The Colored Refresh Server for DRAM

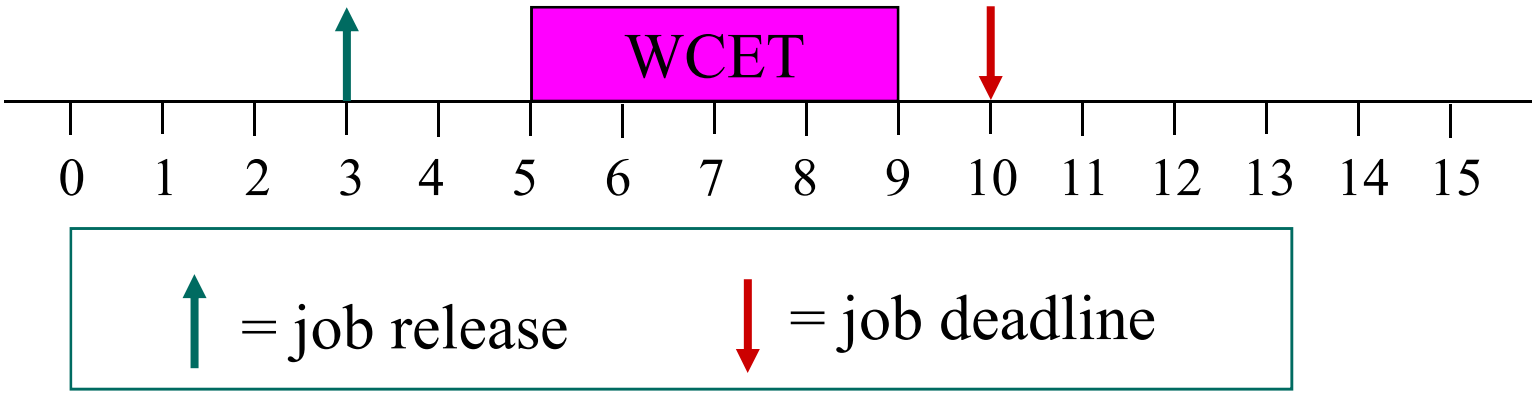
Xing Pan, Frank Mueller

North Carolina State University

NC STATE UNIVERSITY
Department of Computer Science

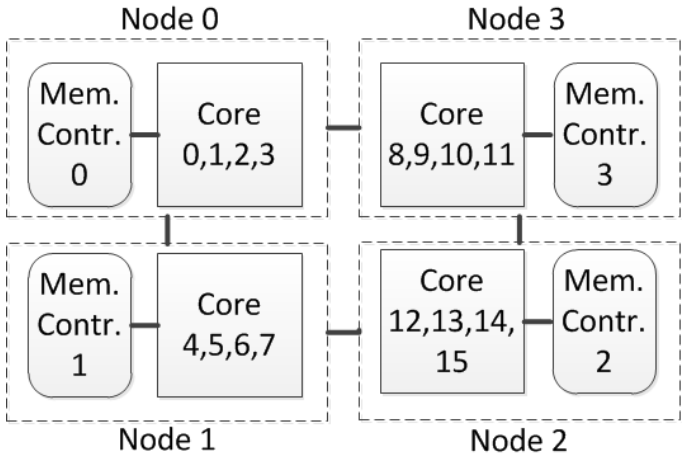
Real-time system

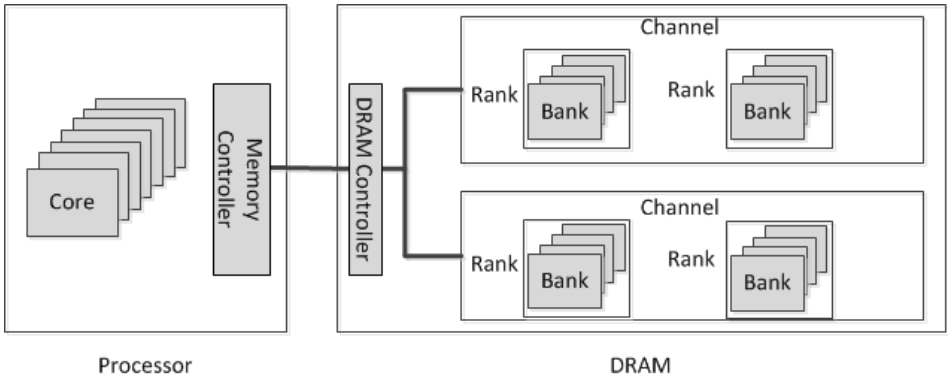
- Real-Time System requires:
  - Logical Correctness: Produces correct outputs.
  - Temporal Correctness: Produces outputs at the right time.
- Real-time task
  - predict its worst-case execution time
  - schedule it to meet its deadline



NUMA Architecture

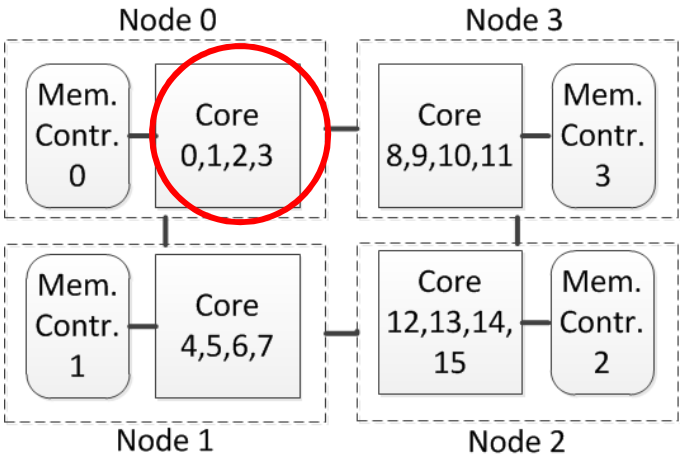
- Modern NUMA (non-uniform memory access) architectures:
  - CPU partitions sets of cores into "node":
    1 local + several remote controllers
  - Each memory controller (node) consists of multilevel resources (channel, rank and bank)





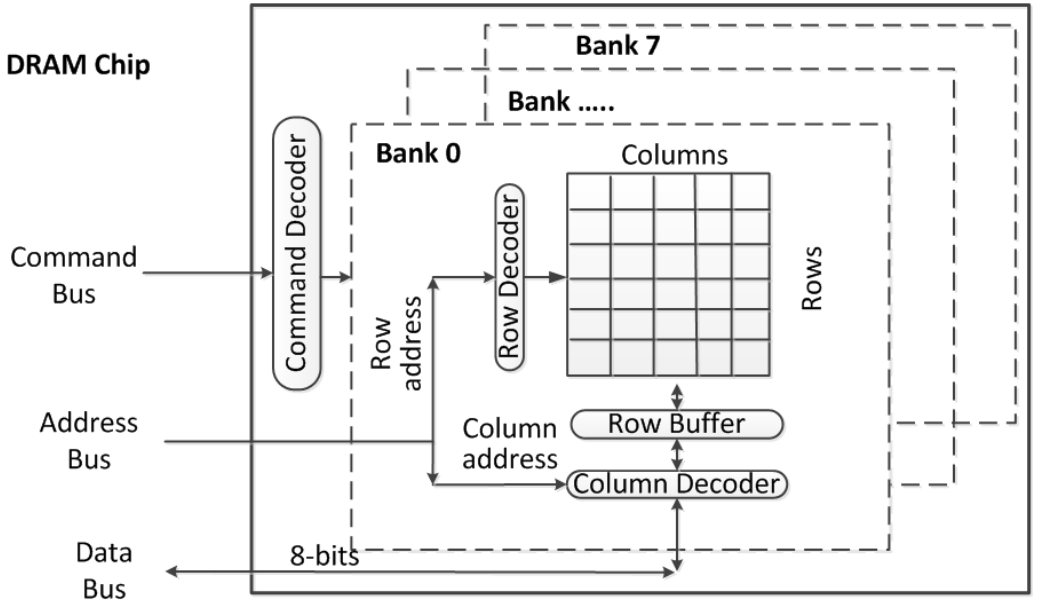
Core Isolation → Hard Real-Time Composition

- Challenge: shared resources
  - One core execution affects other cores
- Objective: Isolate cores
  - Allows compositional timing analysis
- Application: mission critical hard real-time
  - Automated driving…



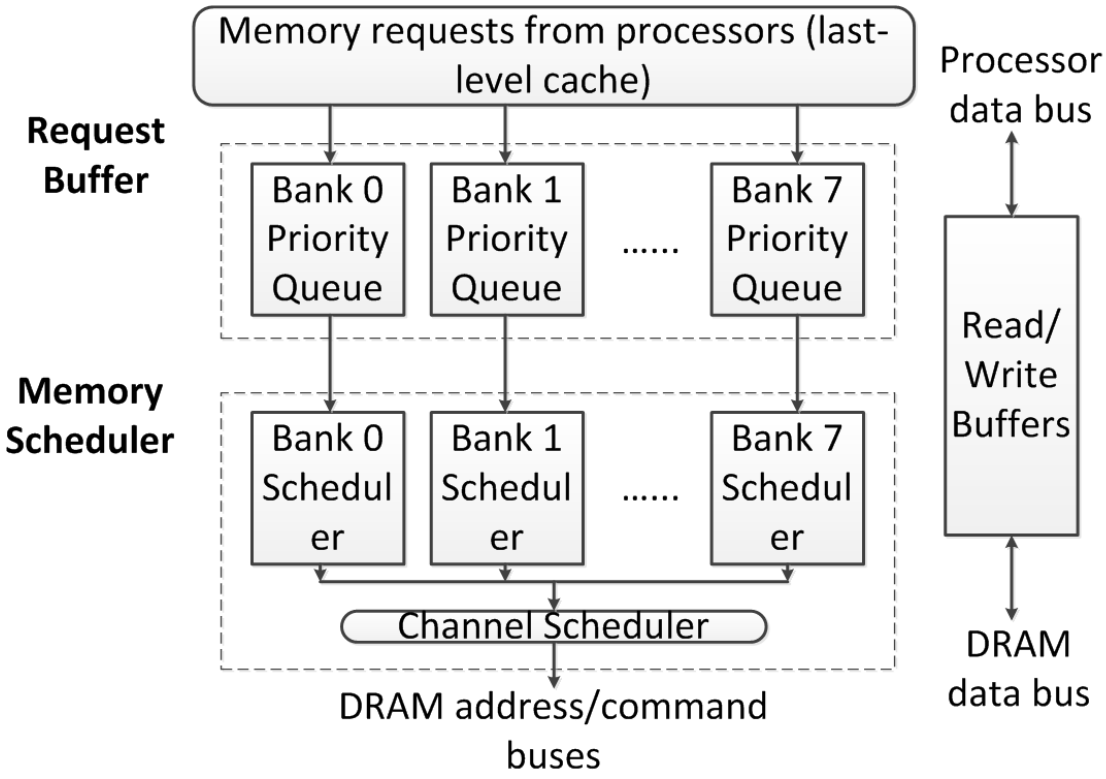
DRAM Organization

- DRAM bank array has: rows+columns of data cells
- Load the row which contains requested data into Row Buffer
  - Row Buffer hit vs. Row Buffer miss



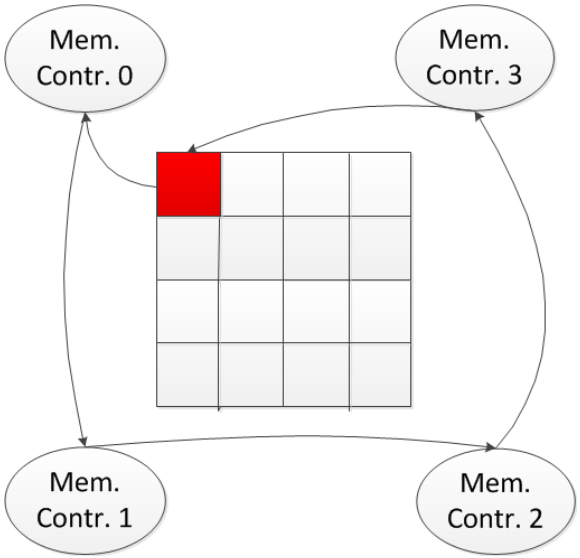
Memory Controller

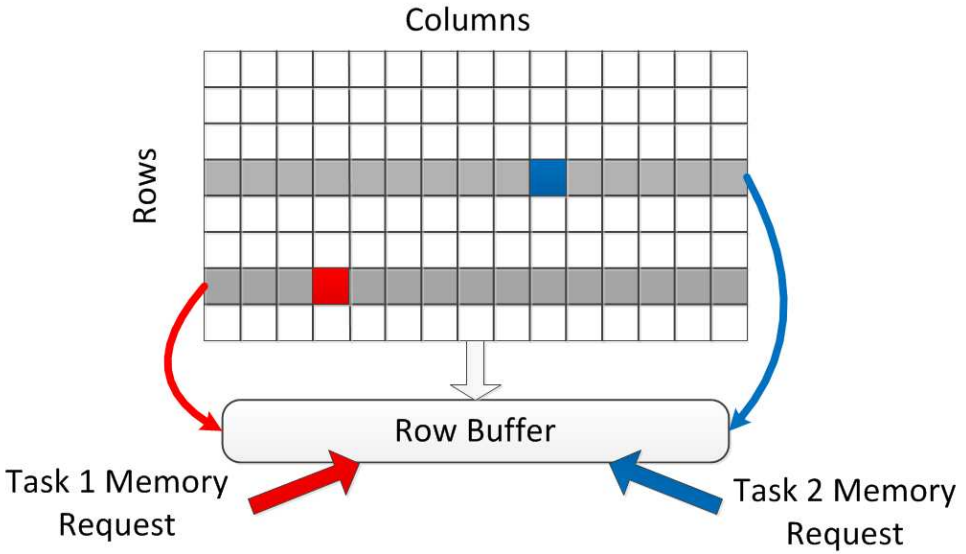
- DRAM banks can be accessed in parallel



Motivation

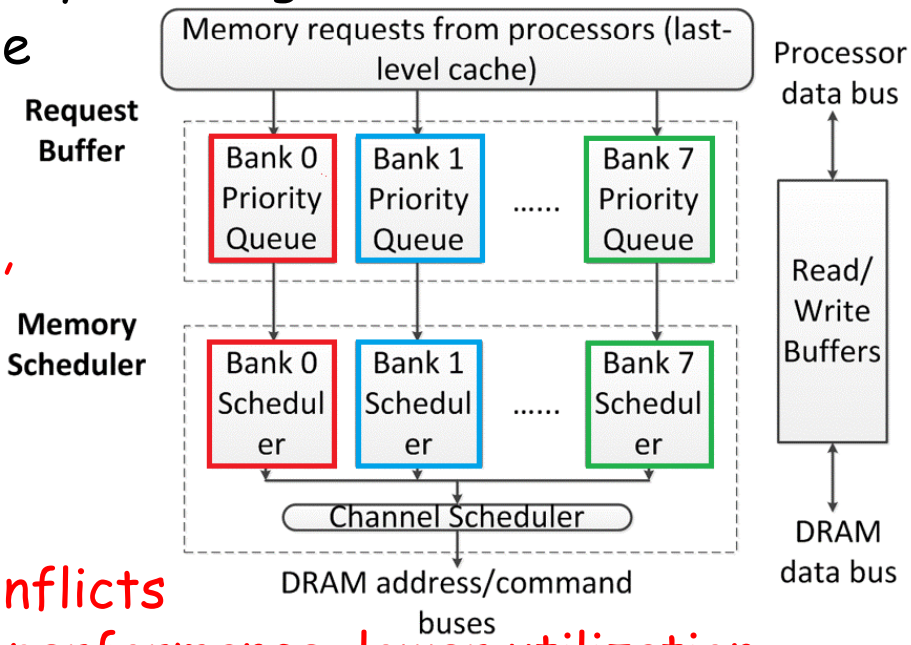
- Apps on NUMA arch. experience varying execution times due to
  - Remote memory node accesses
  - Conflict in memory banks/controllers



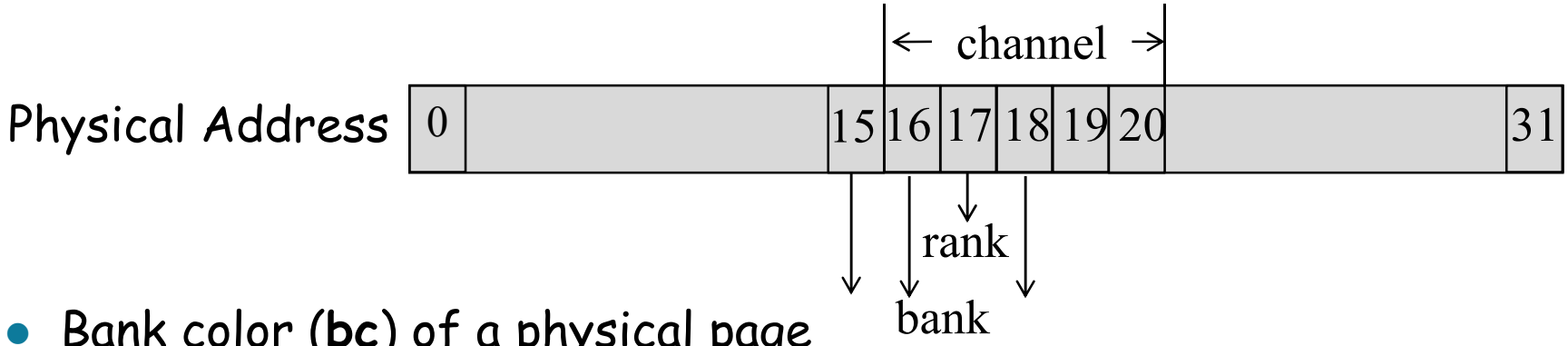


Past: Memory Predictability by Coloring

- Local node policy under standard buddy allocation / numa library
  - Not bank aware
  - numa library only works on heap memory
- Previous Work
  - Our Controller-Aware Memory Coloring (CAMC) @ SAC'18
  - NUMA causes unpredictable execution time
  - New memory allocator in kernel via mmap() syscall, no hardware modifications
  - Each task gets private memory (coloring) on local NUMA node
  - Avoid remote refs, bank conflicts → predictable exec., lower performance, lower utilization



Memory Frame Color Selection



- Bank color (**bc**) of a physical page

  bc = ((node∗NN ∗NC+channel)∗NR+rank)∗NB+bank

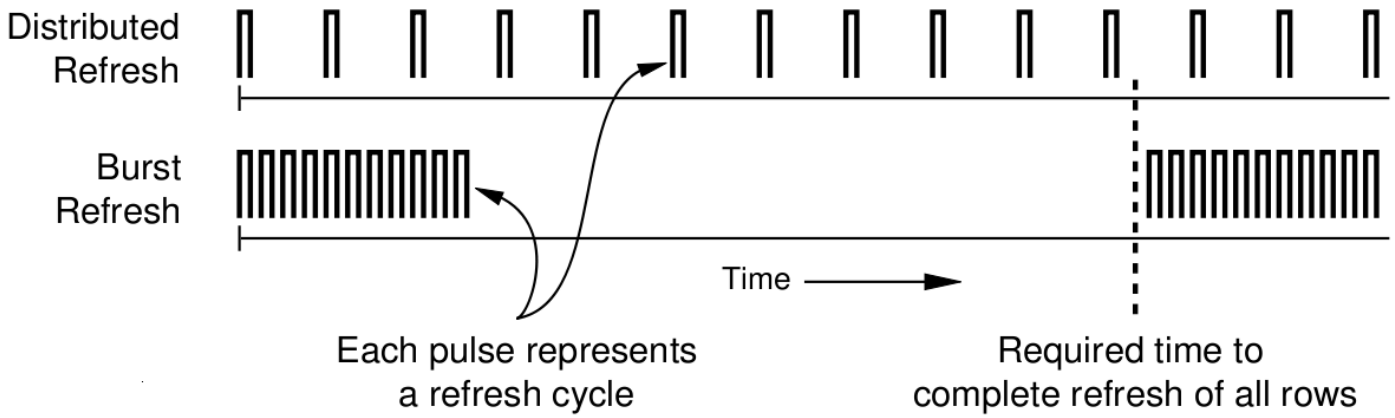
  - NN: # nodes (mem controllers) of a system
  - NC: # channels per controller
  - NR: # ranks per channel
  - NB: # banks per rank
- Opteron 6128: NN=4, NC=2, NR=2, NB=8, Total of 128 colors
- Example: page in node 0, channel 1, rank 1 and bank 2
  → color is ((0∗4∗2+1)∗2+1)*8+2=26

Focus in this Paper: DRAM Refresh

- Dynamic Random Access Memory (DRAM)
  - data is stored in the capacitor as 1 or 0 (electrically charged/discharged)
  - capacitors slowly leak their charge over time
  - requires cells to be refreshed, otherwise data would be lost.

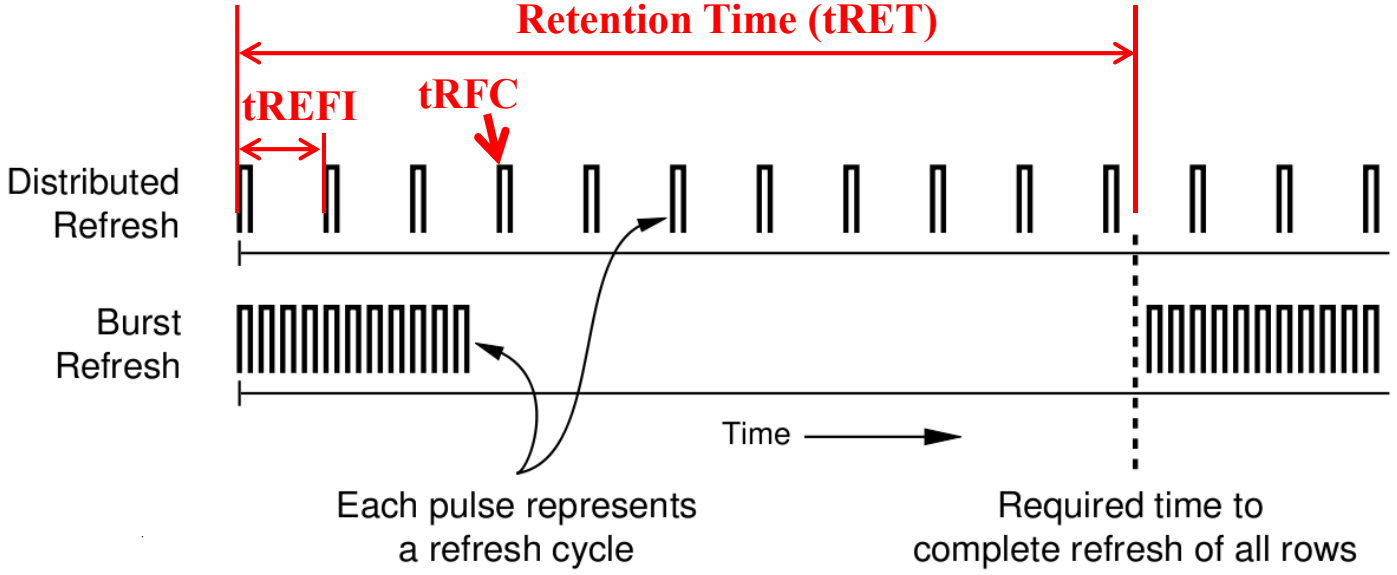
Unpredictability due to DRAM Refresh

- Refresh commands to all DRAM cells periodically issued by DRAM controller to maintain data validity.
  - row-buffer is closed
  - any memory access deferred until refresh completes
- Distributed Refresh vs. Burst refresh



Unpredictability due to DRAM Refresh

- Refresh commands to all DRAM cells periodically issued by DRAM controller to maintain data validity
  - row-buffer is closed
  - any memory access deferred until refresh completes
- Distributed Refresh vs. Burst refresh



DRAM Refresh Trends: It's getting worse

- tRET: 64 ms / 32 ms. determined by temperature (85 C)
- tRFC increases quickly with growing DRAM densities

Chip Density	# banks	#rows/bank	#rows/bin	tRFC
1Gb	8	16K	16	110 ns [1]
2Gb	8	32K	32	160 ns [1]
4Gb	8	64K	64	260 ns [1]
8Gb	8	128K	128	350 ns [1]
16Gb	8	256K	256	550 ns [2]
32Gb	8	512K	512	> 1 us [3]
64Gb	8	1M	1K	> 2 us [3]

- [1] Standard, JEDEC, DDR3 SDRAM
- [2] Standard, JEDEC, DDR4 SDRAM
- [3] Jamie Liu, Onur Mutlu et al. "RAIDR: Retention-aware intelligent DRAM refresh." *ACM SIGARCH Computer Architecture News*. 2012.

Challenge: Refresh Delay

- Auto-refresh : recharges all the memory cells within the "retention time"
  - a rank during refresh becomes unavailable to memory requests until the refresh completes (tRFC).
  - all bank row buffers of this rank closed (tRP) and need to be re-opened (tRAS)
  - More bank row buffer misses around refreshes.

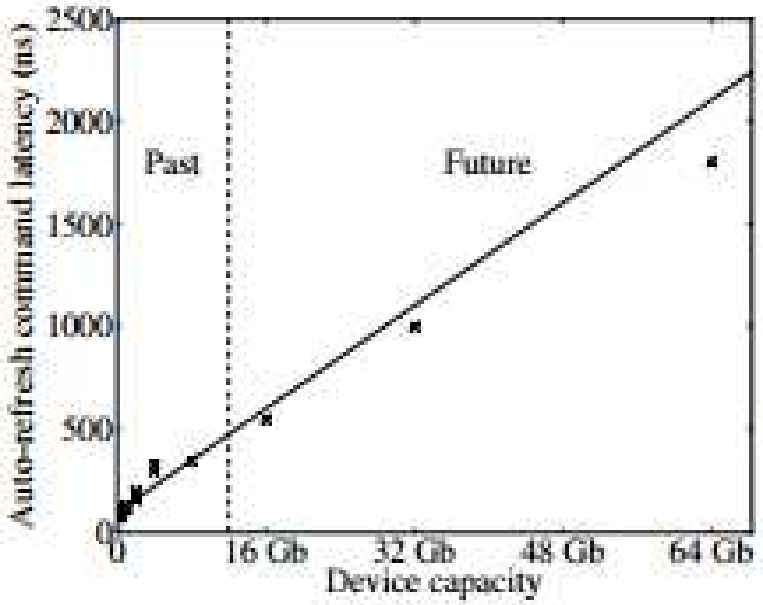
Challenge: Refresh Delay

- Auto-refresh : recharges all the memory cells within the "retention time"
  - a rank during refresh becomes unavailable to memory requests until the refresh completes (tRFC).
  - all bank row buffers of this rank closed (tRP) and need to be re-opened (tRAS)
  - More bank row buffer misses around refreshes.

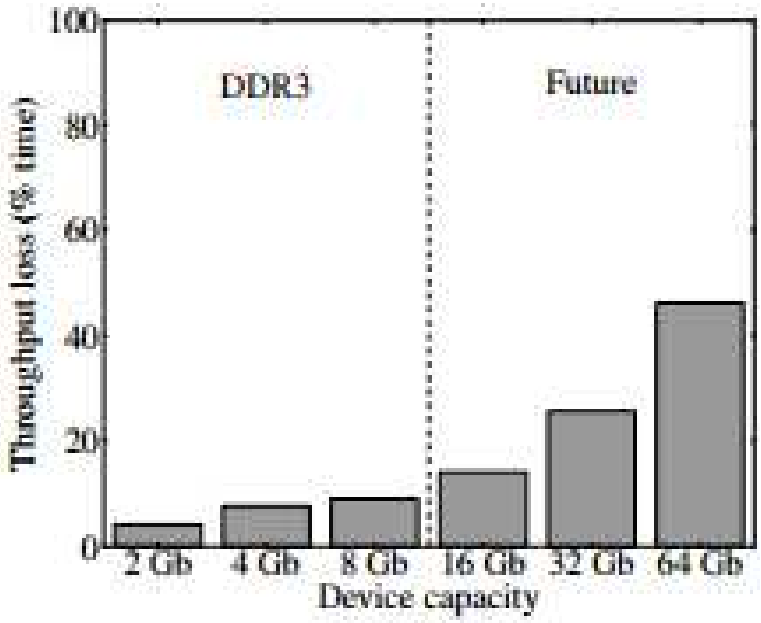
1. Increase in memory latency
2. Significant fluctuation of memory reference latency.

Challenge: Refresh Delay

- As density and size of DRAM grow:
  - more rows required per DRAM chip
  - longer tRFC
  - higher probability for refresh interference



(a) Refresh latency



(b) Throughput loss

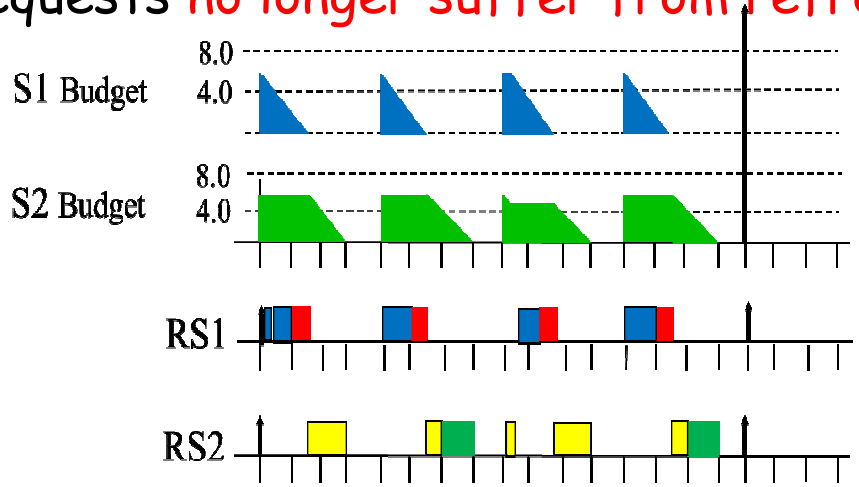
Challenge: Refresh Delay

- As density and size of DRAM grow:
  - more rows required per DRAM chip
  - longer tRFC
  - higher probability for refresh interference

1. Increases length a refresh operation
2. Reduces memory throughput

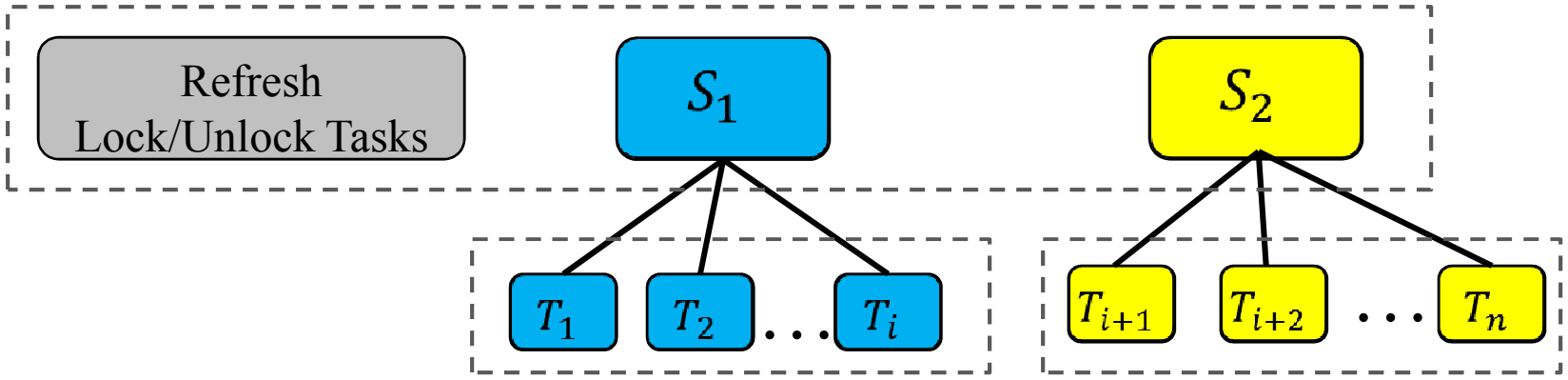
Solution: Colored Refresh Server (CRS)

- Partition DRAM memory at rank granularity
  - Refreshes rotate round-robin from rank to rank
  - Assign real-time tasks to different ranks via colored memory allocation (say: green,blue)
  - Schedule 2 server tasks to refresh green/blue memory
  - Ensure that no blue task runs when green server active and vice versa: no green task runs when blue server active
- Cooperative scheduling real-time tasks and refresh operations → memory requests no longer suffer from refresh interference



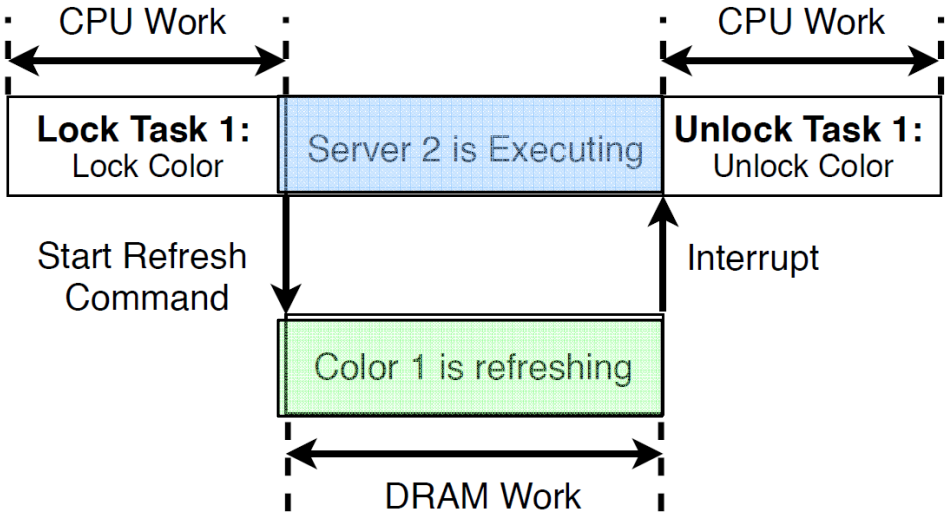
Architecture of Colored Refresh Server

- Hierarchical model
  - System Level
    - Refresh tasks w/ static priority: Refresh Tasks > S_1 > S_2 tasks
  - Server Level (inside the servers)
    - User tasks scheduled inside servers
      - w/ memory colored diametric to server
    - with any real-time scheduling policy: EDF, RM, …
    - Refresh Lock/unlock tasks: no memory blocking during refresh



Refresh Lock and Unlock Tasks

- partition entire DRAM space into two "colors"
  - e.g., $c_1(k_0, k_1 \ldots k_i)$, and $c_2(k_{i+1}, k_{i+2} \ldots k_{K-1})$.
- refresh lock tasks, and
  - period of tRET(64ms)
  - trigger refresh for c_1 (green) and c_2 (blue), respectively
- refresh unlock tasks, and
  - update corresponding color to be available once refresh finishes



Server Model

- Server model, S(W,A, c, p_s , e_s)
  - with CPU time as resource
  - Where:
    - W is the workload model (applications)
    - A is the scheduling algorithm, e.g., EDF or RM
    - c denotes the memory color assigned to this server, i.e., a set of memory ranks available for allocation
    - p_s is the server period
    - e_s is the server budget

Server Model

- Set execution budget to e_s at time instants $k * p_s$, where $k > 0$.
- Any unused execution budget cannot be carried over to next period

- The refresh server can execute when
  - (i) its budget is not zero;
  - (ii) its available task queue is not empty; and
  - (iii) its memory color is not locked by a "refresh task" (introduced above).
  - Otherwise, it remains suspended.

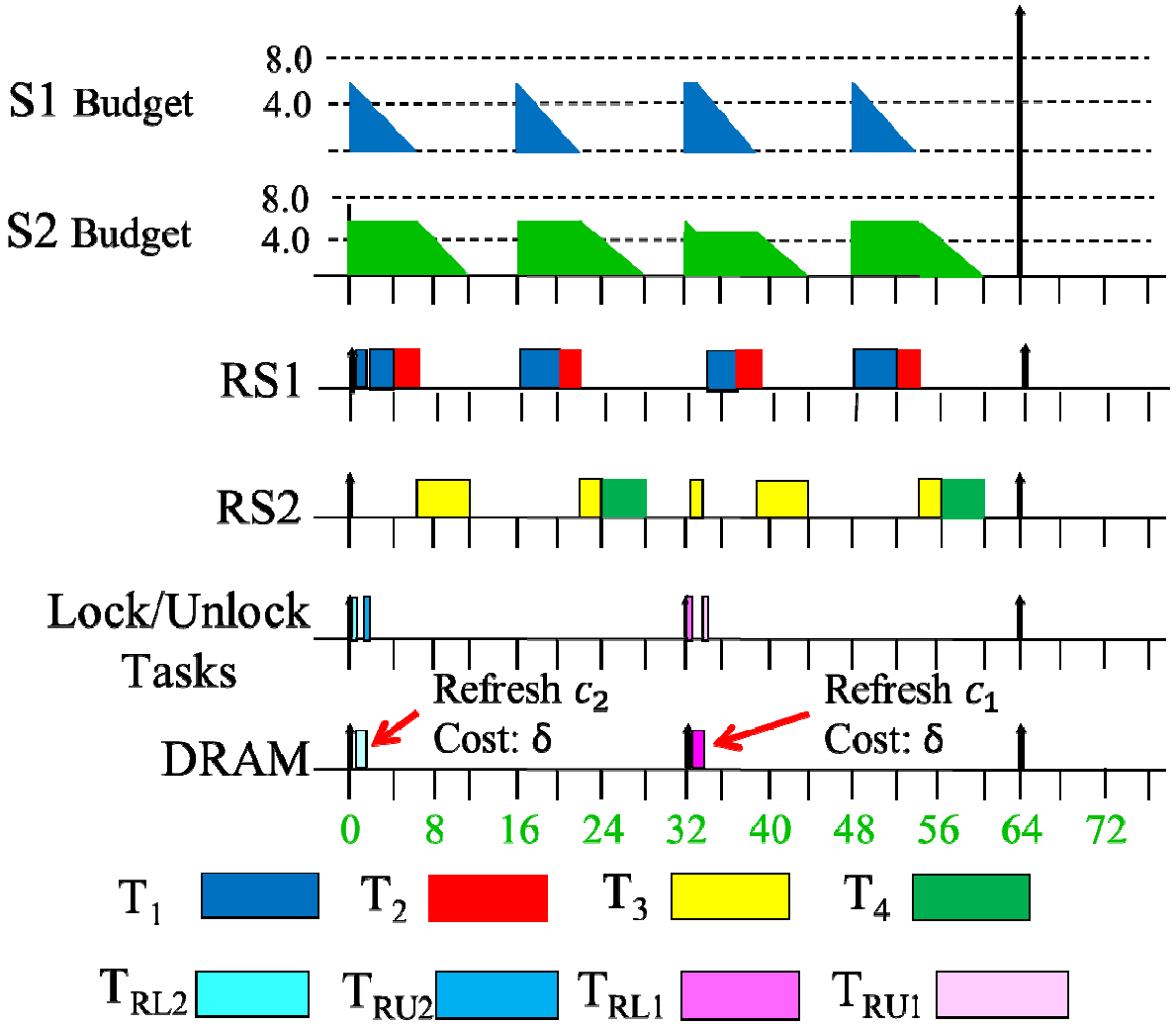
Example of CRS

- T_1(16ms, 4ms)
  T_2(16ms, 2ms)
  T_3(32ms, 8ms)
  T_4(64ms, 8ms)

- $S_1((T_1, T_2), RM, c_1(k_0,k_1,k_2,k_3), 16ms , 6ms)$
  $S_2((T_3, T_4), RM, c_2(k_4,k_5,k_6,k_7), 16ms , 6ms)$

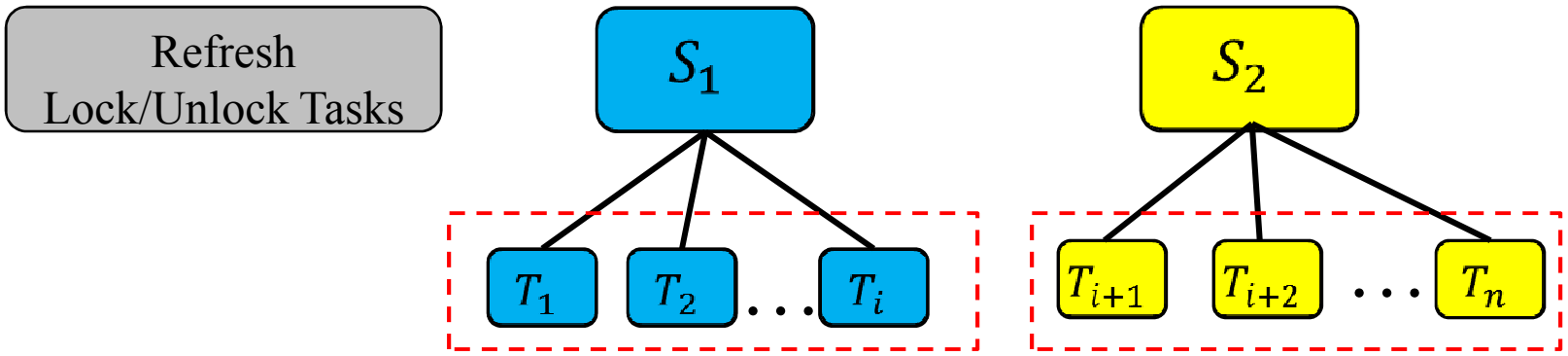
- Phases ϕ of S_1 and S_2 are tRET/2 and 0, respectively
  — i.e., S_2 (colors c_2) refreshed first

Example of CRS



Schedulability Analysis within a Server

- Given a server S(W,A, c , p_s , e_s) [SL03],
  - Periodic Capacity Bound (PCB):
    - bound period (p_s) and deadline (e_s)
    - with workload (W) and algorithm (A)
  - Utilization Bound (UB)
    - Bound utilization of workload
    - with p_s, e_s , and A
- [SL03] Shin, I. & Lee, I. "Periodic resource model for compositional real-time guarantees". RTSS. 2003.

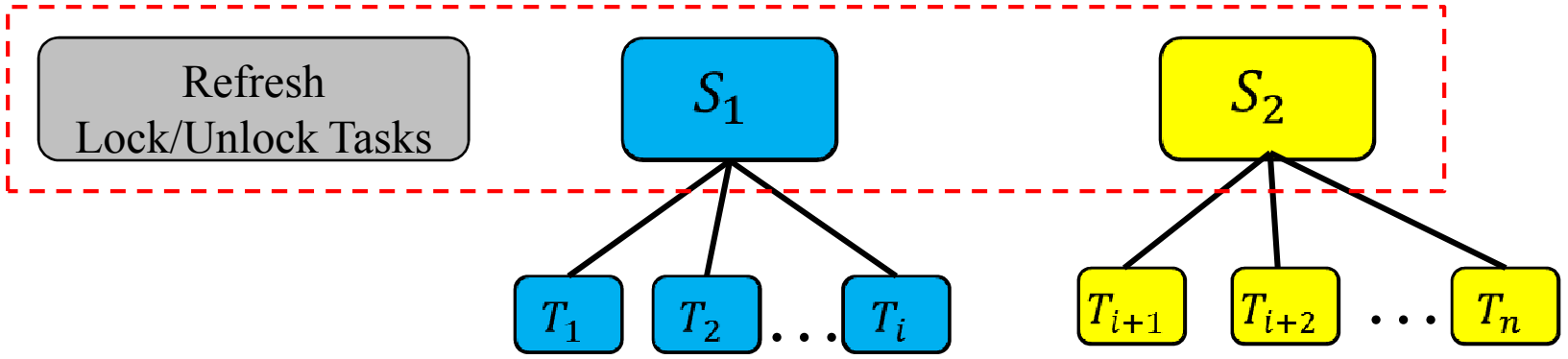


Schedulability Analysis

- Servers + refresh lock/unlock tasks at system level

$$
\begin{aligned}
&T_{rl1}\ (0,\ tRET,\ e_{rl},\ tRET),\\
&T_{rl2}\ (tRET/2,\ tRET,\ e_{rl},\ tRET),\\
&T_{ru1}\ (\delta,\ tRET,\ e_{ru},\ tRET),\\
&T_{ru2}\ (tRET/2+\delta,\ tRET,\ e_{ru},\ tRET),\\
&S_1\ (p_1,\ e_1),\ \text{and}\\
&S_2\ (p_2,\ e_2)
\end{aligned}
$$

- Time Demand Analysis
  - Refresh tasks w/ static priority: Lock/Unlock Tasks > S_1 > S_2



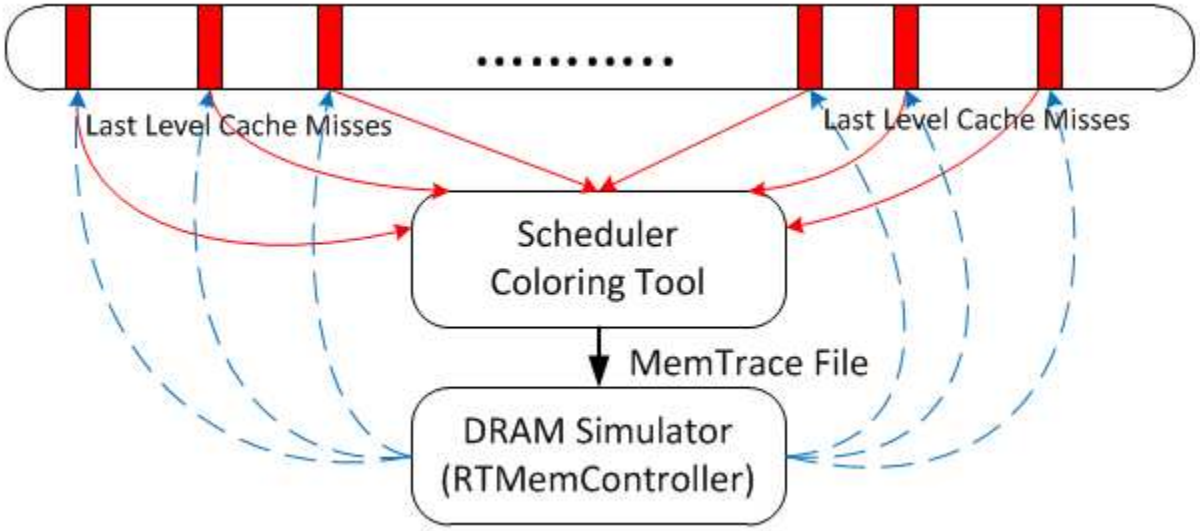
Colored Refresh Server Design

- Off-line algorithm
  - Searches entire range of available configurations
  - Find minimum refresh overhead & budgets for servers
    - Short tasks: create copy tasks
  - See dissertation [Pan'18]

- Colored Refresh Server
  - Guarantees schedulability
    (if task set was schedulable w/o CRS)
  - Cost much lower overhead than auto-refresh
    (removes entire refresh overhead in most cases)

Colored Refresh Server Implementation

- SimpleScalar
  - simulates execution of application
  - generates memory tracefile
- Scheduler & Coloring Tool (from CAMC [SAC'18] work)
- RTMemController (only to obtain timings, no Ethereal support)
  - schedule memory transactions, determine access latency



Experimental Setup

- Single core processor
  - split 16KB data and instruction caches,
  - unified 128KB L2 cache
  - cache line size is 64B.
- JEDEC-compliant DDR3/DDR4 SDRAM
  - varied memory density: 1/2/4/8/16/32/64Gb)
- The DRAM retention time: tRET=64ms
  - 8 ranks (K=8) & 1 memory controller.
  - Issue refresh by memory controllers at rank granularity.

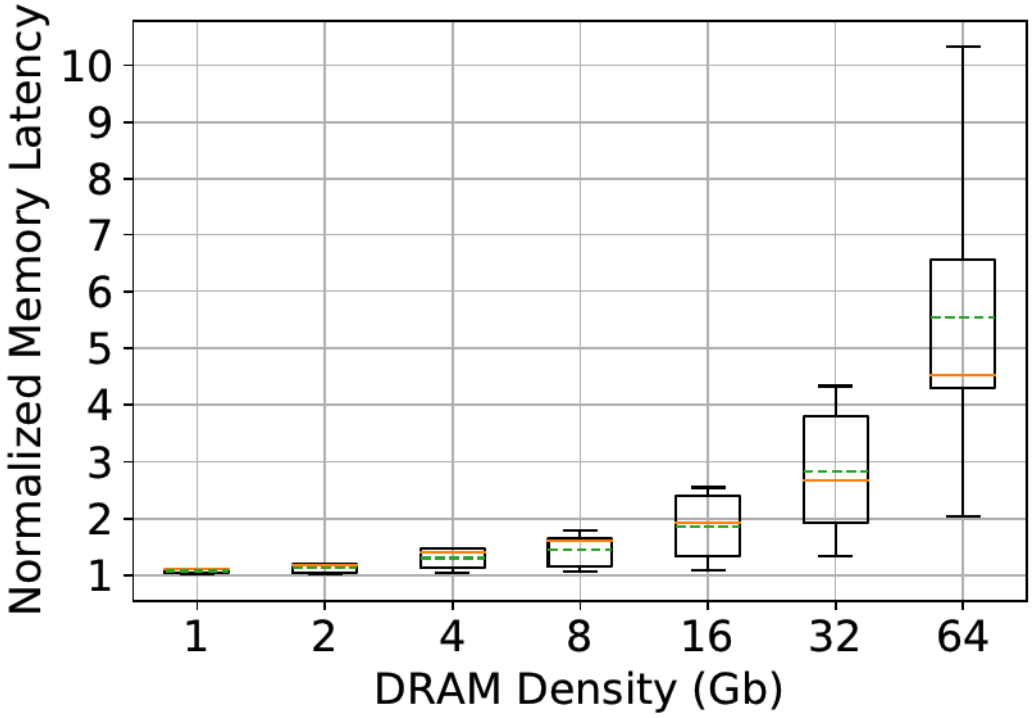
Real-Time Tasks

- Malardalen benchmark task set
- S_1((cnt, lms, st), EDF, $c_1(k_0,k_1,k_2,k_3)$, 4ms, 2.4ms)
  S_2((compress, matmult), EDF, $c_2(k_4,k_5,k_6,k_7)$, 4ms, 1.6ms)

	Execution Time	**Period**
cnt	3 ms	20 ms
compress	1.2 ms	10 ms
lms	1.6 ms	10 ms
matmult	10 ms	40 ms
st	2 ms	9 ms

Evaluation

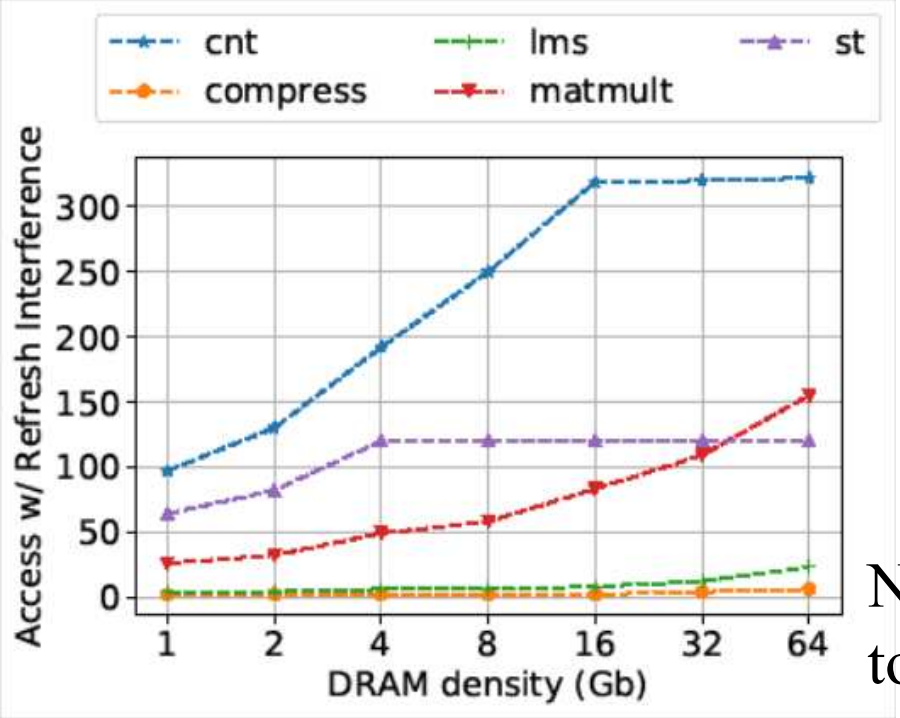
- CRS hides memory latency penalty of auto-refresh, which increases with memory density under autorefresh.



AutoRefresh Latency Normalized to CRS

Evaluation

- Auto-refresh has increasing probability (more accesses) of memory references to interfere with each other with higher DRAM density (depends on memory access patterns in benchmarks) while CRS eliminates this variability

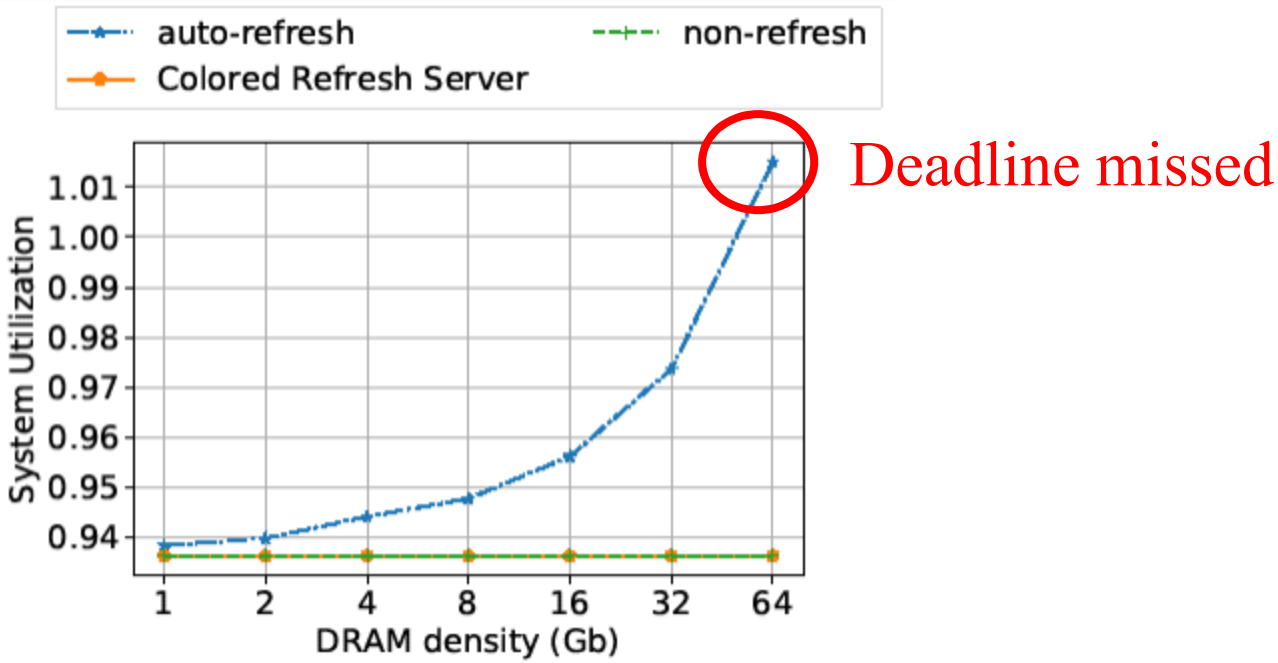


Normalized to CRS

Evaluation

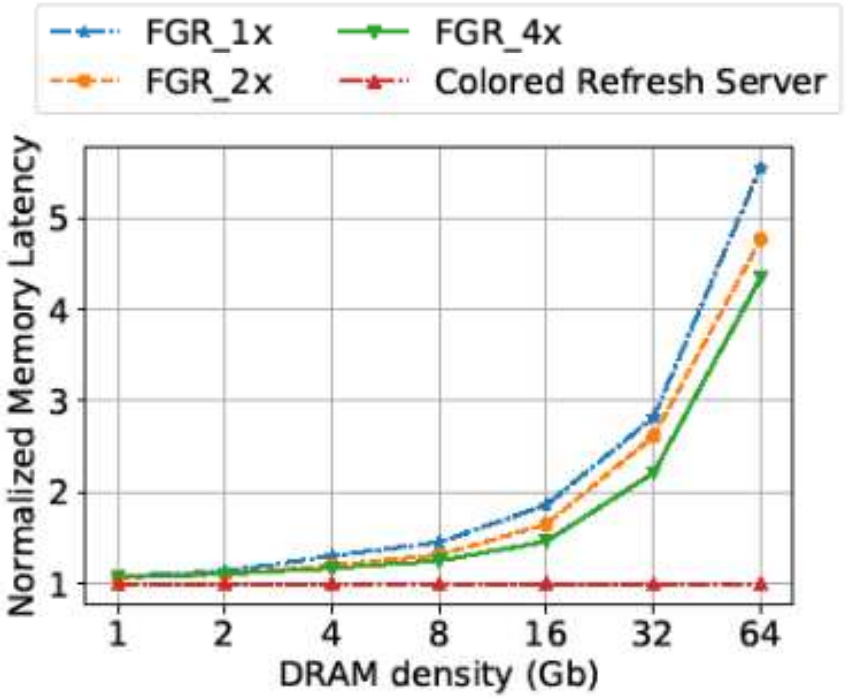
- Compared to auto-refresh,
  - CRS reduces execution time of tasks and system utilization
  - performance of CRS remains stable and predictable irrespective of DRAM density.

➢ **CRS as good as it gets → same as hypothetical "no refresh"**



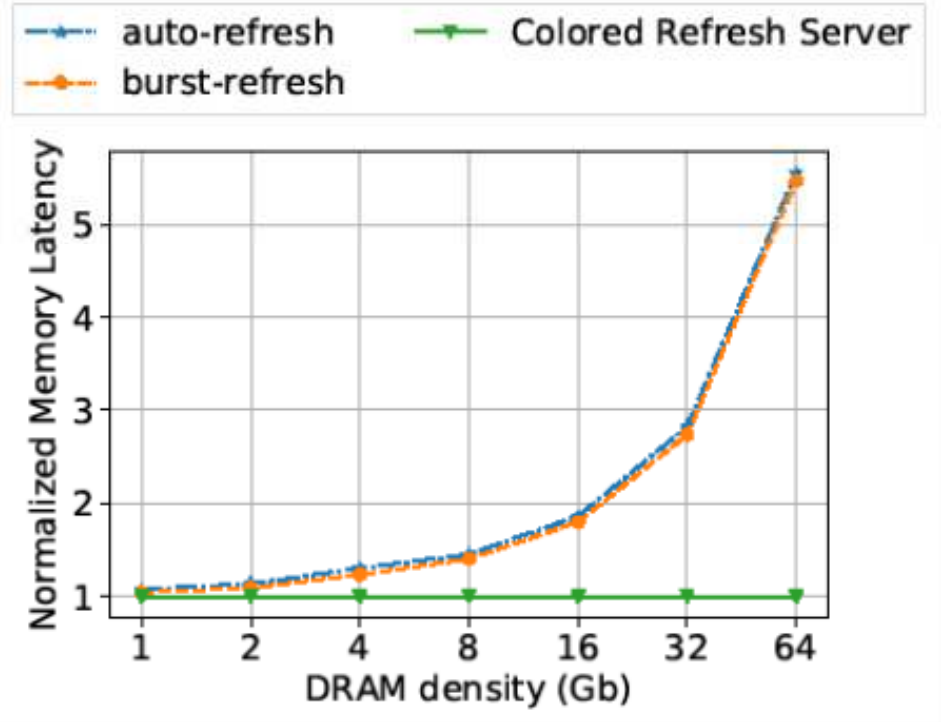
Evaluation

- DDR4 Fine Granularity Refresh (FGR)
  - Create a range of refresh options
  - Provide a trade-off between refresh latency and frequency.
- CRS exhibits better performance and higher task predictability than DDR4's FGR.



Evaluation

- CRS obtains better performance and higher task predictability than burst refresh of the closest prior work. [BM10]



- [BM10] Bhat, Balasubramanya & Mueller, Frank "Making DRAM refresh predictable", ECRTS 2010

Conclusion

- Make memory references more predictable w/ coloring
  - Controller-Aware Memory Coloring (CAMC) [SAC'18]
    - reduce varied memory access latency
    - provide single core equivalence **but** subject to refresh delay
- ***Colored Refresh Server***:
  - hide refresh delays & reduce DRAM access latencies
  - exhibit better performance & higher task predictability than auto-refresh & [BM'10]
  - hierarchical server task scheduling, apps inside servers
  - supports any real-time scheduling policy in server (EDF, RM)
  - realized in software, applicable to commercial off-the-shelf (COTS) systems.
- **Supports Core Isolation → real-time composability**
- supported in part by NSF grants 1239246,1329780,1525609 and 1813004.