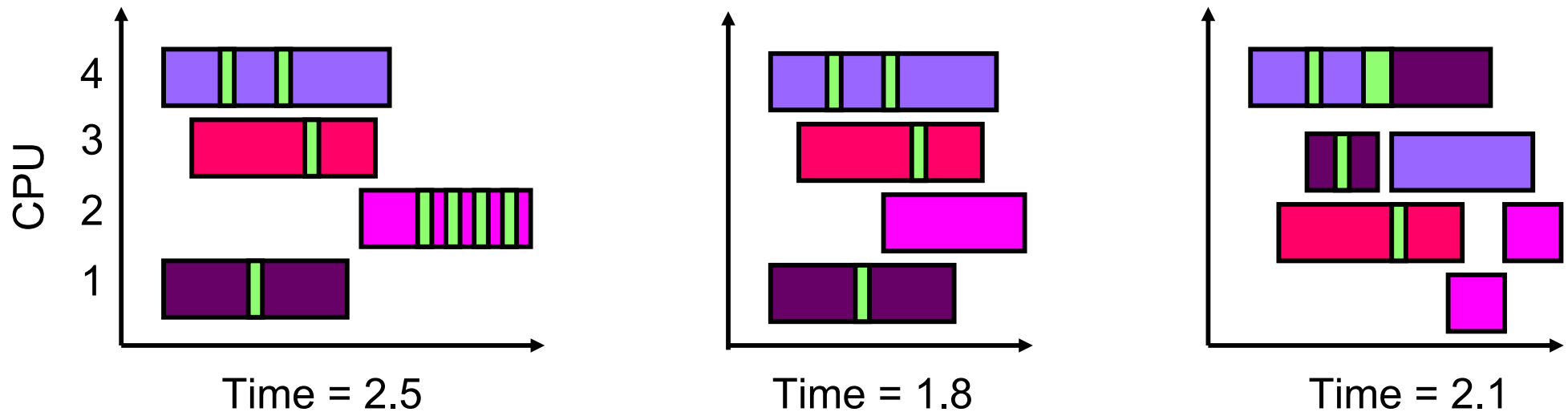
# Multiprocessors simulation is *especially* slow

- More cores →  
More state/complexity →  
Long, complex simulations
- Full system,  
threaded apps →  
More variability →  
More simulation



# For full-system simulations of commercial workloads, subtle variation matters!

(Alameldeen and Wood, 2003)



- All produce same result, each has different runtime
  - DRAM refresh
  - Hard disk arrangement delays DMA
  - Incoming packet interrupts application
  - Locking order reversed
  - Processes migrate
- Is our new gizmo a success? Maybe OS just ordered threads differently!

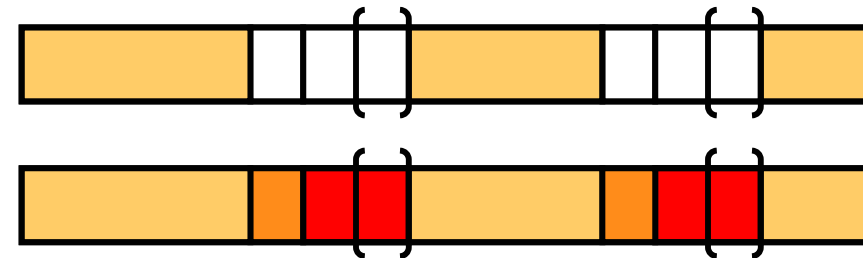
# What is the Memory Timestamp Record (MTR)?

- MTR is abstract picture of an multiprocessor's coherence state

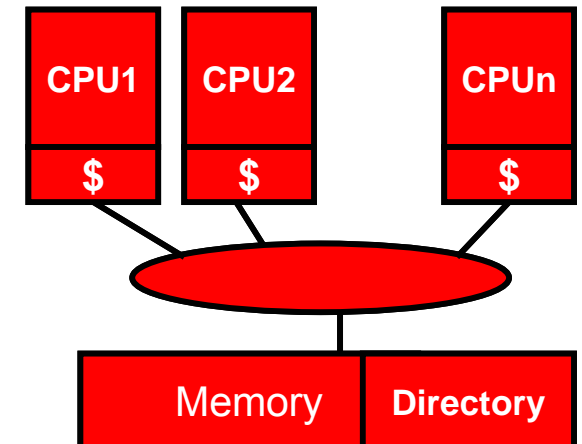
	CPU0	...	CPUn-1		
Block Address	Last Readtime			Last Writetime	Last Writer
0		...			
		...			
		...			
		...			
N-1		...			

# What is the Memory Timestamp Record (MTR)?

- MTR is abstract picture of an multiprocessor's coherence state
  - Allow quick update during fast forwarding
  - Fill in concrete caches and directory prior to sampling

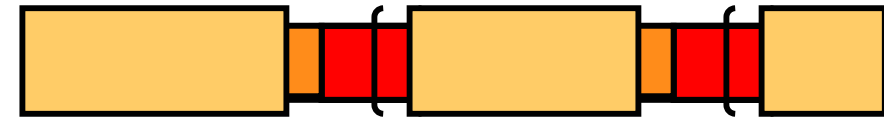


	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
0		...			
		...			
		...			
		...			
N-1		...			



# MTR example: update

- MTR contains one entry per memory block; locality keeps it sparse.
- New access times overwrite old (self-compressing)



Memory Trace:

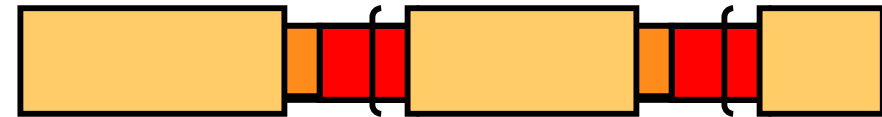
Time	CPU0	CPU1
0		
1		
2		
3		
4		

MTR:

	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
a		...			
b		...			
c					
d					
e		...			

# MTR example: update

- MTR contains one entry per memory block; locality keeps it sparse.
- New access times overwrite old (self-compressing)



Memory Trace:

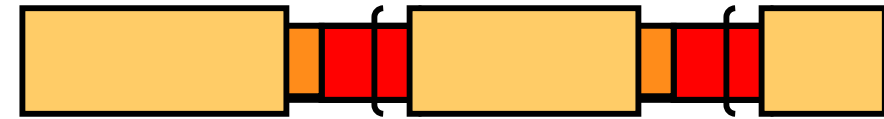
Time	CPU0	CPU1
0	Read a	
1		
2		
3		
4		

MTR:

	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
a	0	...			
b		...			
c					
d					
e		...			

# MTR example: update

- MTR contains one entry per memory block; locality keeps it sparse.
- New access times overwrite old (self-compressing)



Memory Trace:

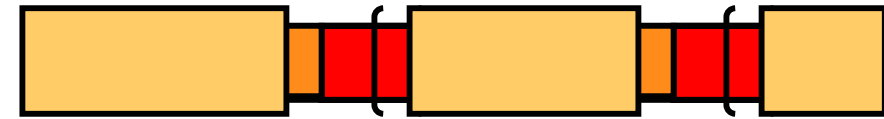
Time	CPU0	CPU1
0	Read a	
1	Read e	
2		
3		
4		

MTR:

	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
a	0	...			
b		...			
c					
d					
e	1	...			

# MTR example: update

- MTR contains one entry per memory block; locality keeps it sparse.
- New access times overwrite old (self-compressing)



Memory Trace:

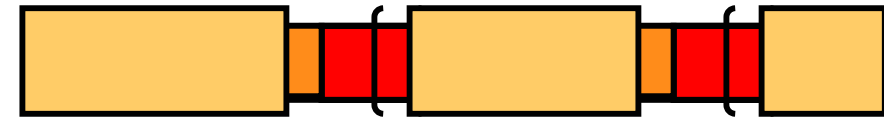
Time	CPU0	CPU1
0	Read a	
1	Read e	
2	Read b	
3		
4		

MTR:

	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
a	0	...			
b	2	...			
c					
d					
e	1	...			

# MTR example: update

- MTR contains one entry per memory block; locality keeps it sparse.
- New access times overwrite old (self-compressing)



Memory Trace:

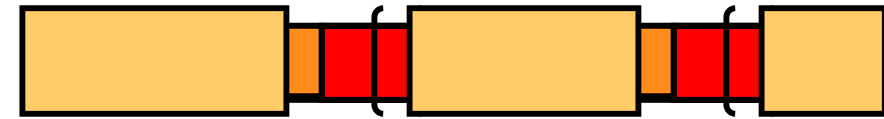
Time	CPU0	CPU1
0	Read a	
1	Read e	
2	Read b	
3	Read c	
4		

MTR:

	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
a	0	...			
b	2	...			
c	3				
d					
e	1	...			

# MTR example: update

- MTR contains one entry per memory block; locality keeps it sparse.
- New access times overwrite old (self-compressing)



Memory Trace:

Time	CPU0	CPU1
0	Read a	
1	Read e	
2	Read b	
3	Read c	
4		Write b

MTR:

	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

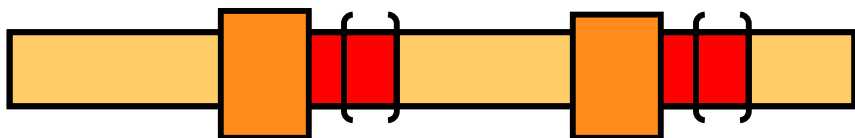
# 1. Coalesce: determining correct cache tags

MTR:

Block Address	MTR			Last Writetime	Last Writer
	CPU0	...	CPU <sub>n-1</sub>		
		...			
		...			
		...			
		...			
		...			
		...			
		...			
		...			

Cache:

	Way 0	Way 1
Set 0	Blue	Blue
Set 1	Green	Green
Set 2	Orange	Orange
Set 3	Red	Red



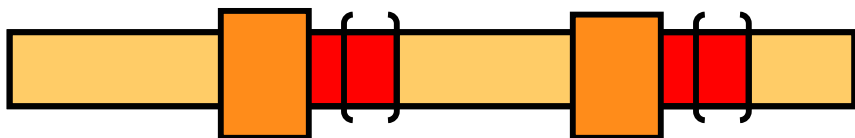
# 1. Coalesce: determining correct cache tags

MTR:

Block Address	CPU			Last Writetime	Last Writer
	CPU0	...	CPU <sub>n-1</sub>		
		...			
		...			
		...			
		...			
		...			
		...			
		...			

Cache:

	Way 0	Way 1
Set 0	Blue	Blue
Set 1	Green	Green
Set 2	Orange	Orange
Set 3	Red	Red



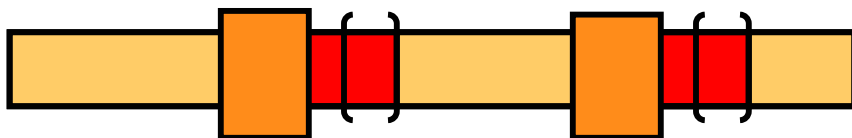
# 1. Coalesce: determining correct cache tags

MTR:

Block Address	CPU0	...	CPU <sub>n-1</sub>	Last Writetime	Last Writer
	Last Readtime				
		...			
		...			
		...			
		...			
		...			
		...			
		...			
		...			
		...			
		...			

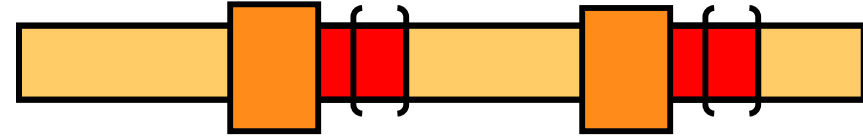
Cache:

	Way 0	Way 1
Set 0		
Set 1		
Set 2		
Set 3		



# MTR example: coalesce

- Choose organization
  - One set, two ways
- Coalesce
  - Determine which blocks map to same set
  - Only *ways* most recent timestamps are present. Check validity later.



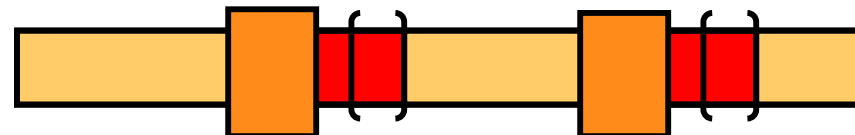
	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
<b>a</b>	<b>0</b>	...			
<b>b</b>	<b>2</b>	...		<b>4</b>	<b>CPU1</b>
<b>c</b>	<b>3</b>				
<b>d</b>					
<b>e</b>	<b>1</b>	...			

	Way 0		Way 1	
Set 0				
Set 1				

	Way 0		Way 1	
Set 0				
Set 1				

# MTR example: coalesce

- Choose organization
  - One set, two ways
- Coalesce
  - Determine which blocks map to same set
  - Only *ways* most recent timestamps are present. Check validity later.



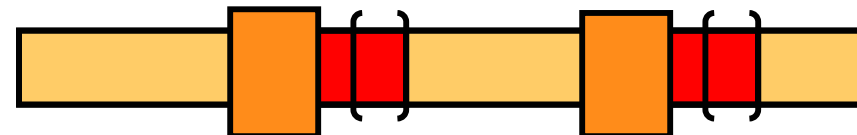
	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

- What are the contents of CPU0 cache?

	Way 0		Way 1	
Set 0				
Set 1				

# MTR example: coalesce

- Choose organization
  - One set, two ways
- Coalesce
  - Determine which blocks map to same set
  - Only *ways* most recent timestamps are present. Check validity later.



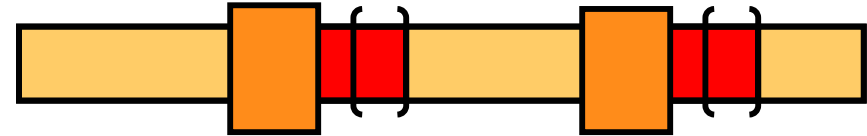
Block Address	CPU0	...	CPU <sub>n-1</sub>	Last Writetime	Last Writer
	Last Readtime				
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

- What are the contents of CPU0 cache?

	Way 0		Way 1	
Set 0	a	0		
Set 1				

# MTR example: coalesce

- Choose organization
  - One set, two ways
- Coalesce
  - Determine which blocks map to same set
  - Only *ways* most recent timestamps are present. Check validity later.



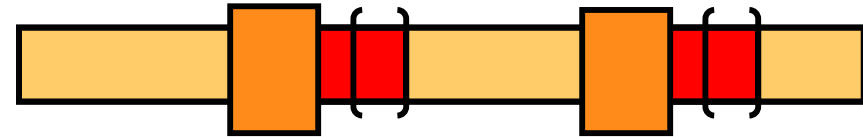
Block Address	CPU0	...	CPU <sub>n-1</sub>	Last Writetime	Last Writer
	Last Readtime				
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

- What are the contents of CPU0 cache?

	Way 0		Way 1	
Set 0	a	0		
Set 1	b	2		

# MTR example: coalesce

- Choose organization
  - One set, two ways
- Coalesce
  - Determine which blocks map to same set
  - Only *ways* most recent timestamps are present. Check validity later.



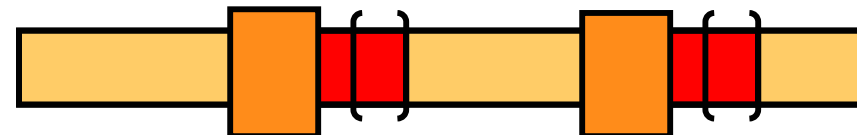
Block Address	CPU0	...	CPU <sub>n-1</sub>	Last Writetime	Last Writer
	Last Readtime				
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

- What are the contents of CPU0 cache?

	Way 0		Way 1	
Set 0	a	0	c	3
Set 1	b	2		

# MTR example: coalesce

- Choose organization
  - One set, two ways
- Coalesce
  - Determine which blocks map to same set
  - Only *ways* most recent timestamps are present. Check validity later.



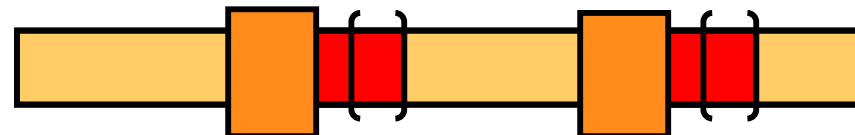
Block Address	CPU0	...	CPU <sub>n-1</sub>	Last Writetime	Last Writer
	Last Readtime				
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

- What are the contents of CPU0 cache?

	Way 0		Way 1	
Set 0	e	1	c	3
Set 1	b	2		

# MTR example: coalesce

- Choose organization
  - One set, two ways
- Coalesce
  - Determine which blocks map to same set
  - Only *ways* most recent timestamps are present. Check validity later.



Block Address	Last Readtime			Last Writetime	Last Writer
	CPU0	...	CPU <sub>n-1</sub>		
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

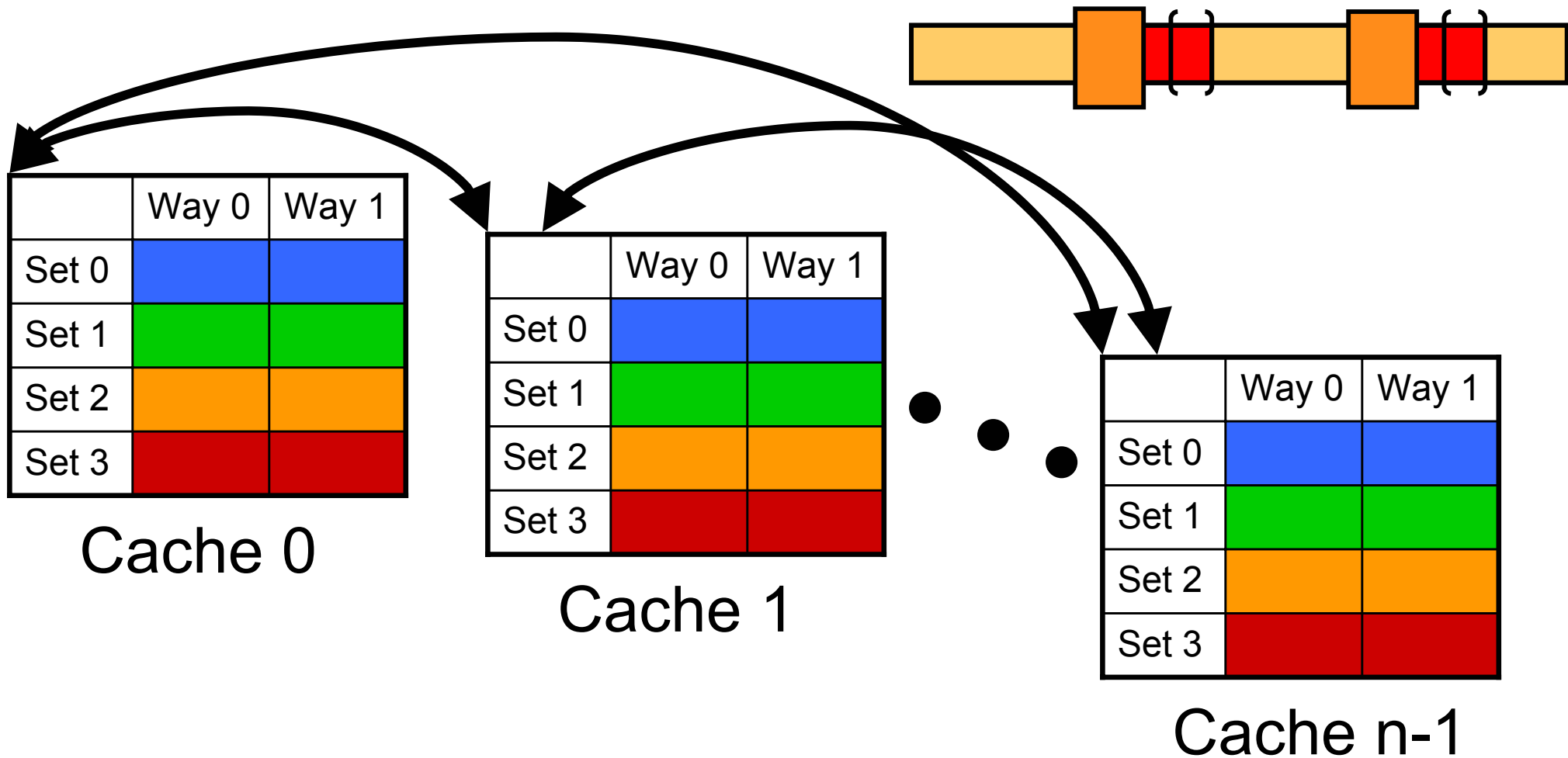
- What are the contents of CPU0 cache?

	Way 0		Way 1	
Set 0	e	1	c	3
Set 1	b	2		

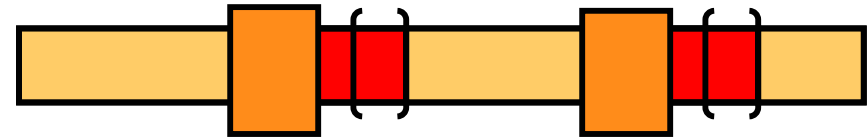
- CPU1?

	Way 0		Way 1	
Set 0				
Set 1	b	4		

## 2 Fixup: determine correct status bits



# MTR example: fixup



- Reads prior to a write are invalid, valid writes are dirty, etc...

Block Address	CPU0	...	CPU <sub>n-1</sub>	Last Writetime	Last Writer
	Last Readtime				
a	0	...			
b	2	...		4	CPU1
c	3				
d					
e	1	...			

Which cache has the most recent copy of 'b'?

	Way 0		Way 1	
Set 0	e	1	c	3
Set 1	b	2		

invalid

	Way 0		Way 1	
Set 0				
Set 1	b	4		

Valid, dirty

# MTR example: directory reconstruction

	CPU0	...	CPU <sub>n-1</sub>		
Block Address	Last Readtime			Last Writetime	Last Writer
<b>a</b>	<b>0</b>	...			
<b>b</b>	<b>2</b>	...		<b>4</b>	<b>CPU1</b>
<b>c</b>	<b>3</b>				
<b>d</b>					
<b>e</b>	<b>1</b>	...			

Block Address	State	Sharers
<b>a</b>	<b>S</b>	<b>CPU0</b>
<b>b</b>	<b>M</b>	<b>CPU1</b>
<b>c</b>	<b>S</b>	<b>CPU0</b>
<b>d</b>	<b>I</b>	
<b>e</b>	<b>S</b>	<b>CPU0</b>

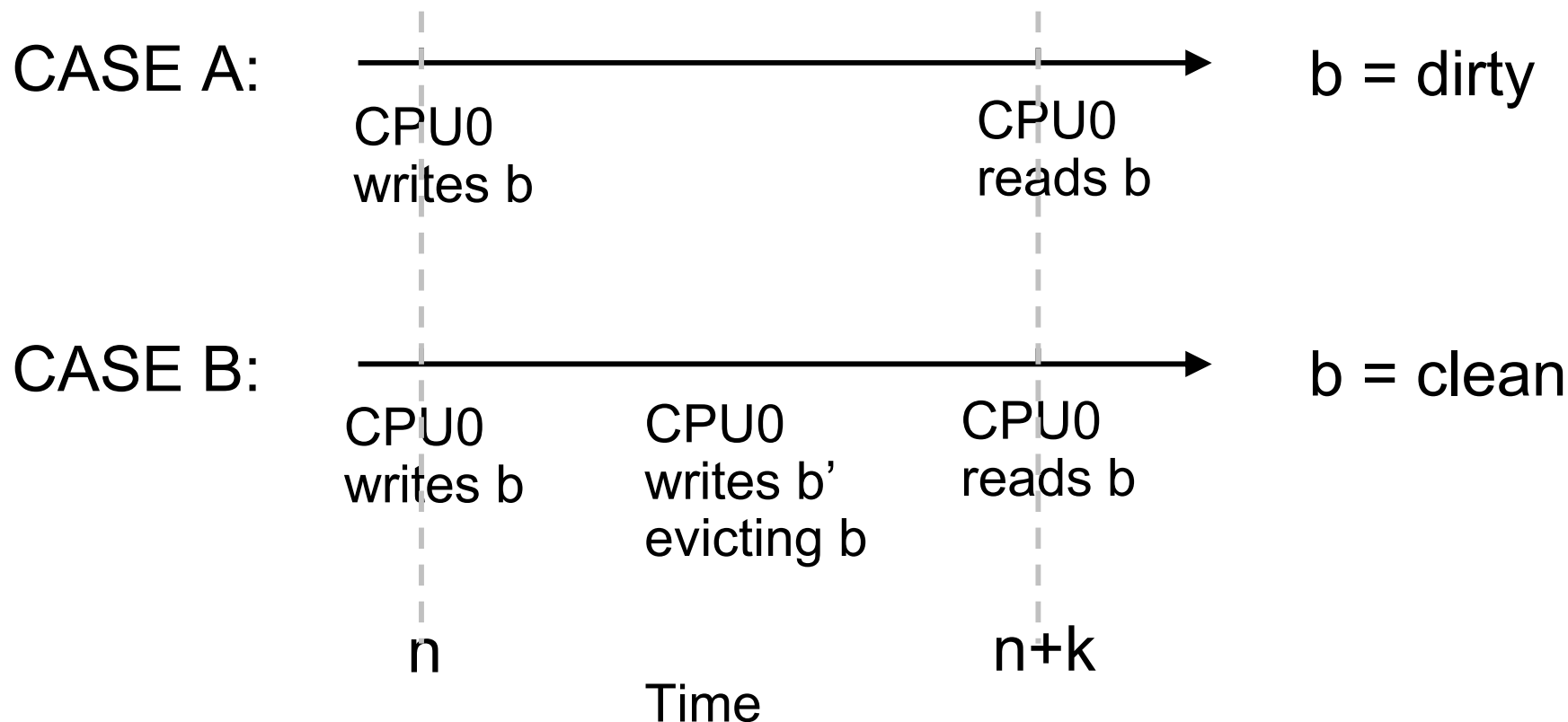
# The MTR supports many popular organizations and protocols

- Snoopy or directory-based
- Multilevel caches
  - Inclusive
  - Exclusive
- Time-based replacement policy
  - Strict LRU
  - Cache decay
- Invalidate, Update, Update-Invalidate
- MSI, MESI, MOESI

# Evicts cannot be recorded in the MTR, but many can be inferred

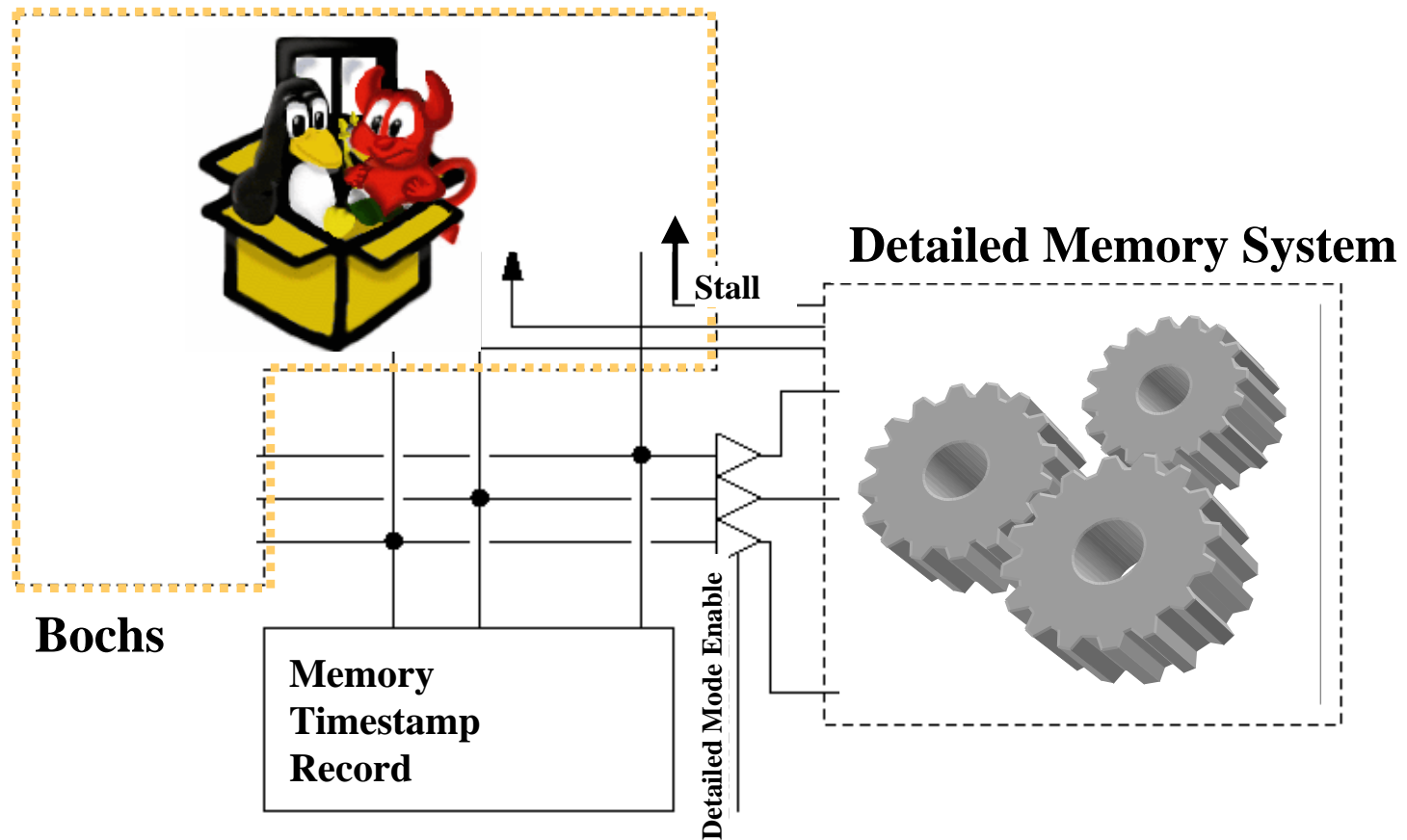
MTR:

address	CPU0	CPU1	Writetime	Writer
<b>b</b>	<b>n+k</b>		<b>n</b>	<b>CPU0</b>

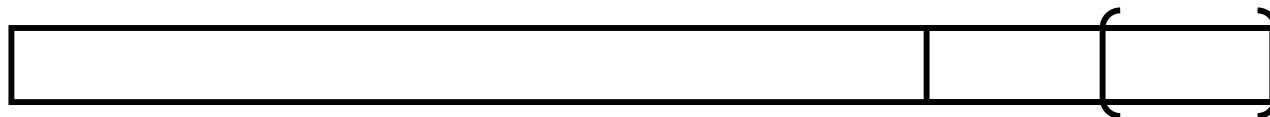


# Evaluation / Results

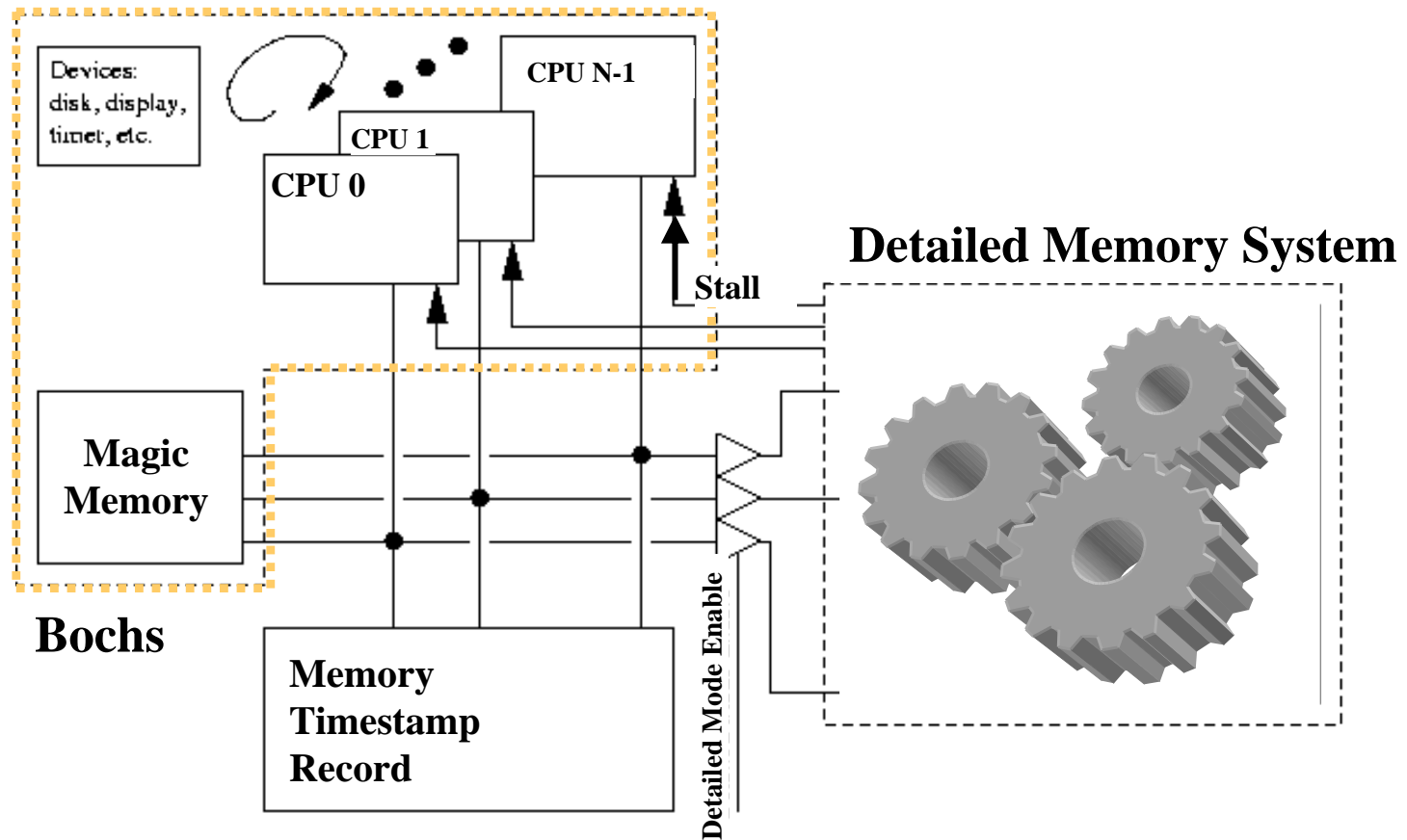
# Detailed, full-system, execution-driven, x86, SMP simulation



- SMP Bochs provides: devices (allowing an OS), x86 Decoding and Execution, Magic Memory



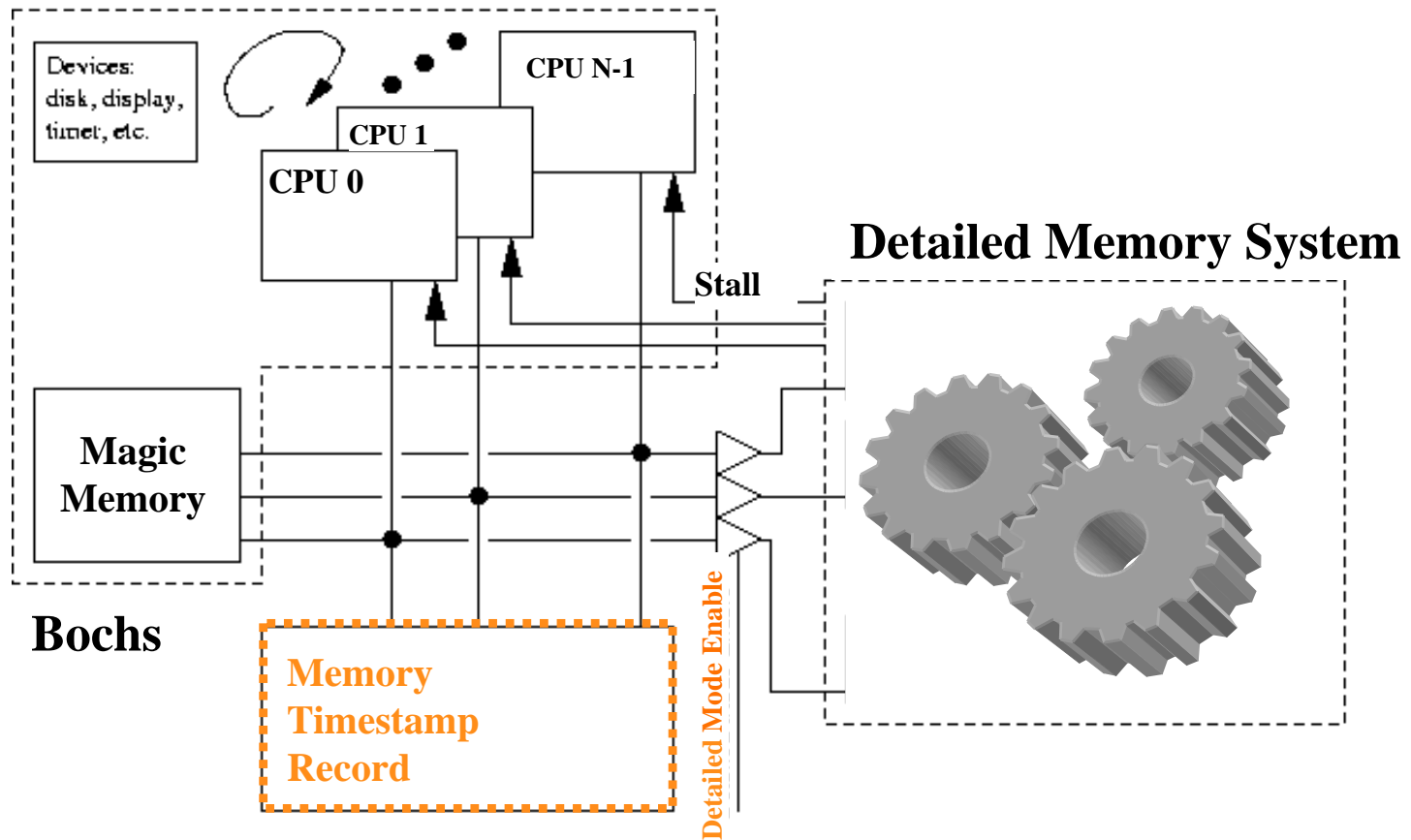
# Detailed, full-system, execution-driven, x86, SMP simulation



- SMP Bochs provides: devices (allowing an OS), x86 Decoding and Execution, Magic Memory



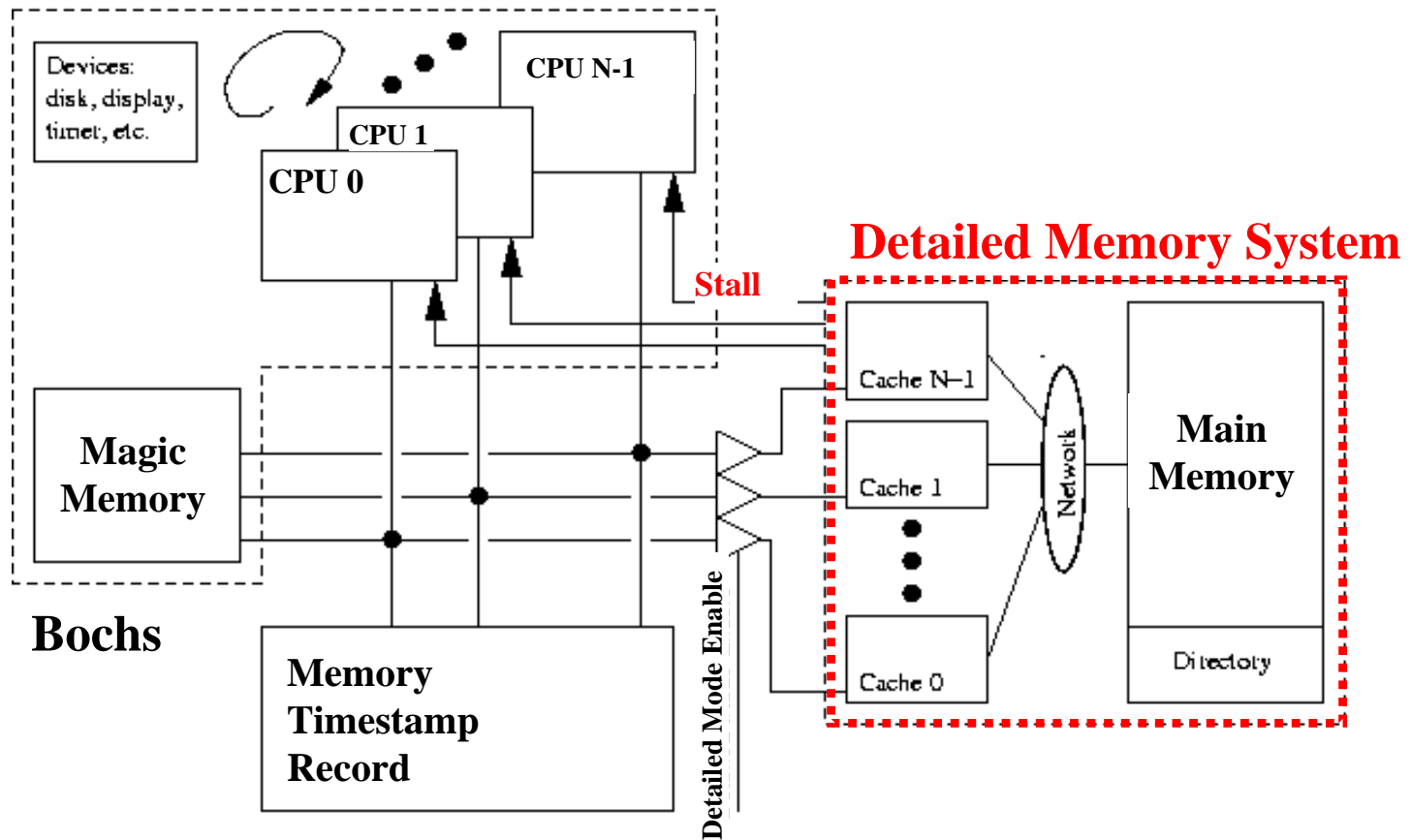
# Reconstructing with the MTR



- **Memory Timestamp Record:** allows switching between functional fast-fwd and detailed simulation



# Our detailed memory model can stall a processor's execution based on timing models

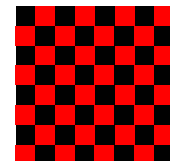
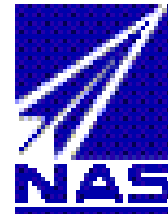


- **Detailed Memory System** provides: cache coherence, network, DRAM timing, stall signal



# Benchmarks!

- NASA Advanced Supercomputing Parallel Benchmarks:
  - scientific (comp. fluid dynamics)
  - OpenMP (loop iterations in parallel)
  - Fortran
- 2 OS benchmarks
  - dbench: (Samba) several clients making file-centric system calls
  - Apache: several clients hammer web server (via loopback interface)
- Cilk: checkers: AI search plies in parallel
  - uses spawn/sync primitives (dynamic thread creation/scheduling)

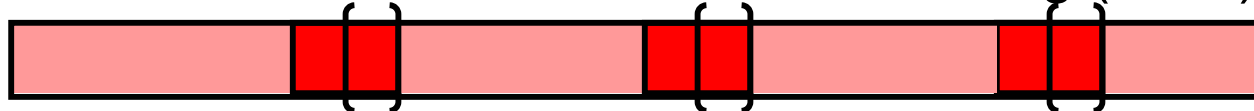


# We compare MTR to full detailed and a “naïve” implementation of SMP fast forwarding.

- Baseline 1: full detailed simulation (overnight)



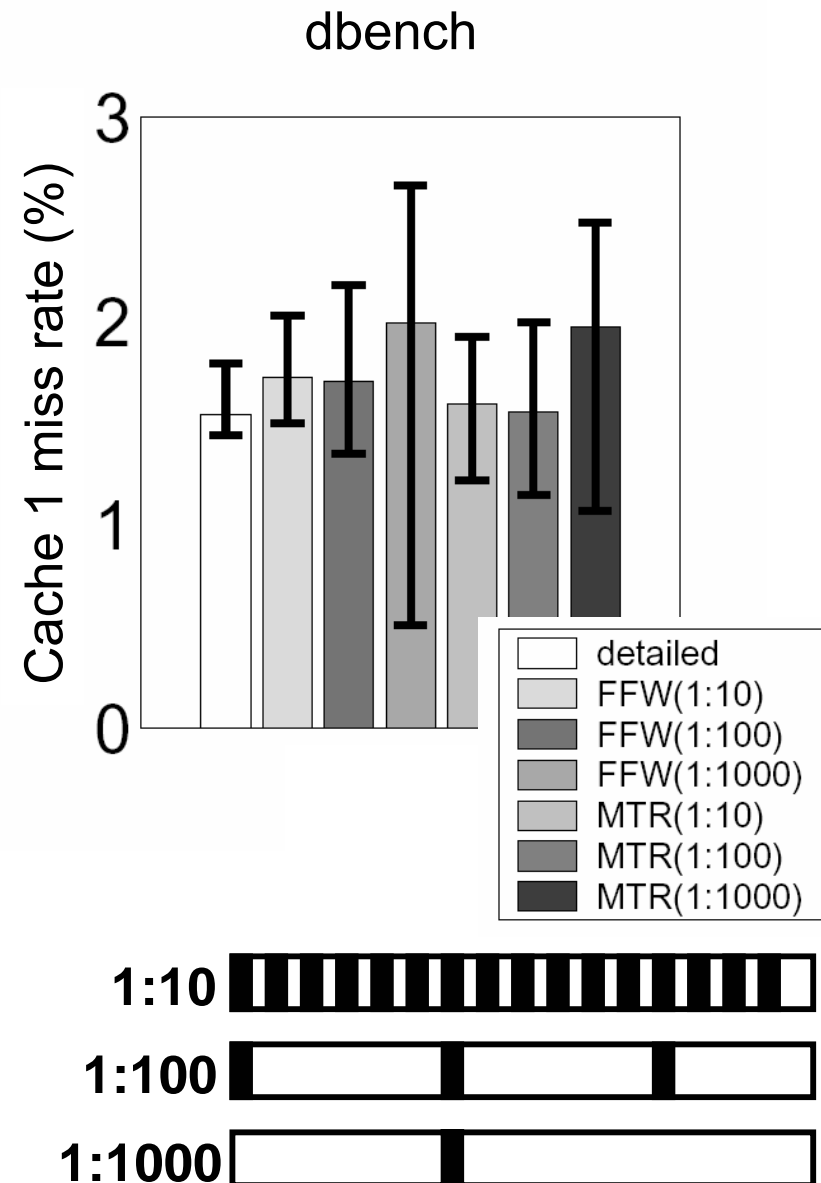
- Baseline 2: “naïve” functional fast forwarding (FFW)



- Functional simulation of ISA
- Cache / directory state kept accurate
  - Tag checks, replacement policy enforced
  - Directory consulted and updated
  - On miss/coherence miss, invalidate outstanding copies
  - Omits network messages, queues, latencies (present in detailed mode)
- Hypothesis
  - Both FFW and MTR should be accurate and fast
  - MTR should be faster than FFW
  - To be useful, FFW and MTR must answer questions in the same way as a detailed model, but faster

# Full-system experiments must respect system variation, or risk incorrect prediction!

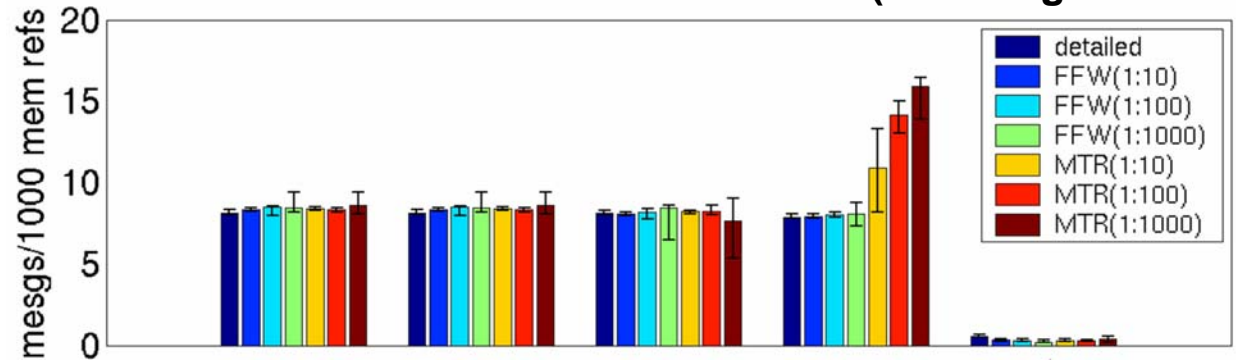
- Methodology
  - Every 10k cycles choose victim processor
  - Victim will run 25% slower to emulate variation
  - Note: variation has MUCH larger effect during fast mode
- Bar shows the median of eight runs, with ticks for min and max. **Each run is a valid result!**
- Ideal: range for fast runs should be **within** range of (all possible) detailed.
- Can't draw conclusions about discrepancy until we understand distribution



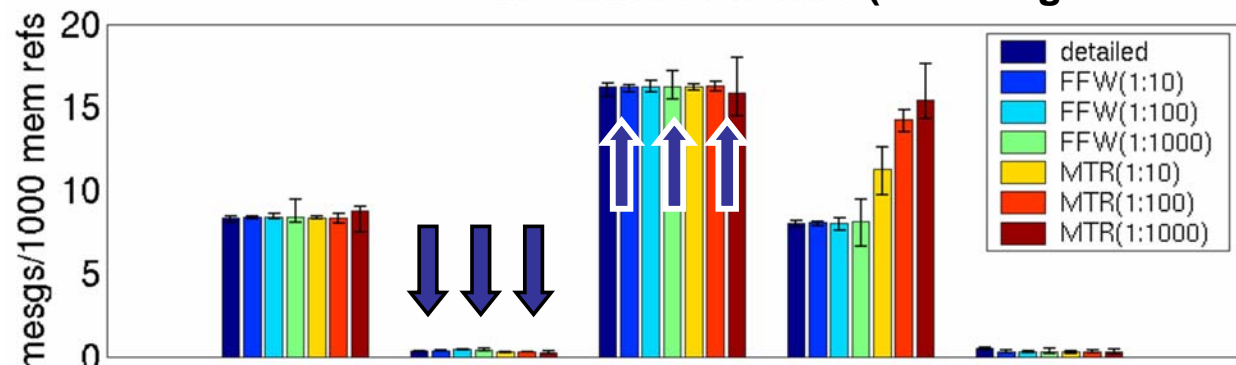
# Replicating “detailed”-mode stats less crucial than accurate answers to design questions

- Assume observed = actual
- With respect to reply message types, the MSI vs. MESI change is dramatic.
  - All fast-fwd bars move with the detailed bar.
  - Movement beyond range of detailed runs
- Discover evicts to more closely match detailed run
  - Or, tune victim/slowdown

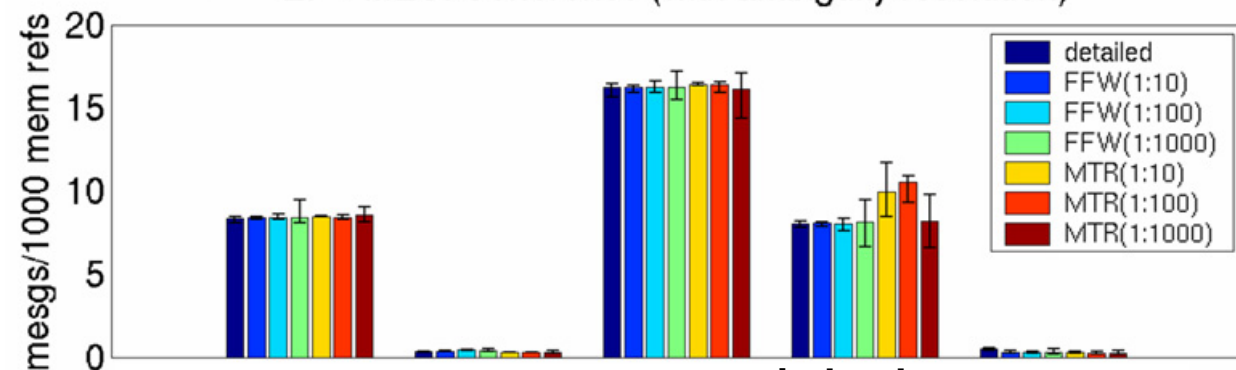
EP - MSI Coherence (no ambig. resolution)



EP - MESI Coherence (no ambig. resolution)



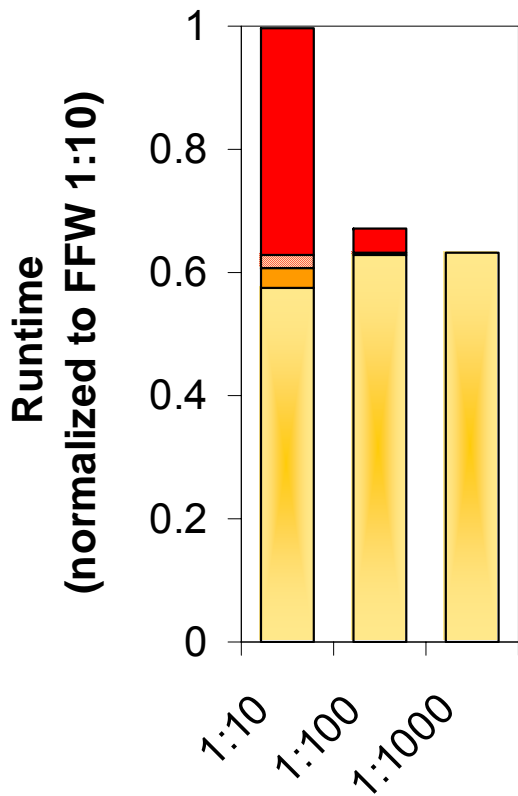
EP - MESI Coherence (with ambiguity resolution)



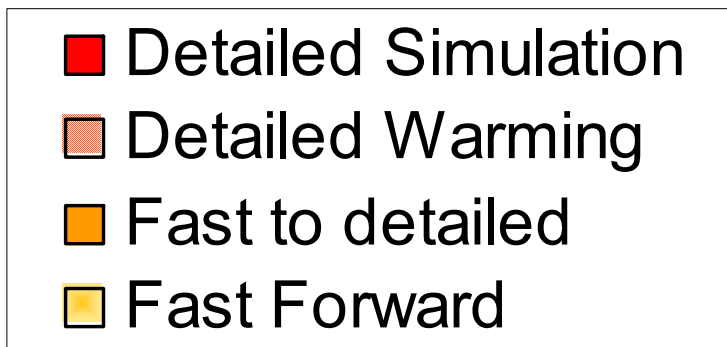
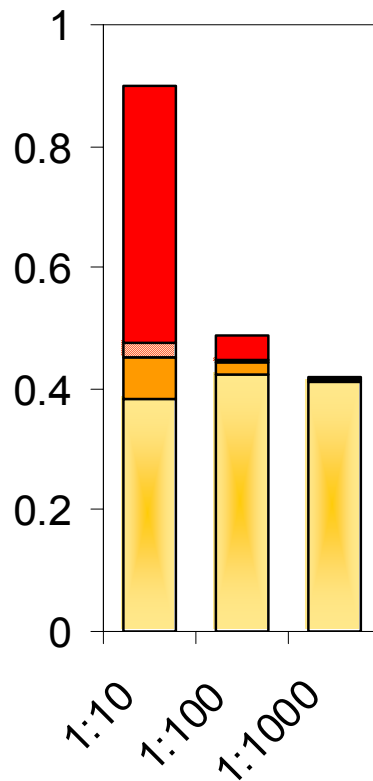
writeback rep

# MTR averages up to 1.45X faster than FFW

FFW (mg)



MTR (mg)



- MTR spends less time in fast forward
  - MTR does less work in common case
- Time saved in fast forward time less than MTR transition cost
  - MTR has costlier transition, but
  - Reconstruction scales with *touched lines* not total accesses



# Conclusion: Memory Timestamp Record provides fast, accurate, and flexible SMP simulation

- MTR
  - 1.45X faster than functional warming
  - 7.7X faster than our detailed simulator
  - Eliminates need to regenerate snapshots
  - Answers architectural questions similar to detailed simulation
- Future work
  - Simultaneous multiple-configuration simulation
  - MTR compression for disk snapshots
  - Parallelized update and reconstruction

<http://cag.csail.mit.edu/scale/bochs.html>