

# Lock Prediction

Brandon Lucia Joseph Devietti Tom Bergan Luis Ceze Dan Grossman

Prediction is everywhere!

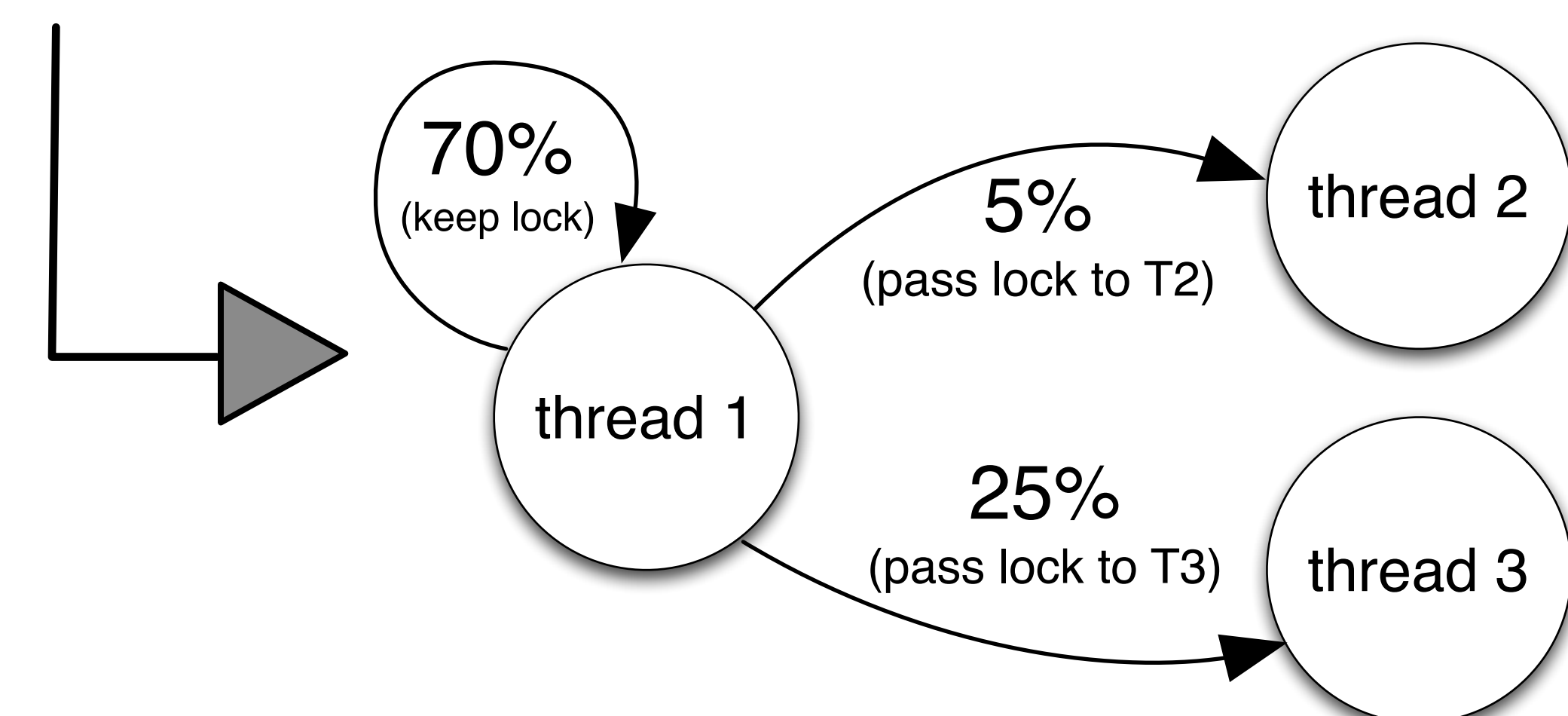
caching    branch predictors  
 speculation    ...

Goal: predict ...

*lock's-next-thread*  
*thread's-next-lock*

## Prediction Algorithms for *lock's-next-thread*

- first-to-acquire
- last-to-acquire
- most-frequent-acquirer
- most-frequent-transition



lock transition diagram for lock L, when held by thread 1

- ... many other possibilities  
example: weighting recent past heavily

## Prediction Algorithms for *thread's-next-lock*

for *thread's-next-lock*

... are the same!

Lock L's Acquirer History				
T1	T2	T1	T2	T1
Old				New

Thread T's Lock History				
L1	L2	L1	L2	L1
Old				New

Lock's-Next-Thread Prediction	
first-to-acq:	T1
last-to-acq:	T1
most-freq-acq:	T1
most-freq-trans:	T2

Thread's-Next-Lock Prediction	
first-to-acq:	L1
last-to-acq:	L1
most-freq-acq:	L1
most-freq-trans:	L2

(just swap the role of threads and locks)

### Aliasing

- "bucket" locks or threads e.g., by creation ctx
- reduce space overheads
- possibly lose accuracy

### Context

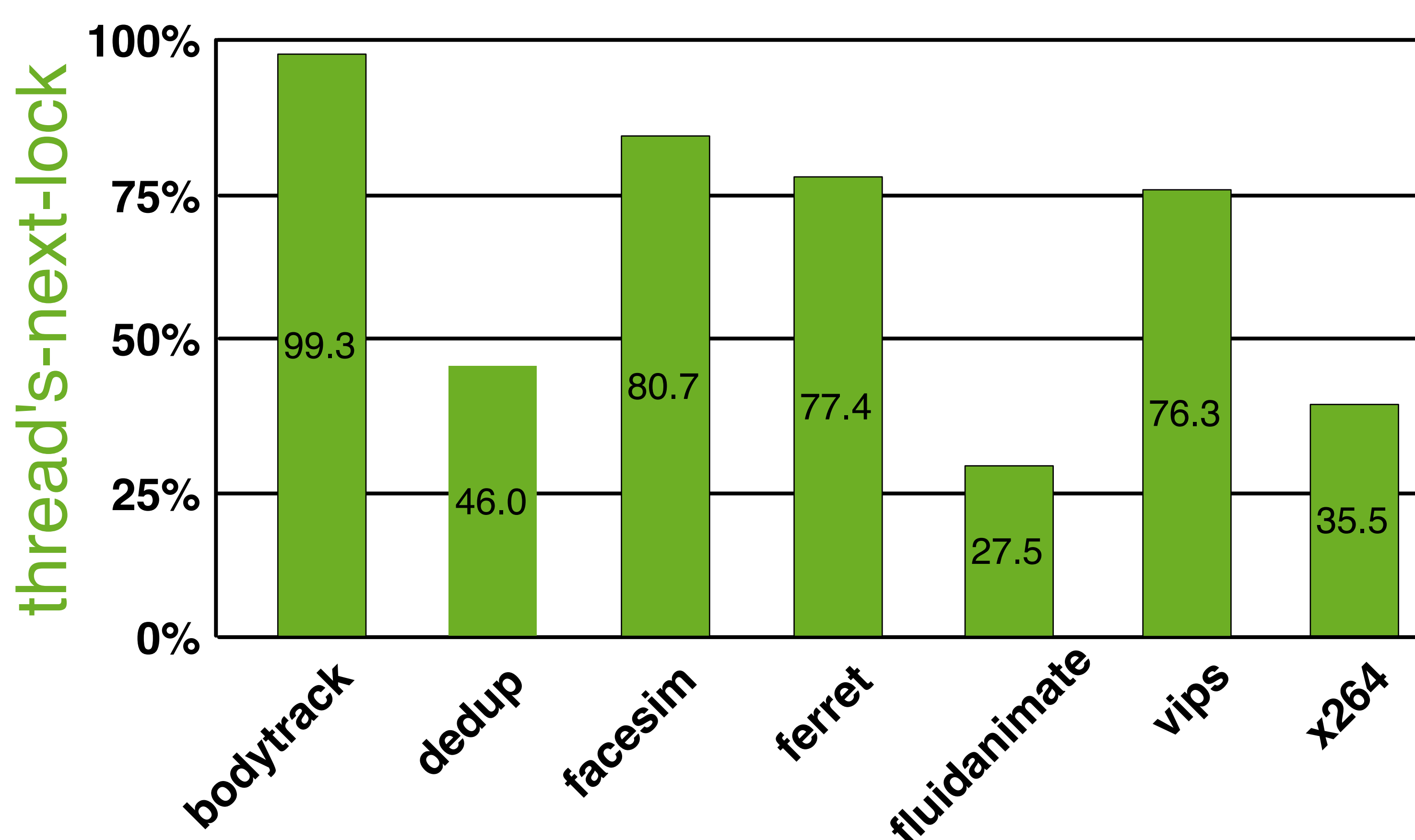
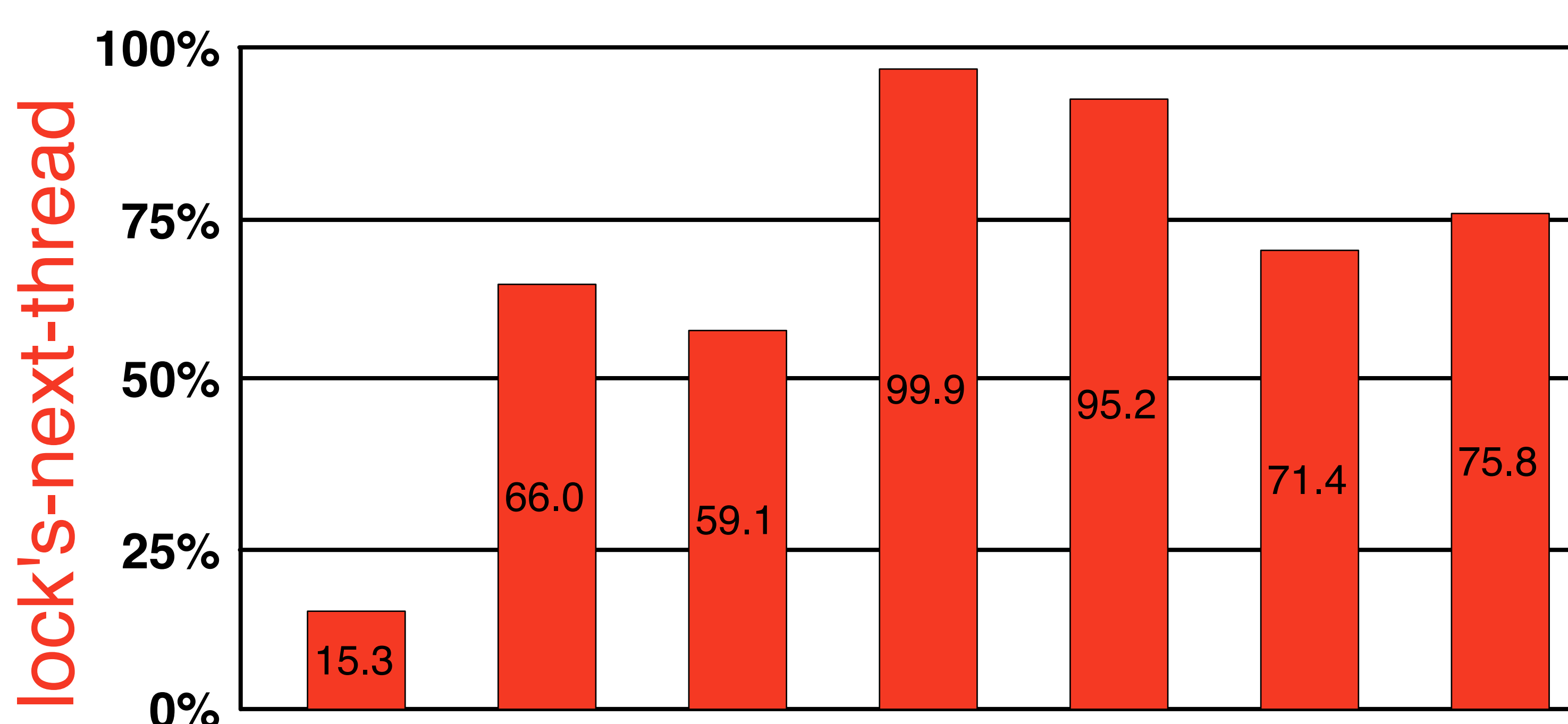
- make histories more precise e.g., add call stacks
- increase space overheads
- possibly gain accuracy

## Why predict locks?

potential to improve ...

- scheduling
- anomaly detection
- prefetching
- determinism

## Experimental Results with parsec / 8 threads



## Results Summary

- most-frequent-transition always the best
- aliasing
  - sometimes improves accuracy (as much as 50%)
  - sometimes hurts accuracy (as much as 65%)
- context
  - sometimes improves accuracy (as much as 20%)
  - never hurts accuracy