

# CACHE OPTIMIZED HASH TABLES



# WHY OPTIMIZE FOR CPU CACHE

RANDOMLY ACCESSED MEMORY LATENCY VS PROCESSING CYCLE TIME HAS BECOME DISPROPORTIONATE.

- IF CPU CACHE MEMORY EFFICIENCY INCREASES BY SMALL AMOUNT PERFORMANCE IMPROVES MUCH MORE

THE CPU IS NOT USED IF IT'S WAITING TO ACCESS MEMORY

OTHER CORES AREN'T WORKING IF THE MEMORY BUS IS OCCUPIED

# CACHE AND LOCALITY

## FILLING LINES/BLOCKS

- DATA AROUND ACCESS SHOULD ALSO BE USEFUL.

## REDUCING MEMORY USE

- REDUCE MEMORY USE AND IMPROVE CACHE MISS RATES

## INSTRUCTION CACHE OPTIMIZATIONS

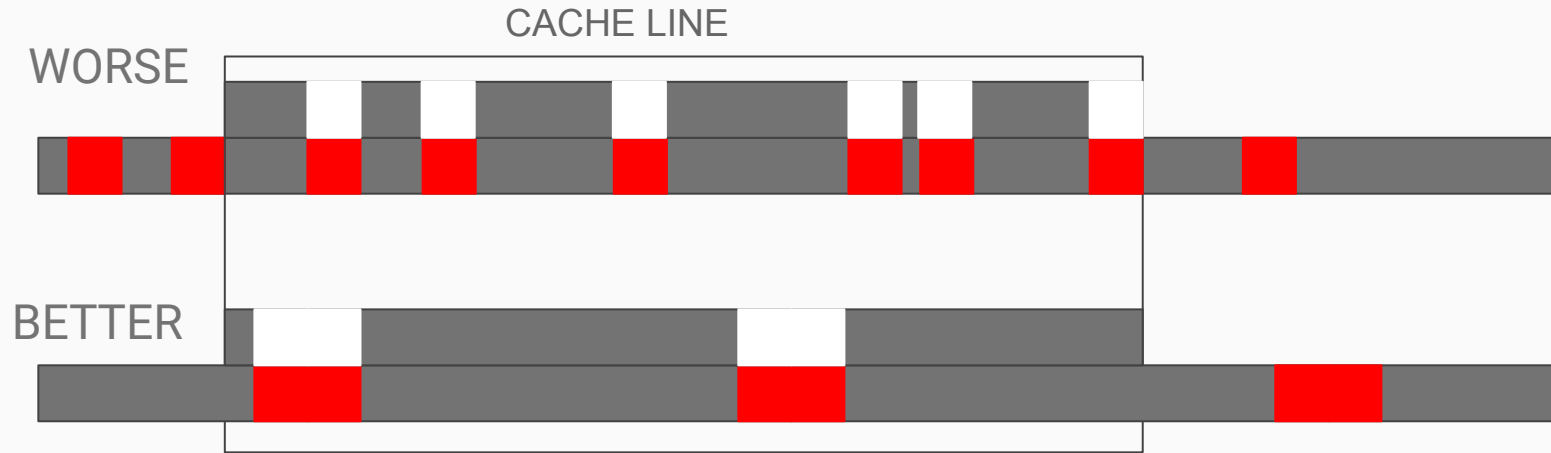
- INFREQUENTLY CALLED FUNCTIONS SHOULD NOT BE INLINED - OPTIMIZER DOES NOT ALWAYS KNOW WHICH -  
PROFILE,PROFILE,PROFILE

## LRU AND LOCALITY OF REFERENCE

- ISOLATING AND MIRRORING MOST USED PARTS OF A DATA STRUCTURE REDUCES EVICTION OF MOST USED PARTS

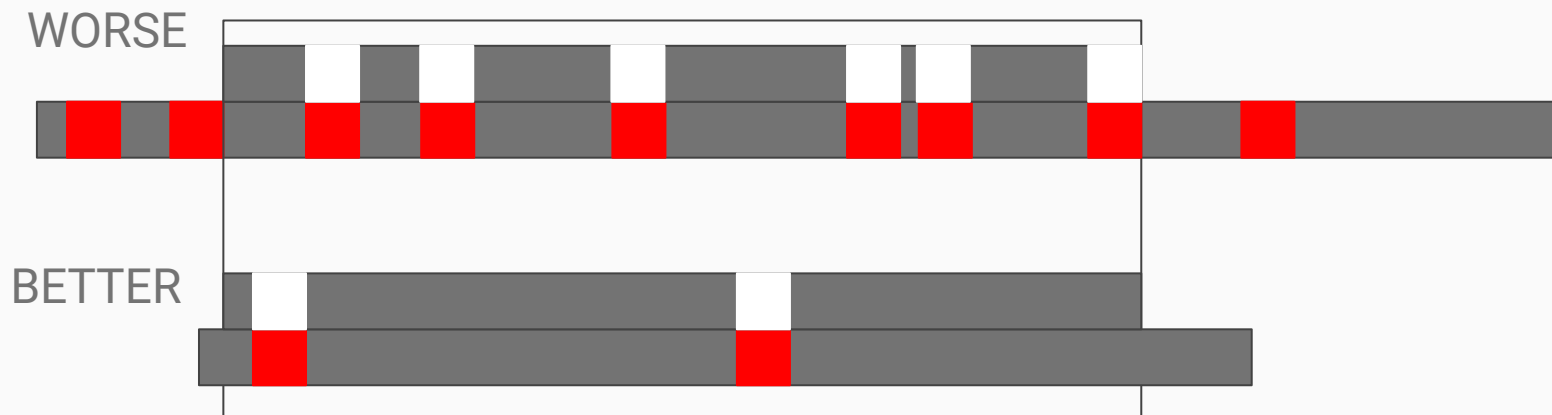
# FILLING LINES/BLOCKS/PAGES

ENSURE MEMORY ACCESSED IS CLOSE TO RECENTLY USED MEMORY



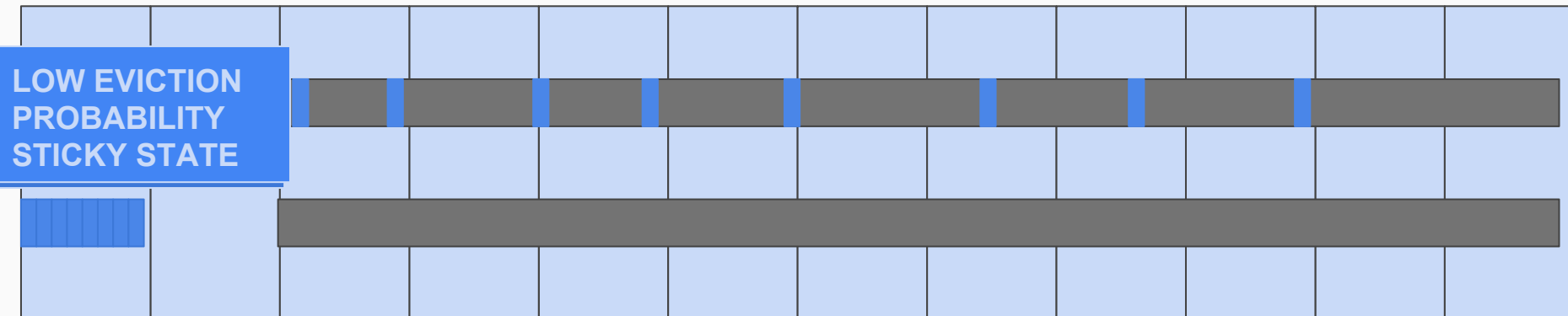
# REDUCE MEMORY USE

IF LESS MEMORY IS ACCESSED BOTH THE CHANCE OF EVICTION IS REDUCED  
AND FILL RATIO IMPROVED



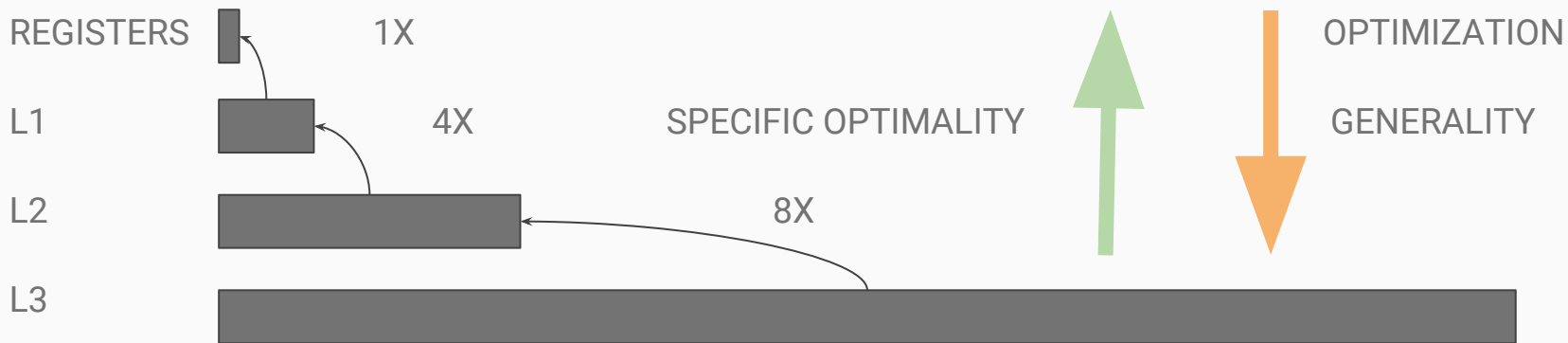
# LRU LOCALITY OF REFERENCE

LEAST RECENTLY USED BLOCKS GETS EVICTED ON ACCESSES IF CACHE IS FULL. ISOLATE PARTS OF DATA STRUCTURE WHICH WILL GET USED MOST OFTEN. EVEN IF STATE HAS TO BE REPLICATED. THESE PARTS BECOMES STICKY AND REMAIN CACHED MORE OFTEN



# MIRRORING THE CACHE HIERARCHY

SMALL BUT FREQUENTLY USED PARTS OF A STANDARD DATA STRUCTURE ARE MIRRORED TO HAVE BETTER LOCALITY IN DIFFERENT CONTEXTS. MIRRORING IS AT EXPENSE OF SPACE



SOMETIMES IT'S POSSIBLE TO EXTRACT THE MOST FREQUENTLY USED PARTS WITHOUT USING EXTRA SPACE

# HASH TABLES

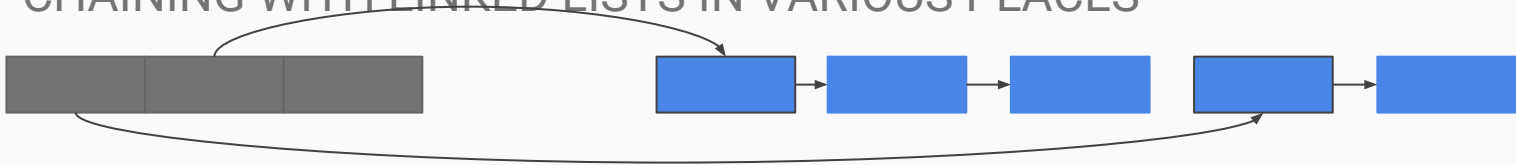
OPEN ADDRESSING (BETTER LOCALITY OF REFERENCE)

- I.E. LINEAR PROBING, ROBIN HOOD HASHING



CLOSED ADDRESSING (WORSE LOCALITY OF REFERENCE)

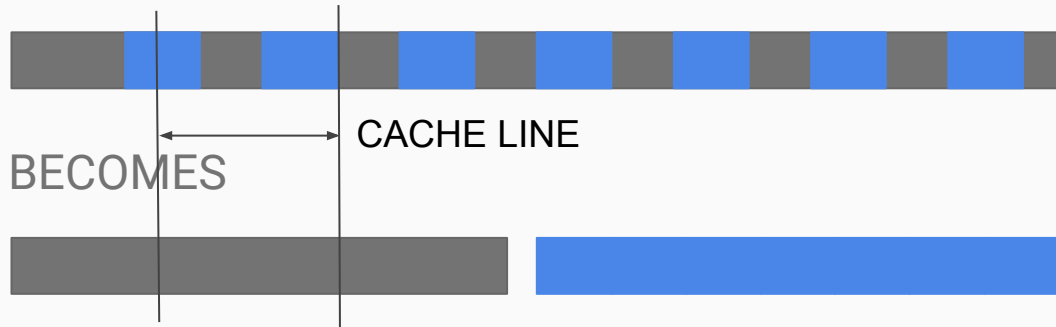
- CHAINING WITH LINKED LISTS IN VARIOUS PLACES








# KEY VALUE SEGREGATION

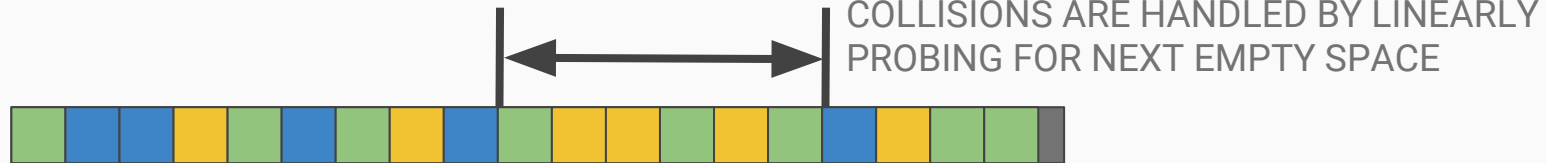
SPLITTING KEYS AND VALUES LET'S CACHE LINES/PAGES CONTAIN MORE RELEVANT DATA. MOST EFFECTIVE WHEN LOAD FACTOR IS HIGH



# LINEAR PROBING

EACH KEY HAS ONE OF THREE STATES

- OCCUPIED 
- EMPTY 
- DELETED 



# LINEAR PROBING ISSUES

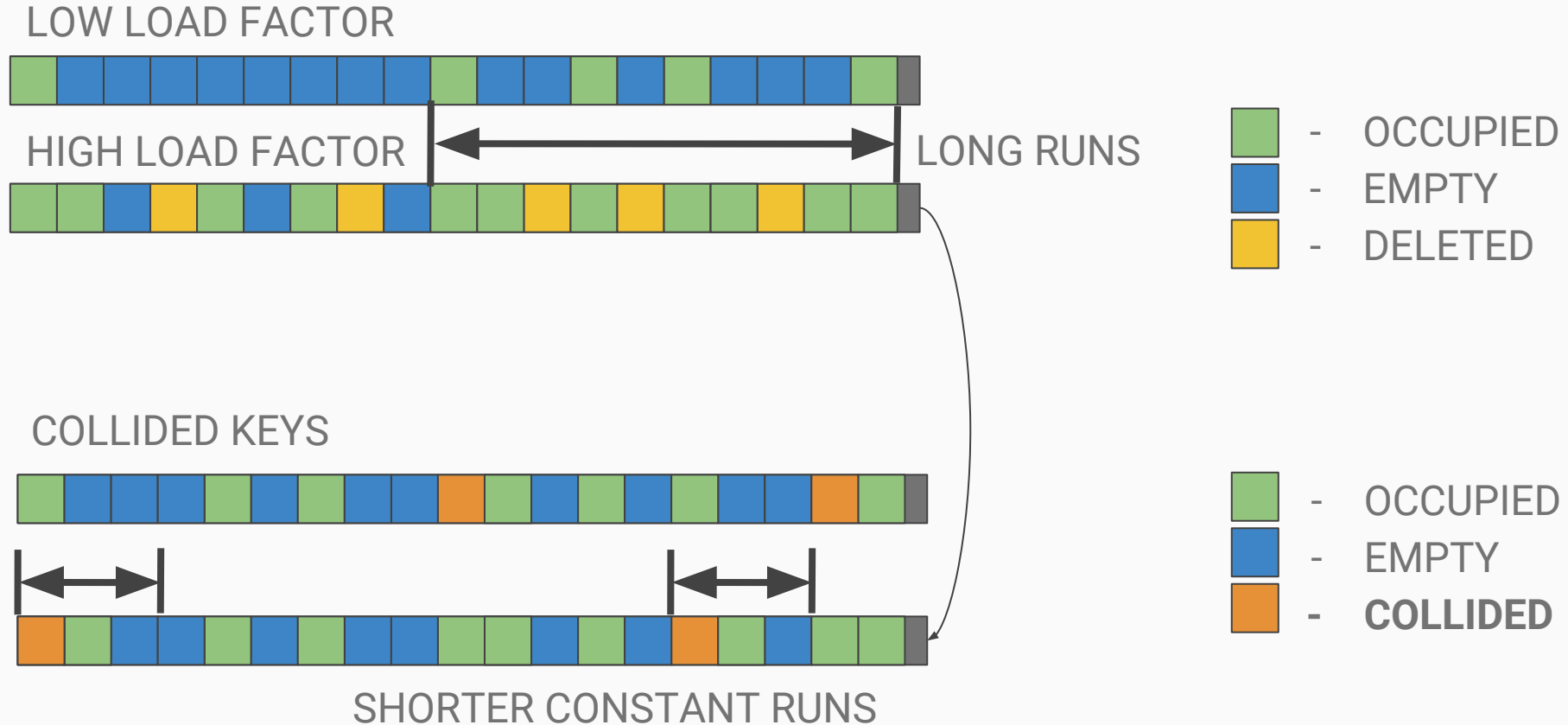


TABLE NEEDS TO MAINTAIN RANDOM AND UNIFORMLY DISTRIBUTED KEYS TO REMAIN EFFICIENT. THERE NEEDS TO BE ENOUGH EMPTY KEYS FOR ITERATIONS TO STOP

THIS LEADS TO POOR LOCALITY OF REFERENCE

- SOLUTIONS LIKE QUADRATIC PROBING ALSO HAS POOR LOCALITY OF REFERENCE

# LOAD FACTOR AND COLLISION FACTOR



## THE REST

OVERFLOWS STORED IN SINGLE BUCKET AT THE END. NO POINTERS REQUIRED AND TABLE SIZE/REHASHING IS REDUCED

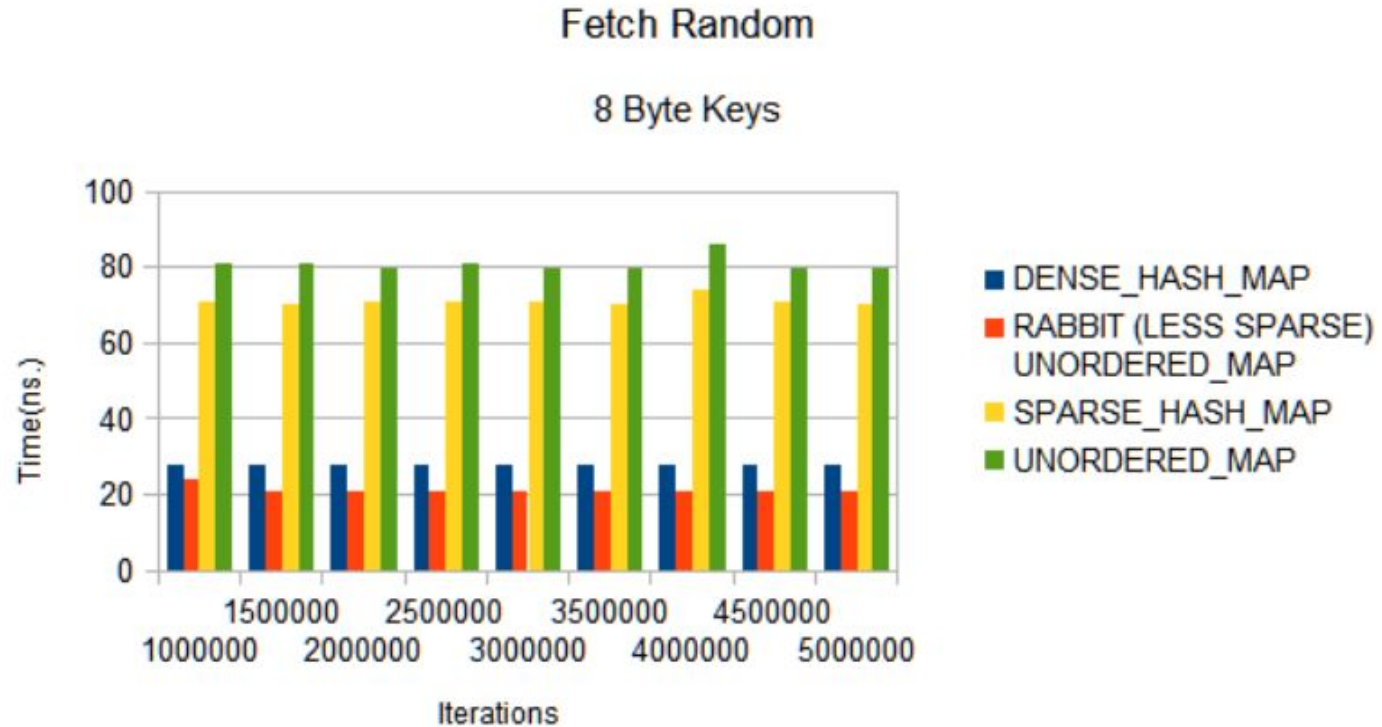


STATE IS STORED AS BITMAP INSTEAD OF KEYS WHICH SIMPLIFIES API AND CREATES STICKY STATE WHICH IS MORE CACHE FRIENDLY

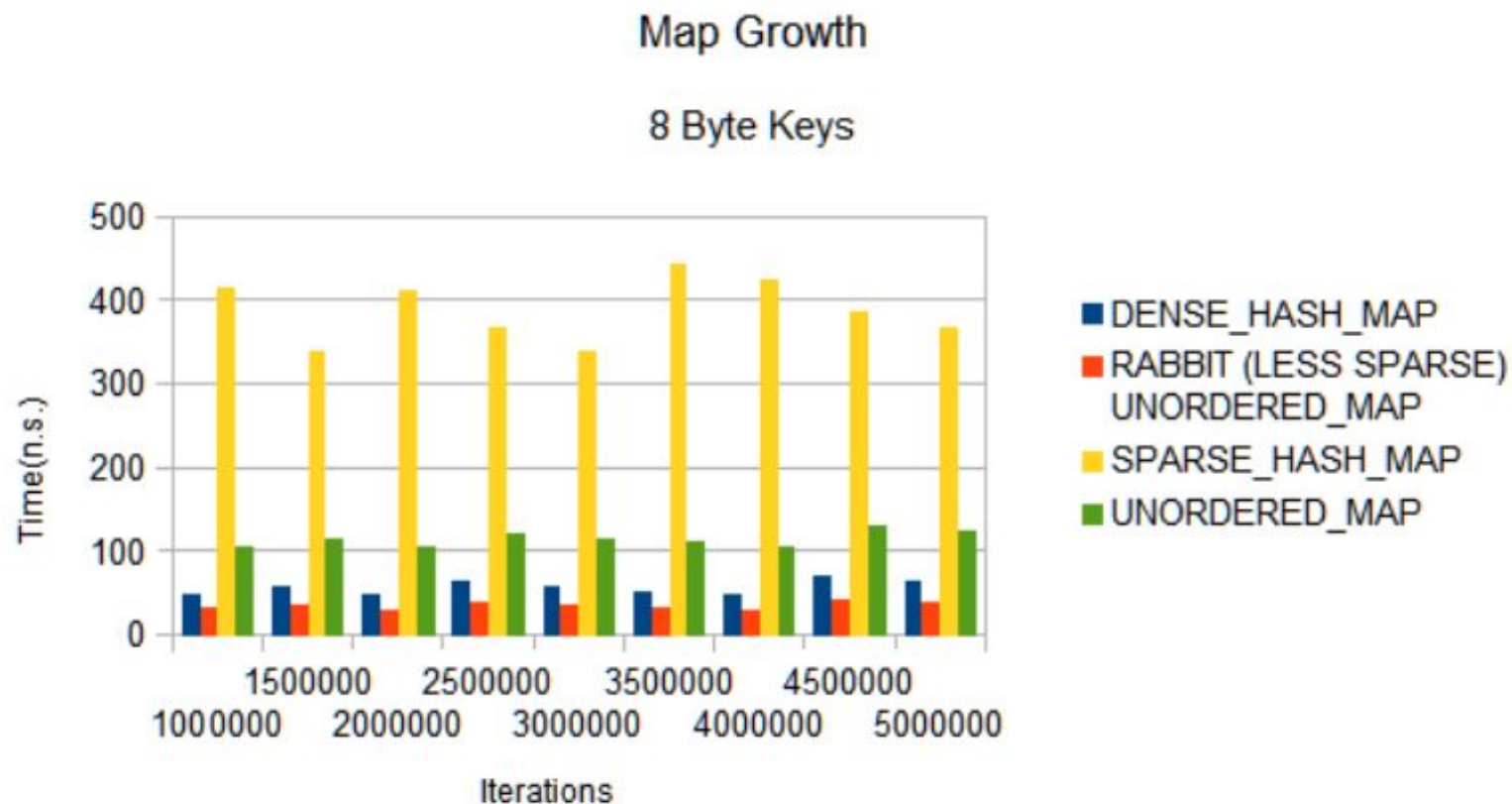
# BENCHMARKS

## COMPILERS

GCC 5.1 x64



## BENCHMARKS CONTINUED



## BENCHMARKS CONTINUED MORE...

