Introduction to Design Considerations of DRAM Memory Controllers

Agenda

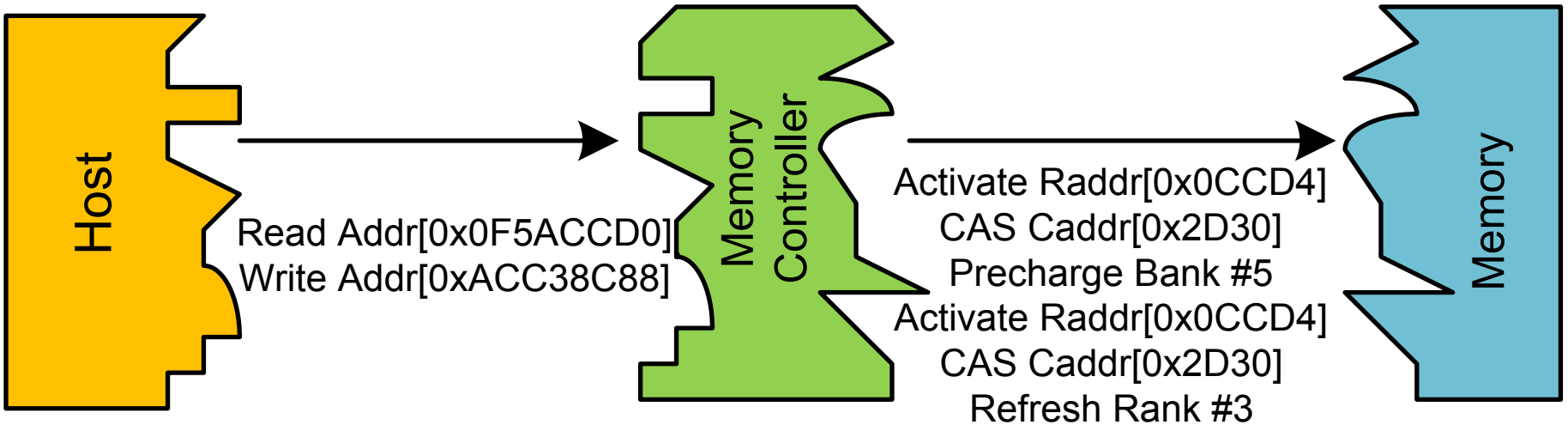
- Self Introduction
- Overview
- Initialization, Training and DRAM PHY
- Transaction Interface
- DRAM Paging Policy and Address Mapping
- Operating Frequency and Latency
- DRAM Command and Transaction Scheduling
- DRAM Power Management
- Reliability, Availability, Serviceability Features
- Summary

Self Introduction

- Co-Author of *Memory Systems: Cache, DRAM, Disk*
- PhD, University of Maryland
  - Memory Systems Performance Analysis
- Distinguished Engineer, Inphi Corporation
- Lead Memory Systems Architect, MetaRAM
- Lead development of multiple generations of memory buffering devices
  - Looks like memory devices to memory controller
  - Looks like memory controller to memory devices

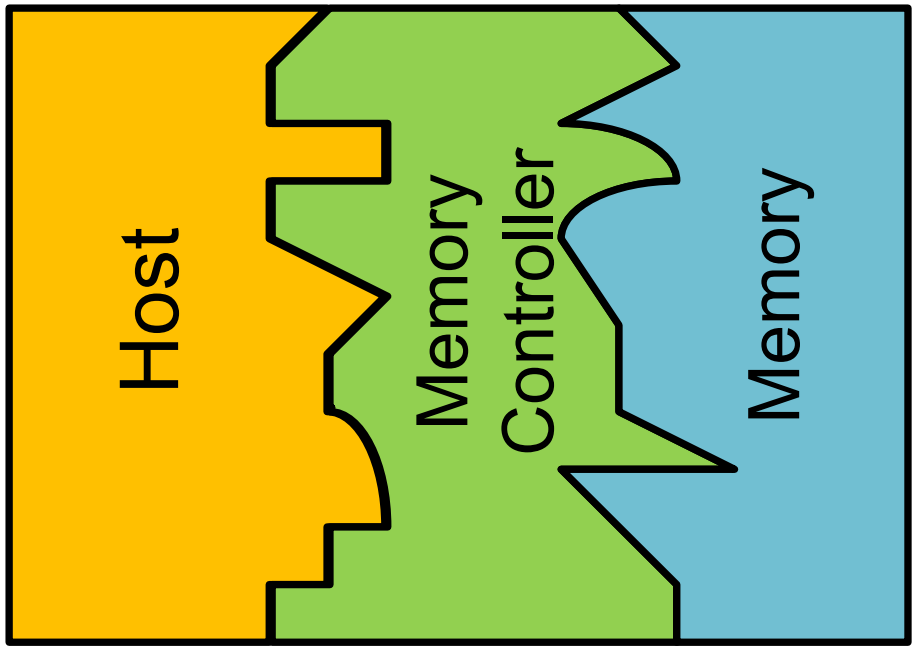
Overview

Host (CPU), Memory Controller and Memory



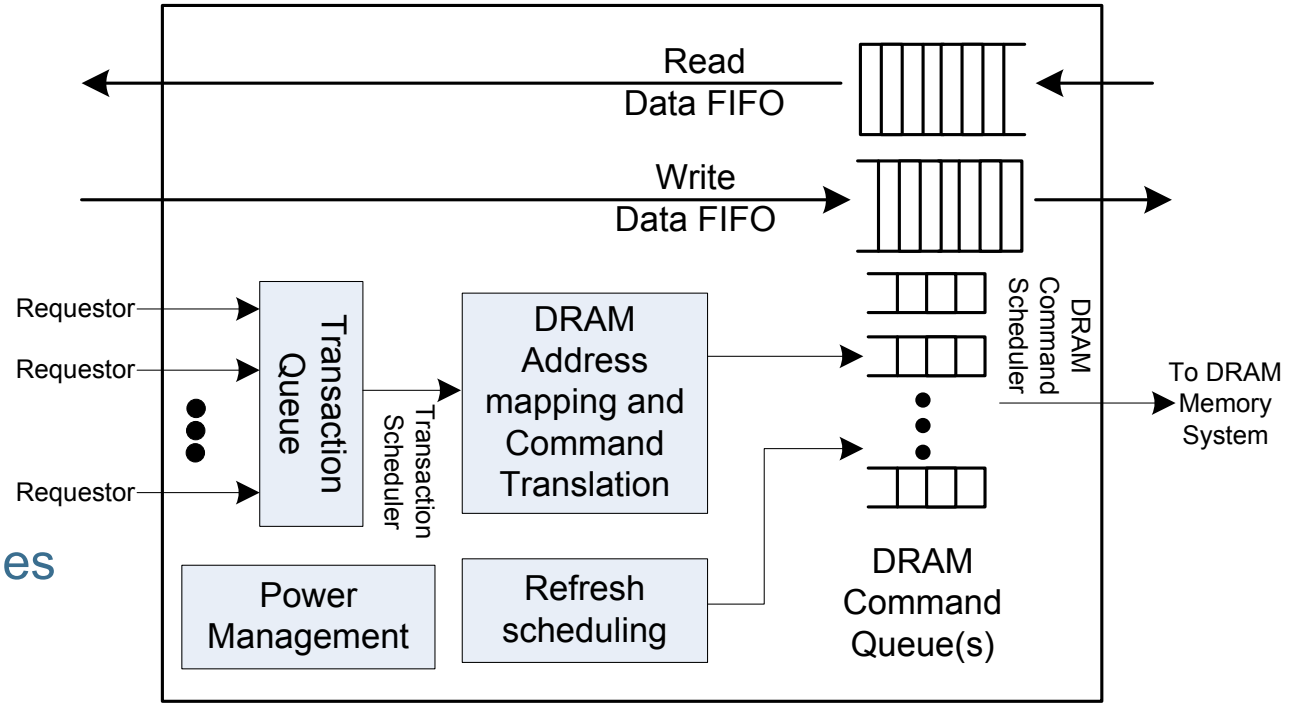
- Memory Controller is a widget that supports specific requests of the host(s) and accounts for the constraints provided by the memory device(s)
- Provide functionally correct and reliable operations for both host(s) and memory devices
  - Then, optimize for performance and power

Host (CPU), Memory Controller and Memory



- To design a good memory controller, we need to understand
  - Host(s) requirements
  - Host(s) to memory controller interface
  - Memory device peculiarities (constraints)
  - Controller to memory devices interface

Basic Memory Controller



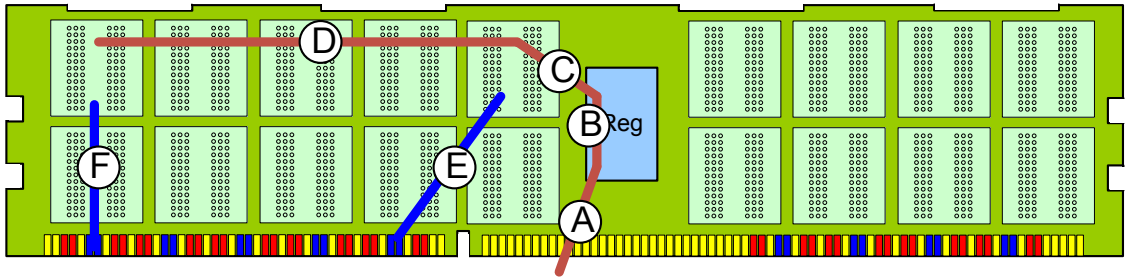
- Command Queues
- Data FIFO's
- PHY
  - Fine grained AC timing control
  - Trained to capture data accurately for read, launch data for write
  - Adjust for temperature and voltage variance
- Memory Controller Output
  - Properly Scheduled DRAM Command Sequences
  - Meet all {Activate, Precharge, Refresh, read/write/ODT turnaround time} requirements

Initialization, Training and DRAM PHY

PHY

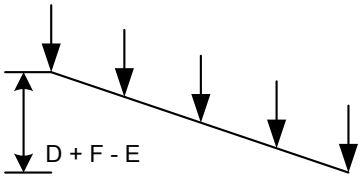
- Interfaces the parallel, synchronous, digital part of memory controller with the skewed, mesosynchronous, analog-ish memory system
- Drive signals at the correct voltage and timing
- Capture signals at the correct voltage and timing
- Fine grained timing adjustments
  - Skew, jitter, noise
- Voltage level shifters
  - Noise rejection, compatibility enablement (e.g. different voltages for different memory technologies)

RDIMM Timing and System Training

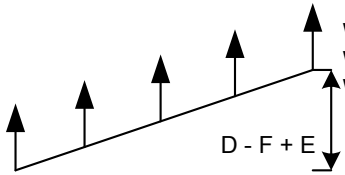


Differential DQS
DQ
Gnd/unlabeled signals

A. Connector Edge to Register Input Clock Delay
B. Register t_{PDM} + ½ tCK
C. Clock delay from register to nearest DRAM device
D. Clock delivery difference between nearest and farthest DRAM device
E. Data trace length from nearest DRAM device to connector edge
F. Data trace length from farthest DRAM device to connector edge



Read Data Timing at DQS8/17: A + B + C + E
Read Data Timing at DQS0/9: A + B + C + D + F
Read Data Timing difference between DQS0/9 and DQS8/17 = D + F - E



Write Data Timing at DQS8/17: A + B + C - E
Write Data Timing at DQS0/9: A + B + C + D - F
Write Data Timing difference between DQS0/9 and DQS8/17 = D + E - F

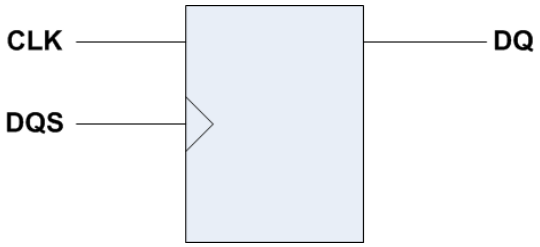
- The timing of when a DRAM device starts to drive DQ and assert ODT depends on when it “sees” the arrival of clock at chip interface
- Read training and write leveling needed to accommodate differences: Register t_{PDM}, DIMM Topology, DRAM t_{DQSCK}, system board trace lengths

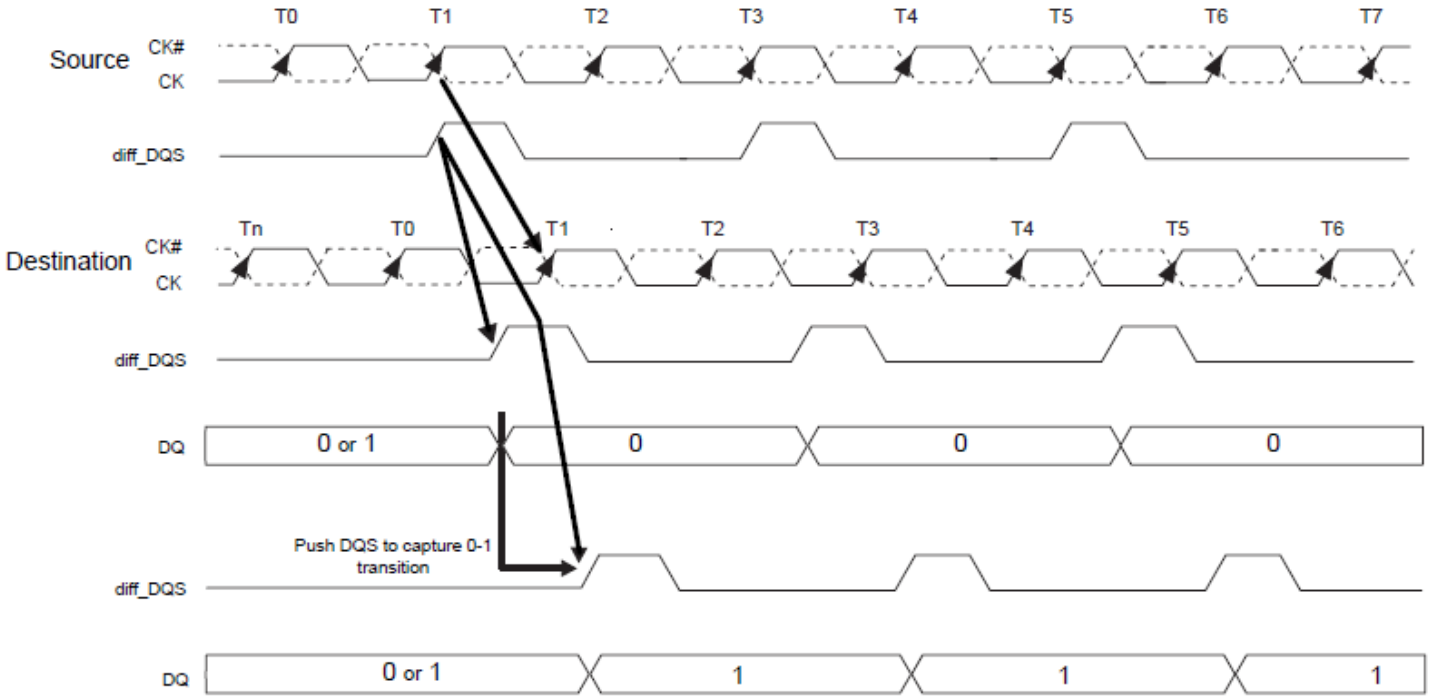
Training

- Clock/Address training
  - Aligns command and address with clock at receiver
- Read Receiver Enable Training
  - When to enable DQS receiver, per DIMM, per channel
  - Determine effective width of read preamble
- Read DQS Training
  - Place DQS strobe at center of DQ data eye
- Write Levelization
  - When to launch “write data wave” to DRAM devices
- Write DQS Training
  - Align DQ to DQS for each DRAM device

Write Leveling

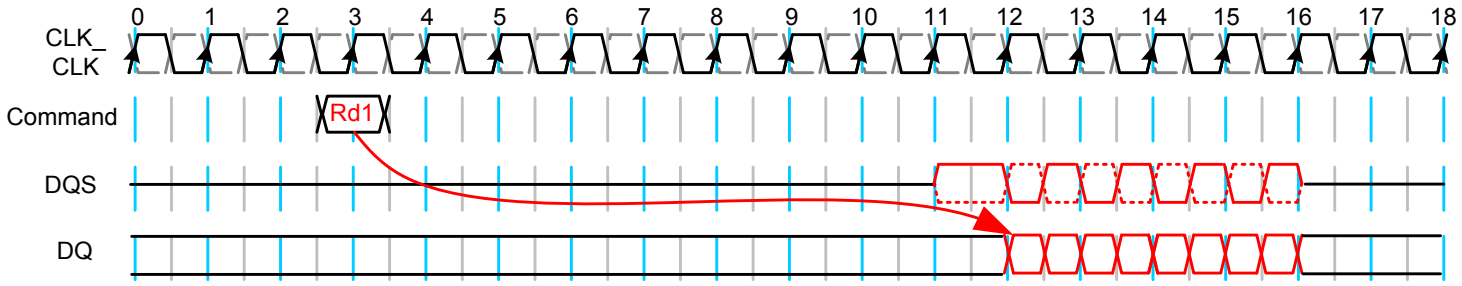
- Write to DRAM MRS to set leveling mode
- DRAM enables special flip-flop
- Host delays DQS pulses until DQ changes to discover clock positions



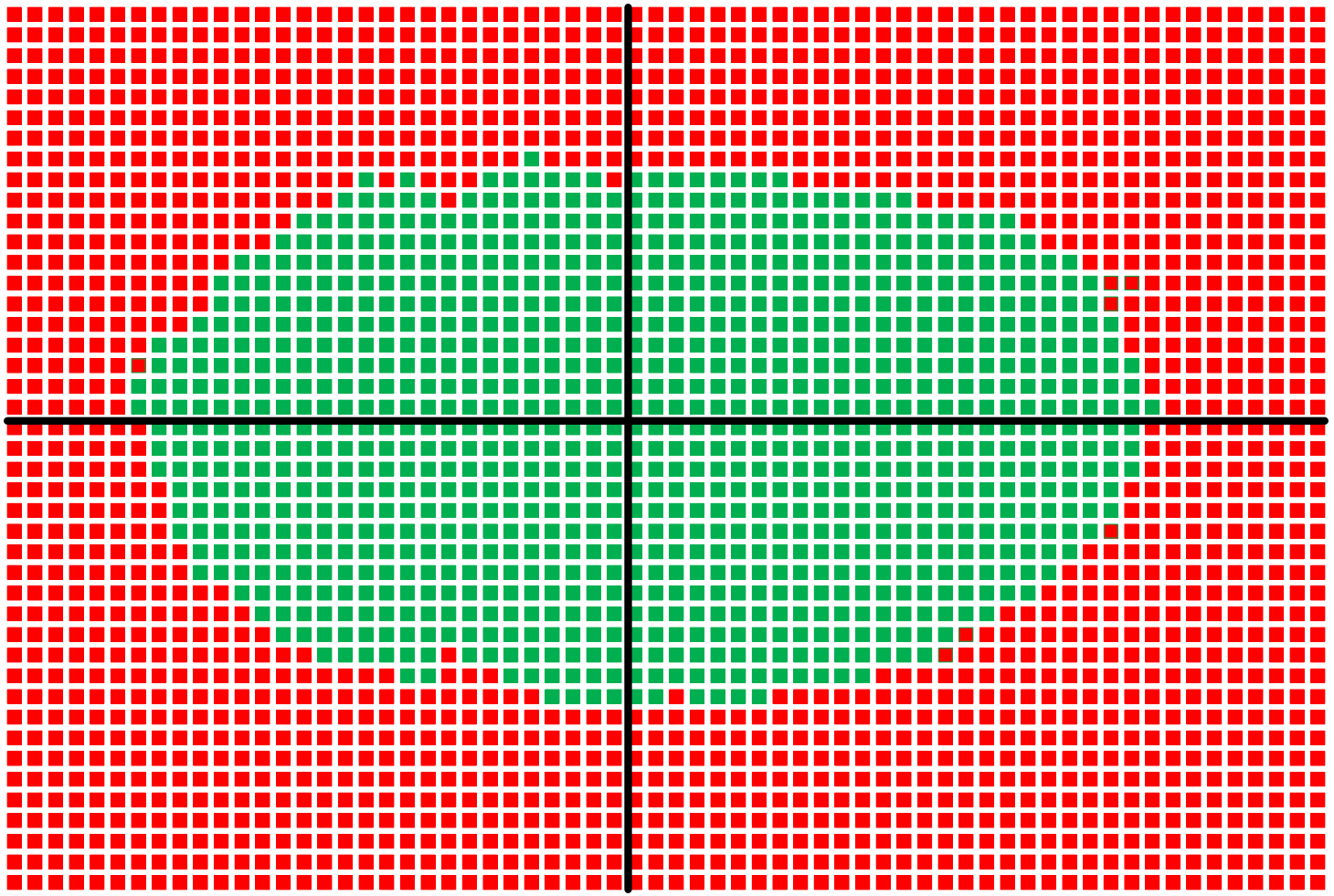


Read {DQS, Receiver Enable} Training

- DQS/DQ arrival requirements need to be discovered by host memory controller
- Host performs read cycles with DRAM MPR enabled
  - DRAM data pattern is fixed
- Host has to determine when DRAM asserts DQS pre-amble
- Also determine DQS edge position
- Per {bit, nibble, byte} lane, per rank



Schmoo Algorithms to Discover Center of Data Eye



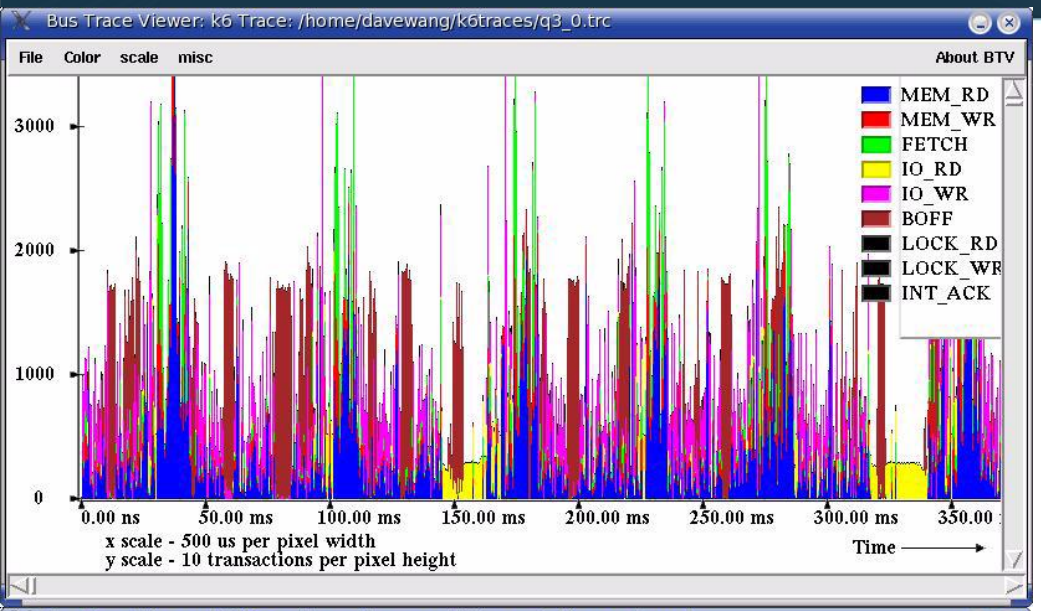
- Shifting vref and timing to find the center of the data eye in terms of timing and voltage

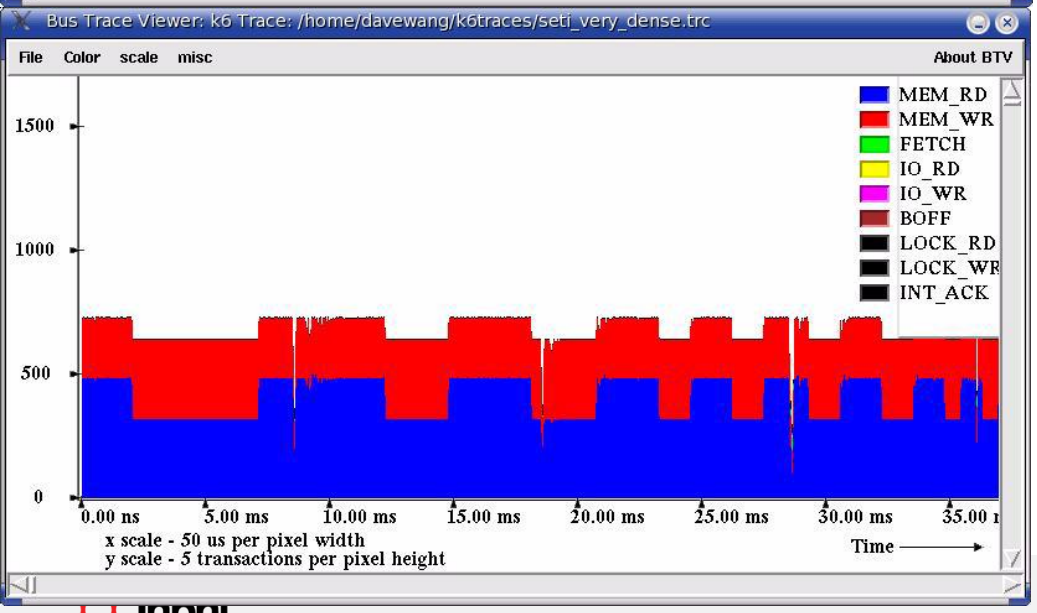
Host (CPU) Requirements – Transaction Interface

What Does the Host Look Like?

- What kind of host(s) will the memory controller support?
  - Server/Enterprise Processors
  - Mobile Processors
  - GPU/GPGPU
  - FPGA
  - Multiple hosts
- Workload Characteristics?
  - Read/Write Ratio
  - Request Rate (Bandwidth)
  - Locality characteristics
    - Spatial (How “close” are the addresses of the requests?)
    - Temporal (How close are the requests to each other in terms of time?)
- Number of Concurrent Contexts?
- Cache Sizes?

K6 Processor to Memory Controller Bus Traces

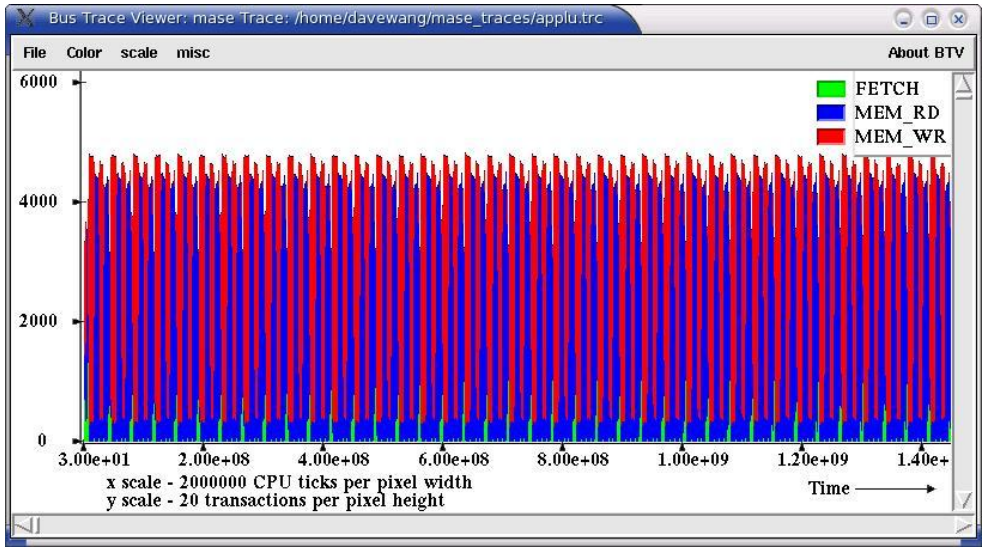




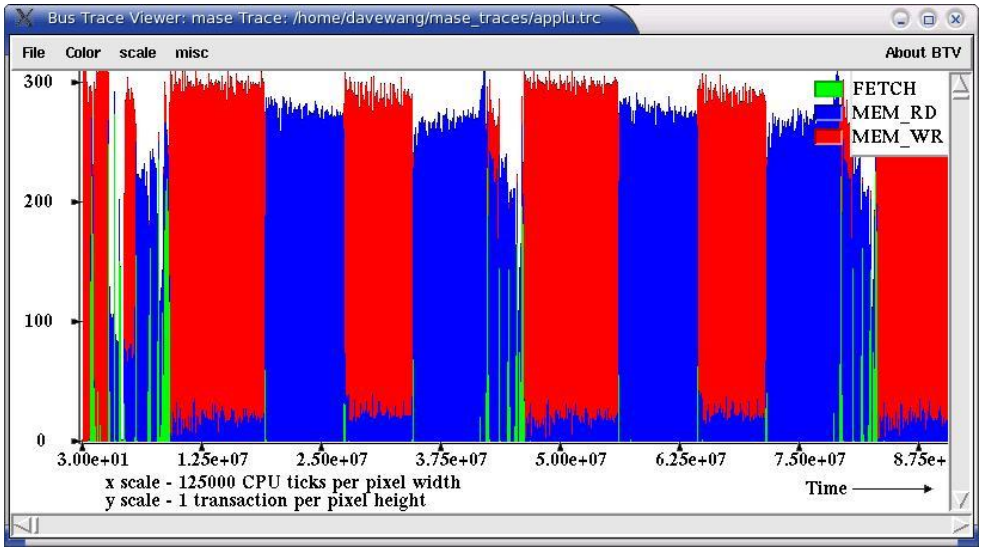
- Quake trace shows processing of each frame
  - 50~75 ms per frame
  - Or, 20~30 FPS

- Stream trace shows memory array access
  - Periods of 1:1 R/W ratio
  - Periods of 2:1 R/W ratio
  - Every 10 ms, array access is interrupted by system timer (context switch to O/S scheduler)

Simplescalar Traces (SPEC CPU – Applu)



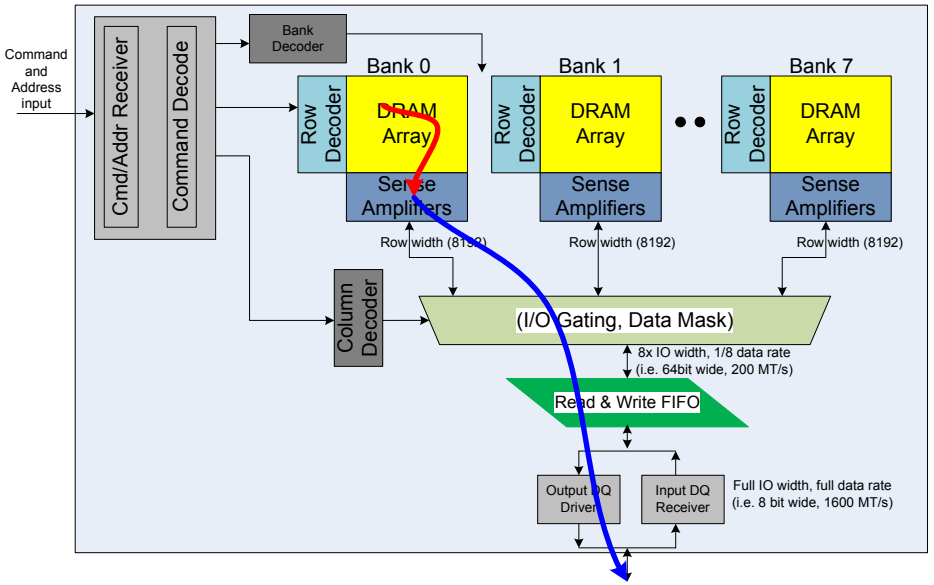
- Shows Memory Read, Memory Write, Memory Fetch Requests from Simplescalar Simulator



- Zoomed in to show periods of purely read access pattern followed by write dominant access pattern

Address Mapping

Paging Policy



- DRAM Access consist of two basic steps
  - Row Access (Bank Activate)
  - Column Access (read or write)
- Takes ~15 ns to open a row (destructive read), ~15 ns to restore data to cells, ~15 ns to precharge bitline
- Once you open a row, do you keep it open for subsequent access or do you precharge immediately?

Paging Policy – Figuring it out

Open page – page hit latency = t_{CL}
Open page – page miss (bank conflict) latency = $t_{CL} + t_{RP} + t_{RCD}$
Close page – read latency = $t_{RCD} + t_{CL}$

Let A = the percentage of open page hits and
(1 - A) = percentage of page misses (bank conflict)

For open page to be better than close page

$$A * t_{CL} + (1 - A) * t_{CL} + t_{RP} + t_{RCD} < t_{RCD} + t_{CL}$$

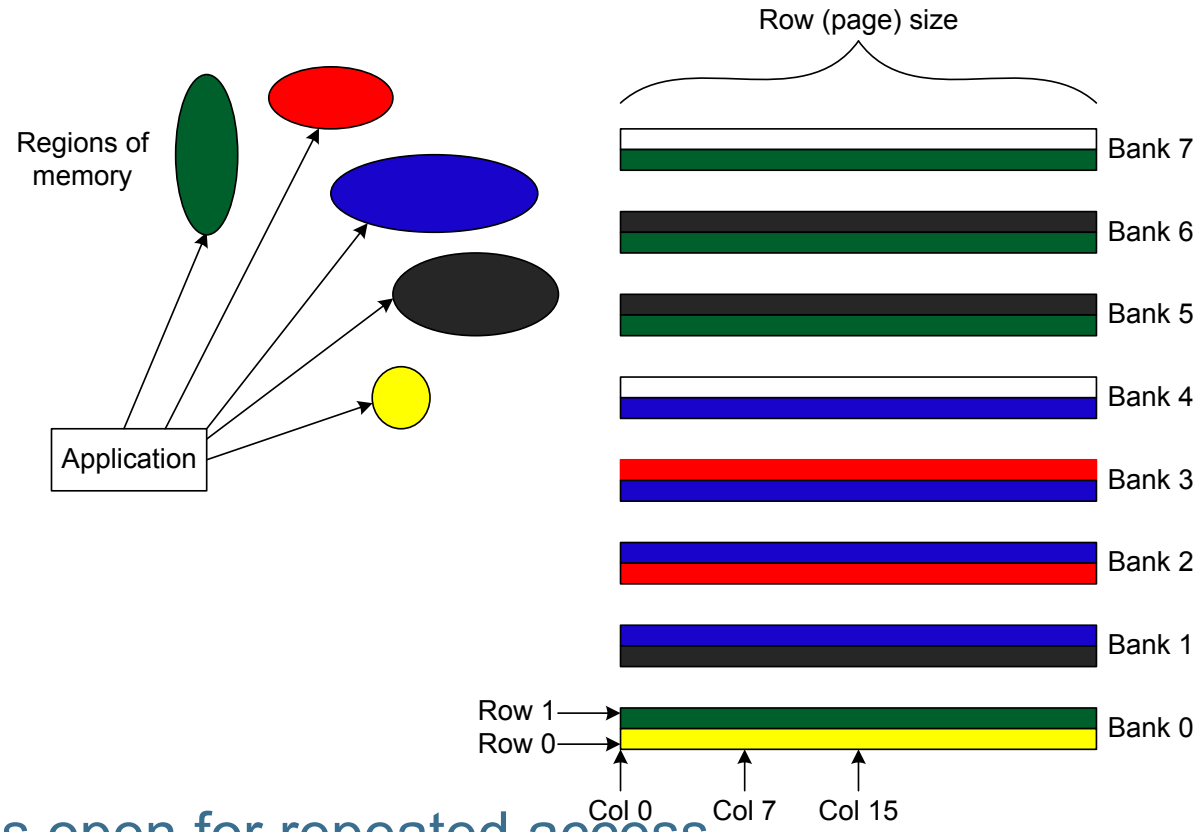
$$A * t_{CL} + t_{CL} + t_{RP} + t_{RCD} - A * t_{CL} - A * t_{RP} - A * t_{RCD} < t_{RCD} + t_{CL}$$

$$t_{RP} < A * t_{RP} - A * t_{RCD}$$

$$t_{RP} / (t_{RP} + t_{RCD}) < A$$

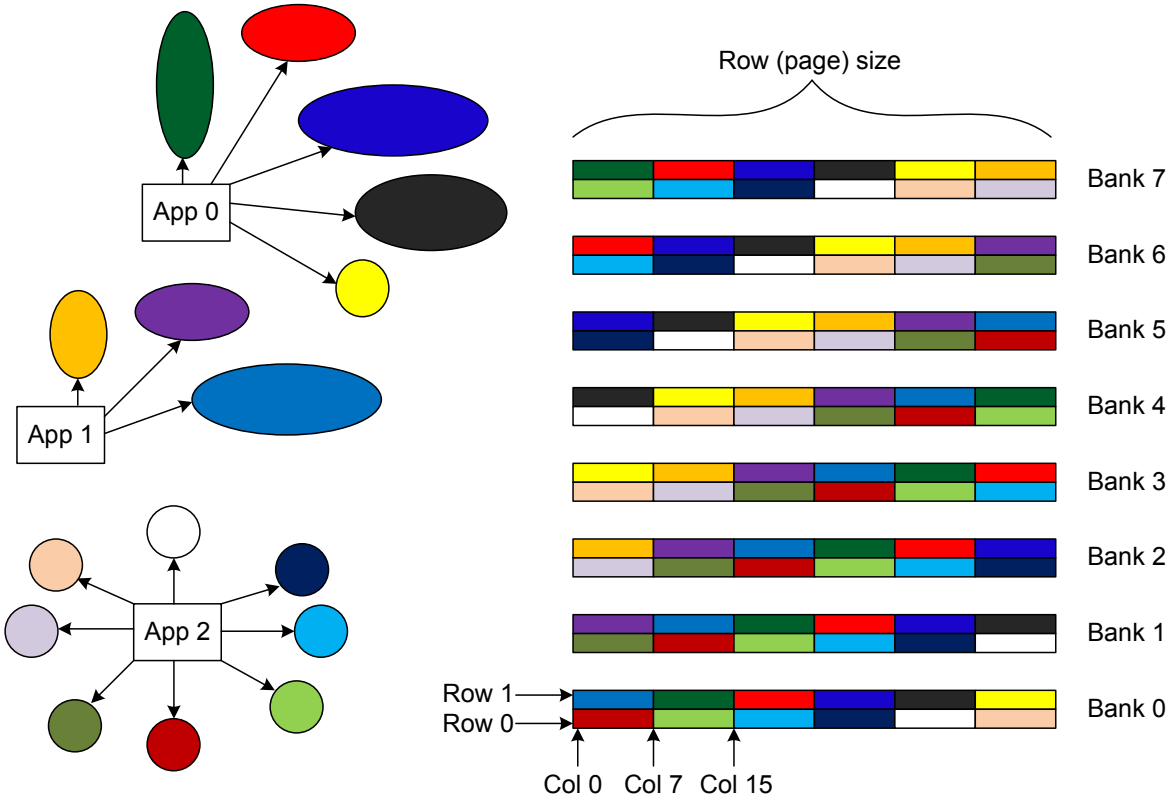
- For DRAM devices where $t_{RP} = t_{RCD}$, open page hit percentage has to be greater than 50% for open page to be better than close page

Open Page for Single Threaded Apps



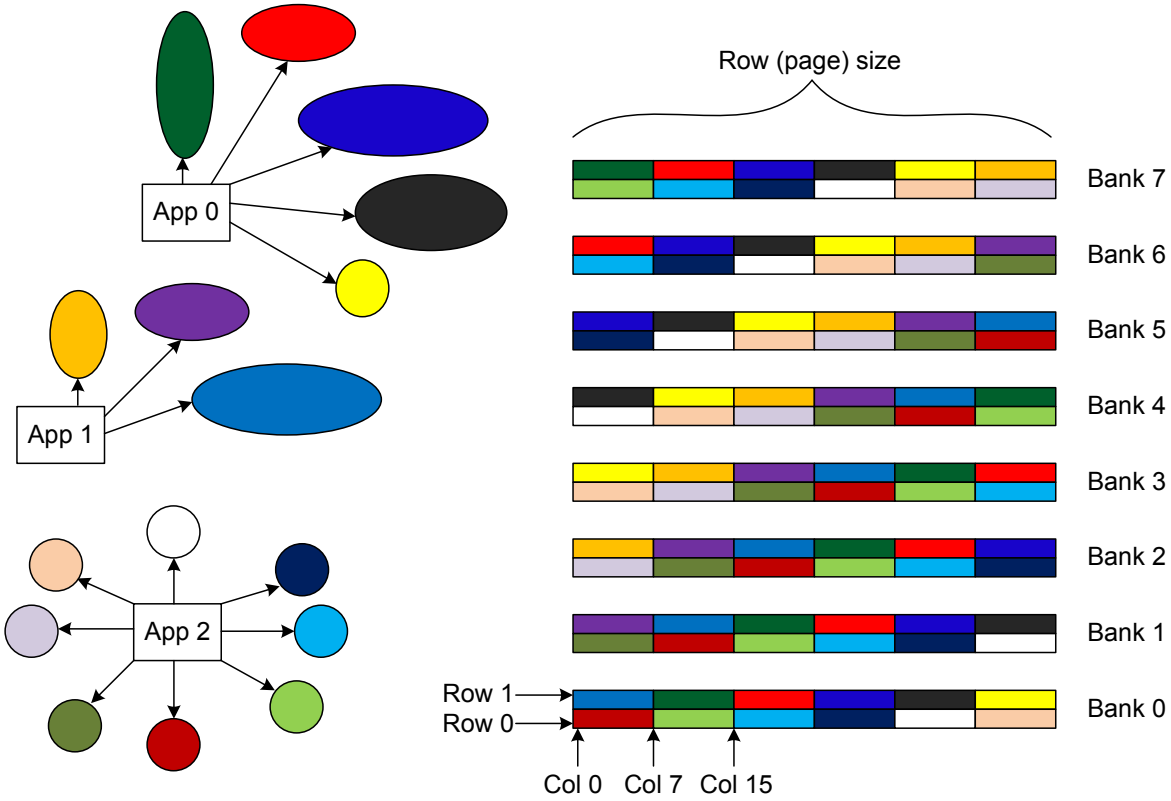
- Keep pages open for repeated access
- Ideal for single threaded applications
- Take advantage of inherent locality
- One row access, multiple column accesses

Closed Page for Multi-Threads



- One (or few) column access per row access
- Open page for single access, then close immediately
- Better for (highly) multi-threaded applications

Side Effects of Large Caches



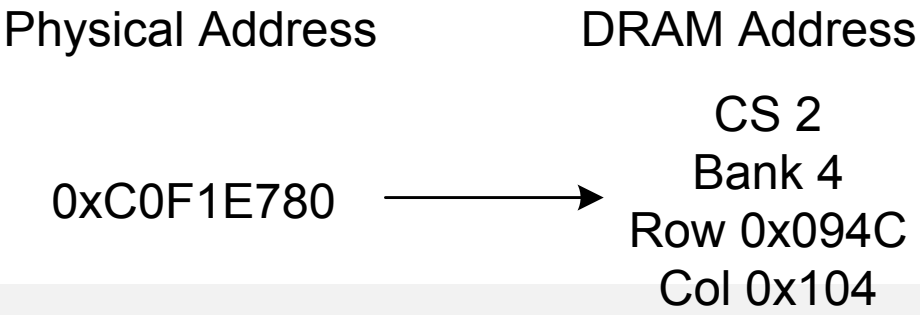
- Modern processors have large caches
- Writes are often cached for performance reasons
- Dirty write evictions have poor temporal locality with DRAM pages open for read

Paging Policy

- Low core count (Desktop/Mobile) memory controller
  - Open page is good
- High core count Server memory controller
  - Close (or two accesses per page, then close) page is better
  - Increasing core count and cache sizes decrease locality seen at memory controller interface
- Comes down to this . . .
  - How many “regions” of memory will be accessed simultaneously?
  - How many DRAM banks do you have in the system?
  - If “region count” >> DRAM banks, then close page is better
  - If “region count” << DRAM banks, then open page is better

Address Mapping

- Why spend so much time on Paging Policy?
  - Paging policy dictates how banks indices should be allocated
  - Expandability versus parallelism influence rank indices placement
  - May wish to permute bank and rank indices to avoid collisions of stride accesses
- Optimize to take advantage of DRAM memory system structure(s)
  - Exploit parallelism
  - Avoid structural hazards
- Quite a bit to think about in the simple Physical to DRAM address mapping



Address Mapping
Covered in detail in Ch13 of ***Cache, DRAM, Disk***

Per channel, per-rank address mapping scheme for single/asymmetric channel mode

rank capacity (MB)	rank configuration row count x bank count x column count x column size	physical address 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
128	8192 x 4 x 512 x 8	10 9 8 7 6 5 4 3 2 1 0 11 12 | 0 1 | 8 7 6 5 4 3 2 1 0 | x x x
256	8192 x 4 x 1024 x 8	12 10 9 8 7 6 5 4 3 2 1 0 11 | 1 0 | 9 8 7 6 5 4 3 2 1 0 | x x x
512	16384 x 4 x 1024 x 8	13 12 10 9 8 7 6 5 4 3 2 1 0 11 | 1 0 | 9 8 7 6 5 4 3 2 1 0 | x x x
512	8192 x 8 x 1024 x 8	12 11 10 9 8 7 6 5 4 3 2 1 0 | 0 1 2 | 9 8 7 6 5 4 3 2 1 0 | x x x
1024	16384 x 8 x 1024 x 8	13 12 11 10 9 8 7 6 5 4 3 2 1 0 | 0 1 2 | 9 8 7 6 5 4 3 2 1 0 | x x x

single/asymmetric channel

row ID

bank ID

col ID

byte offset per DIMM

symmetric dual channel

chan ID

128	8192 x 4 x 512 x 8	10 9 8 7 6 5 4 3 2 1 0 11 12 | 0 1 | 8 7 6 5 4 3 | 0 | 2 1 0 | x x x
256	8192 x 4 x 1024 x 8	12 10 9 8 7 6 5 4 3 2 1 0 11 | 1 0 | 9 8 7 6 5 4 3 | 0 | 2 1 0 | x x x
512	16384 x 4 x 1024 x 8	13 12 10 9 8 7 6 5 4 3 2 1 0 11 | 1 0 | 9 8 7 6 5 4 3 | 0 | 2 1 0 | x x x
512	8192 x 8 x 1024 x 8	12 11 10 9 8 7 6 5 4 3 2 1 0 | 0 1 2 | 9 8 7 6 5 4 3 | 0 | 2 1 0 | x x x
1024	16384 x 8 x 1024 x 8	13 11 12 10 9 8 7 6 5 4 3 2 1 0 | 0 1 2 | 9 8 7 6 5 4 3 | 0 | 2 1 0 | x x x
rank capacity (MB)	rank configuration row count x bank count x column count x column size	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 physical address

Per-rank address mapping scheme for dual channel symmetric mode

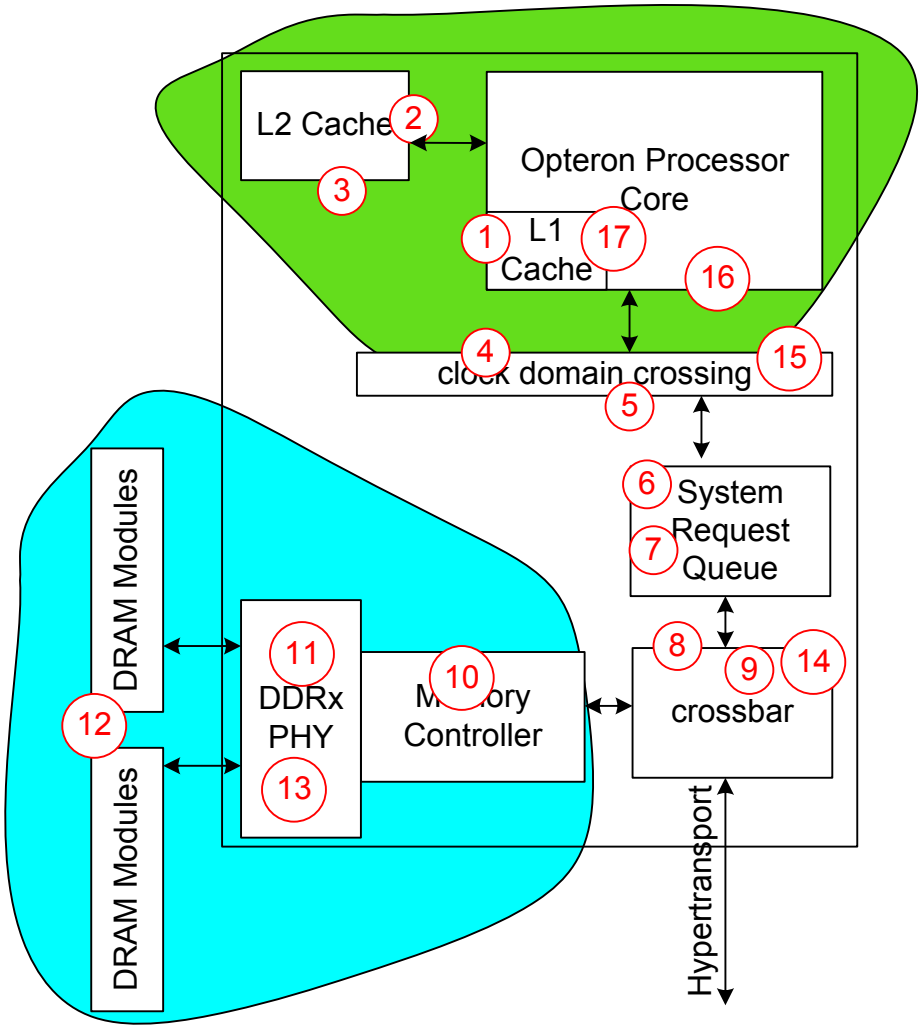
FIGURE 13.4: Per-rank address mapping schemes in the 82955X MCH.

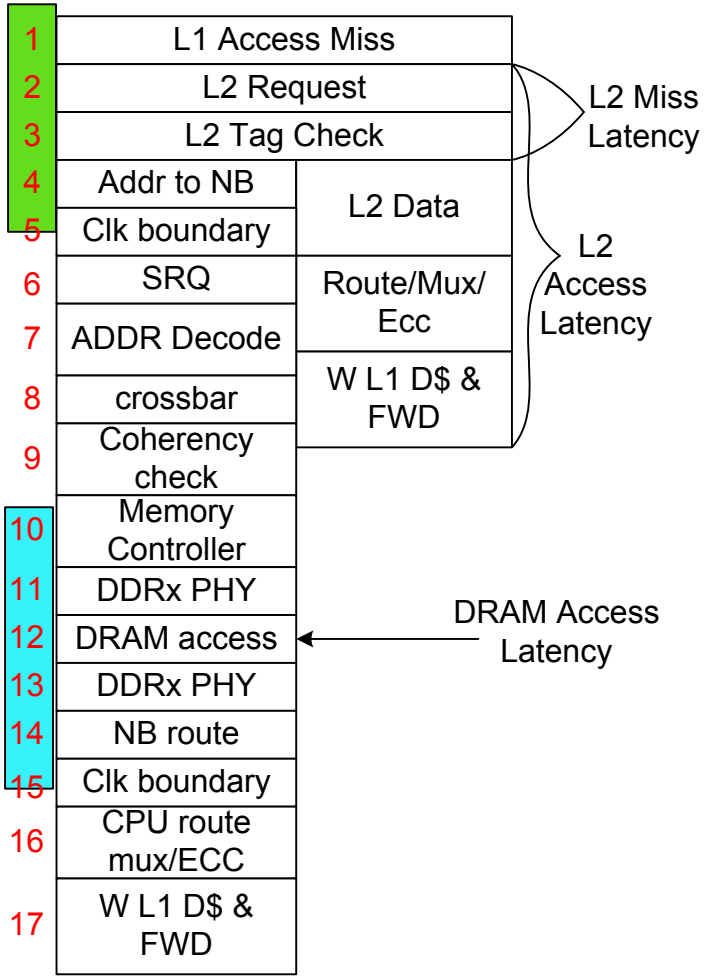
Operating Frequency and Latency

Gearbox Logic

- Host CPU's typically operate at high frequencies
  - E.g. 3+ GHz
- DRAM operates at somewhat lower base frequencies
  - E.g. (1067 MHz) 2133 MT/s
- Gearbox logic interfaces with multiple clock domains, moving command/data between them
- Lowest memory access latency or optimal bandwidth depends on ratio of CPU frequency to DRAM operating frequency
- If designing to support multi-frequency Host/DRAM, need to find harmonic frequencies or design gear box
- Need to stage logic carefully to minimize latency
- One alternative – use low base frequency and group schedule multiple DRAM command per "cycle"
  - E.g. FBDIMM

Frequency Dependence of Memory Access Latency





High Datarate and Low Latency

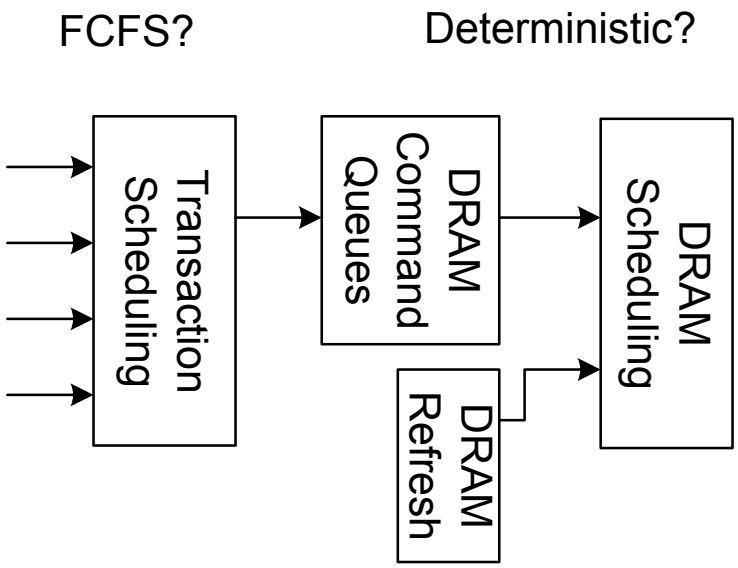
Configuration		data				units
Memory	DIMM Type	RDIMM	LRDIMM			
	Datarate	800	800	1066	1333	MT/s
Sandra	Latency	92	96	90	85	ns
	Assignment (Int BW)	6	6	7.8	9.7	GB/s
	Scaling (Int BW)	6	5.8	7.8	9.7	GB/s
	Addition (Int BW)	5.9	5.9	7.8	9.7	GB/s
	Triad (Int BW)	6	5.8	7.8	9.6	GB/s

- Operating at highest datarate of given controller typically means lowest latency
  - Latency through controller in number of cycles
- LRDIMM (added buffering) @ 1066 MT/s had lower latency than RDIMM @ 800 MT/s

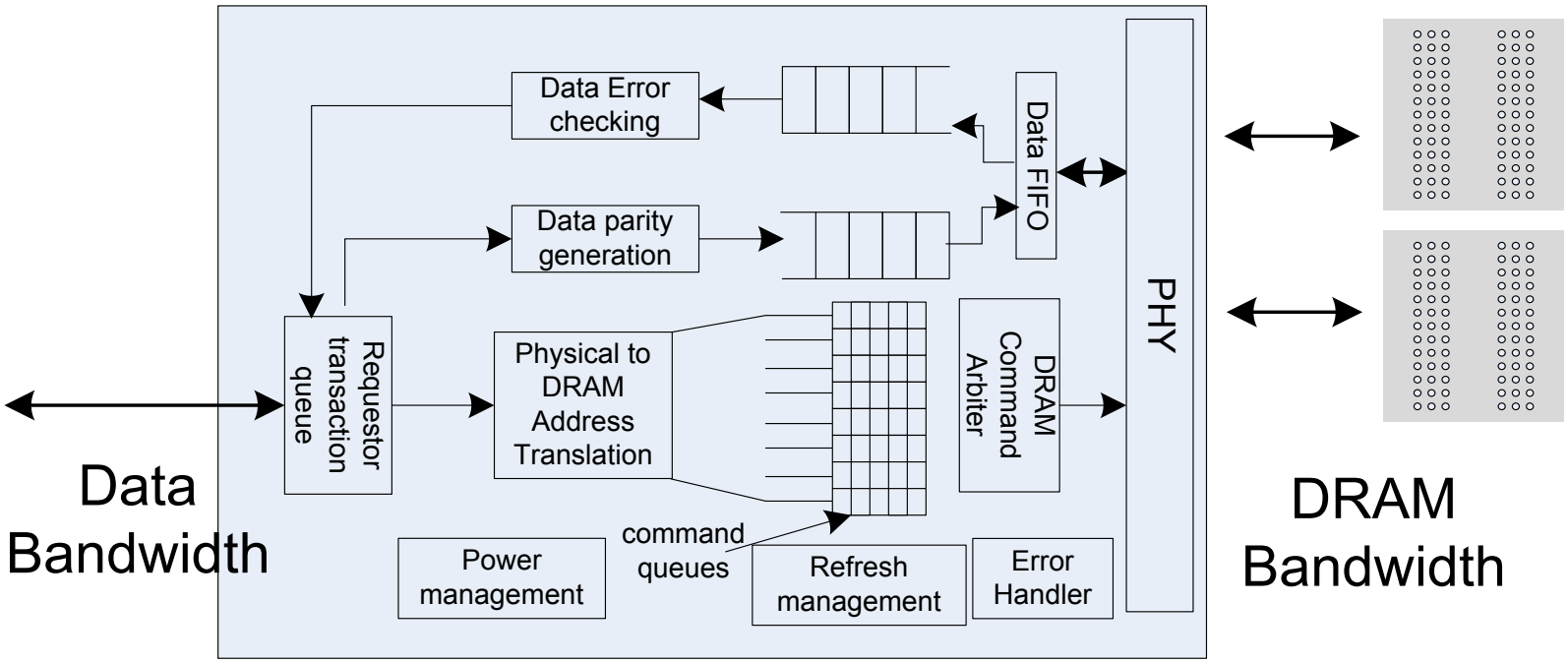
DRAM Command and Transaction Scheduling

Transaction and DRAM Command Scheduling I

- No single, optimal solution for all applications can satisfy conflicting requirements
- Considerations
  - Process (thread) Priority
  - Transaction Type
  - Fairness
    - Guarantee against starvation
  - DRAM Structure
- Example
  A. High Priority Thread Read
  B. Low Priority Thread Instruction Fetch
  C. I/O Read
- What if
  - Scheduling “A” causes bank conflict, while “B” does not?
  - I/O read has been bypassed for 1 us



Transaction and DRAM Command Scheduling II



- Different controllers make different trade offs
- Optimize for the resource that is more constrained
- If (DRAM Bandwidth > Data bandwidth), focus on Transaction scheduling (simple DRAM command scheduler)
- If (Transaction Bandwidth > Data Bandwidth), focus on DRAM command scheduling (keep DRAM wires busy)

DRAM Command Scheduling - Constraints

- DRAM Timing Parameter Constraints
  - t_{CAS}, t_{CWL}, t_{RCD}, t_{CCD}, t_{RP}, t_{WR}, t_{WTR}, t_{RTP}, etc.
- DRAM Power Constraints
  - t_{RRD}, t_{FAW}, etc.
- Bus Turnaround Times (Shared Resource Constraints)
  - "rank-to-rank switching time"

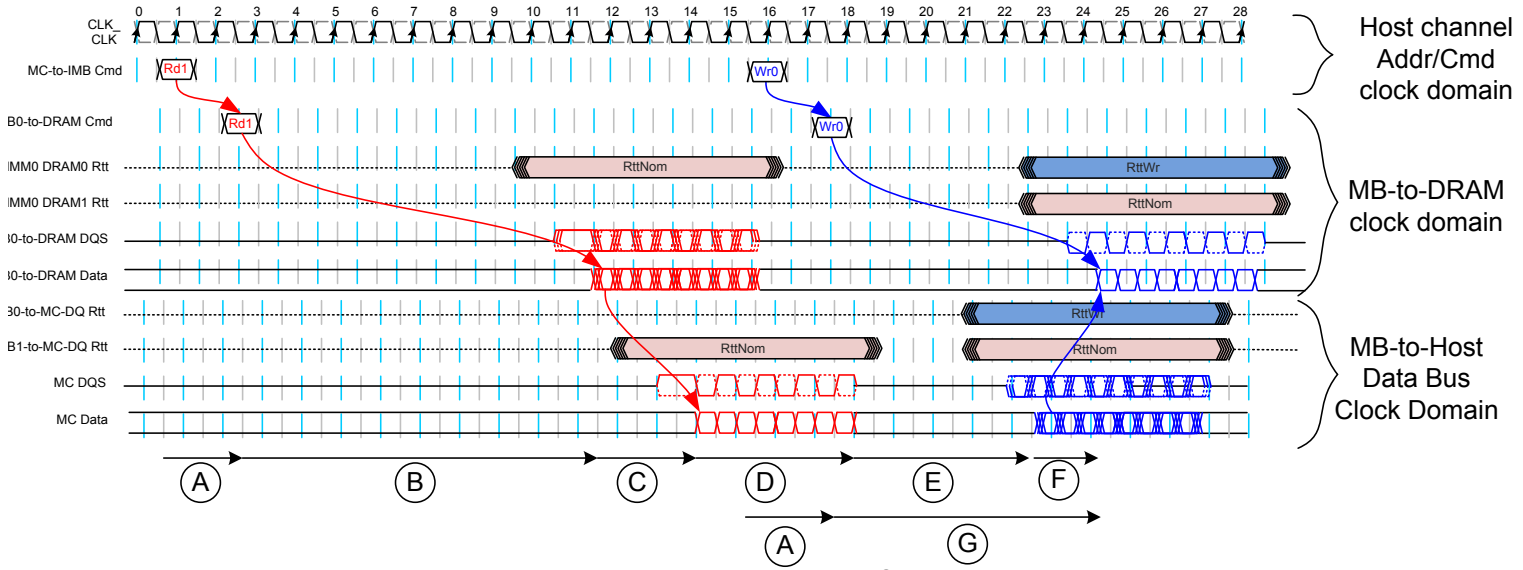
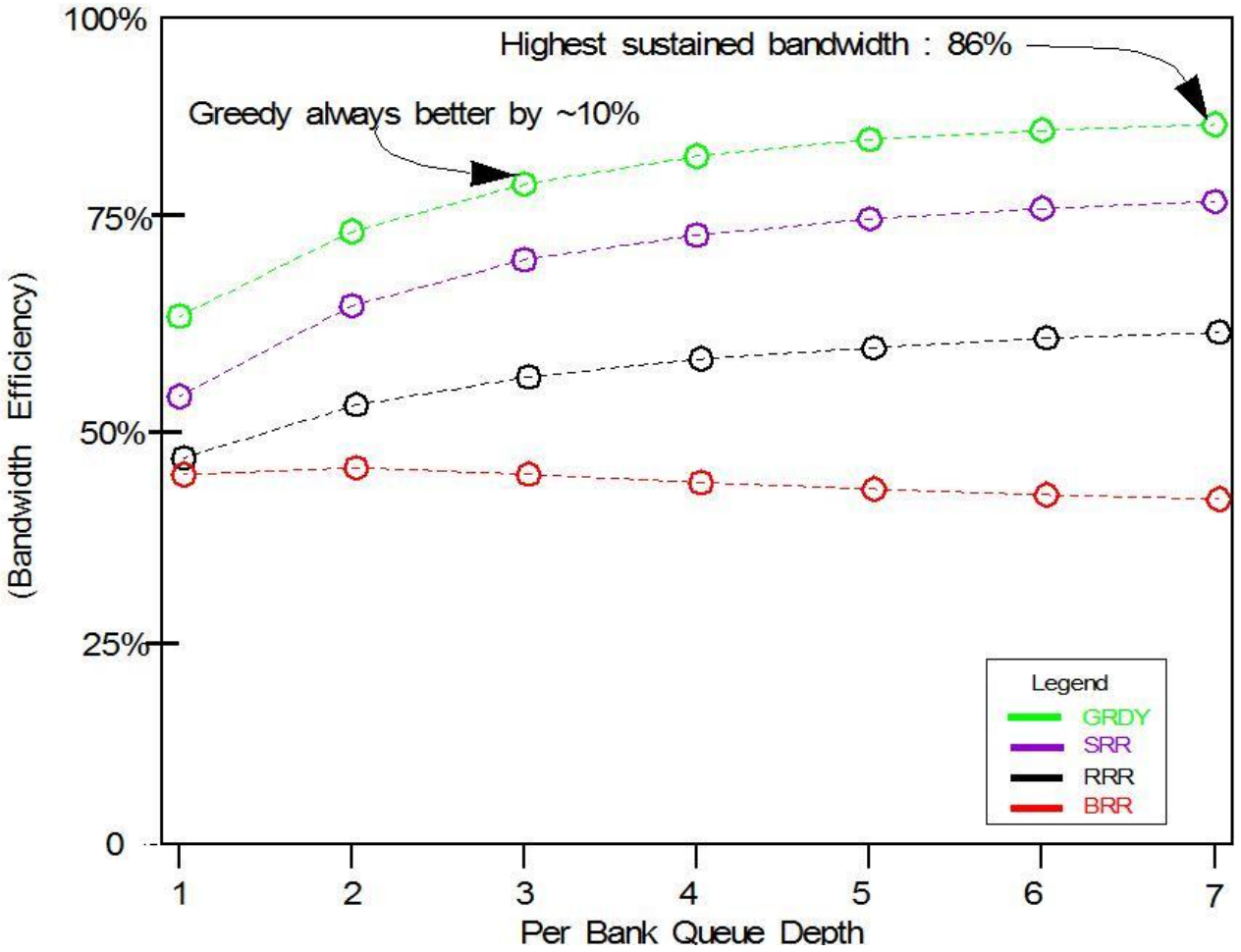


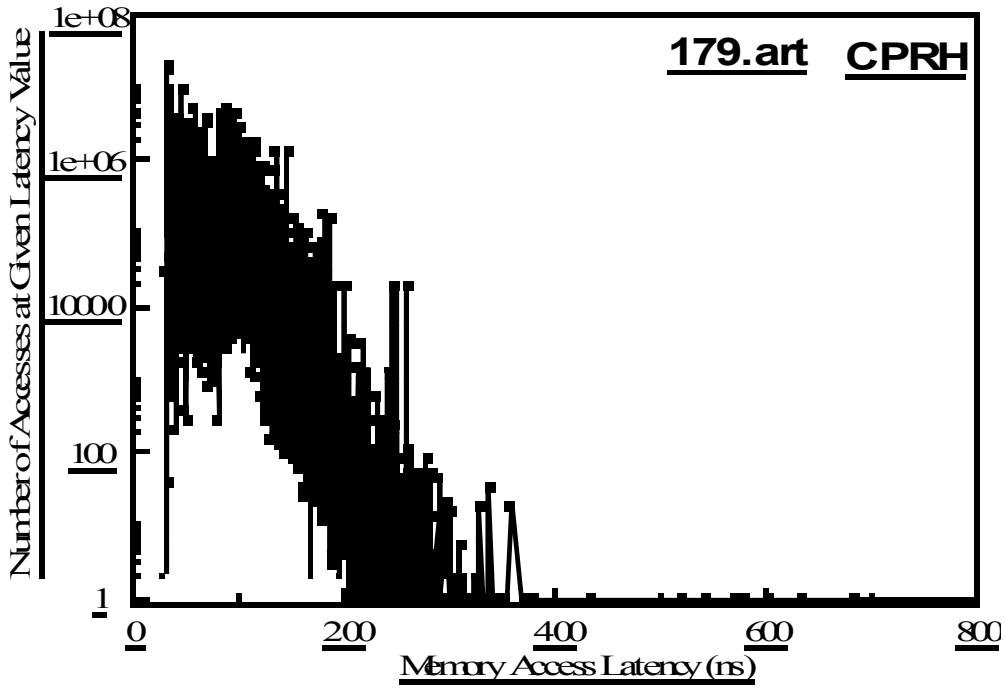
Diagram illustrates 1333 MT/s operation with CL9 DRAM devices

DRAM Command Queue Structure and Scheduling



- Simulation of queue structure and scheduling algorithm

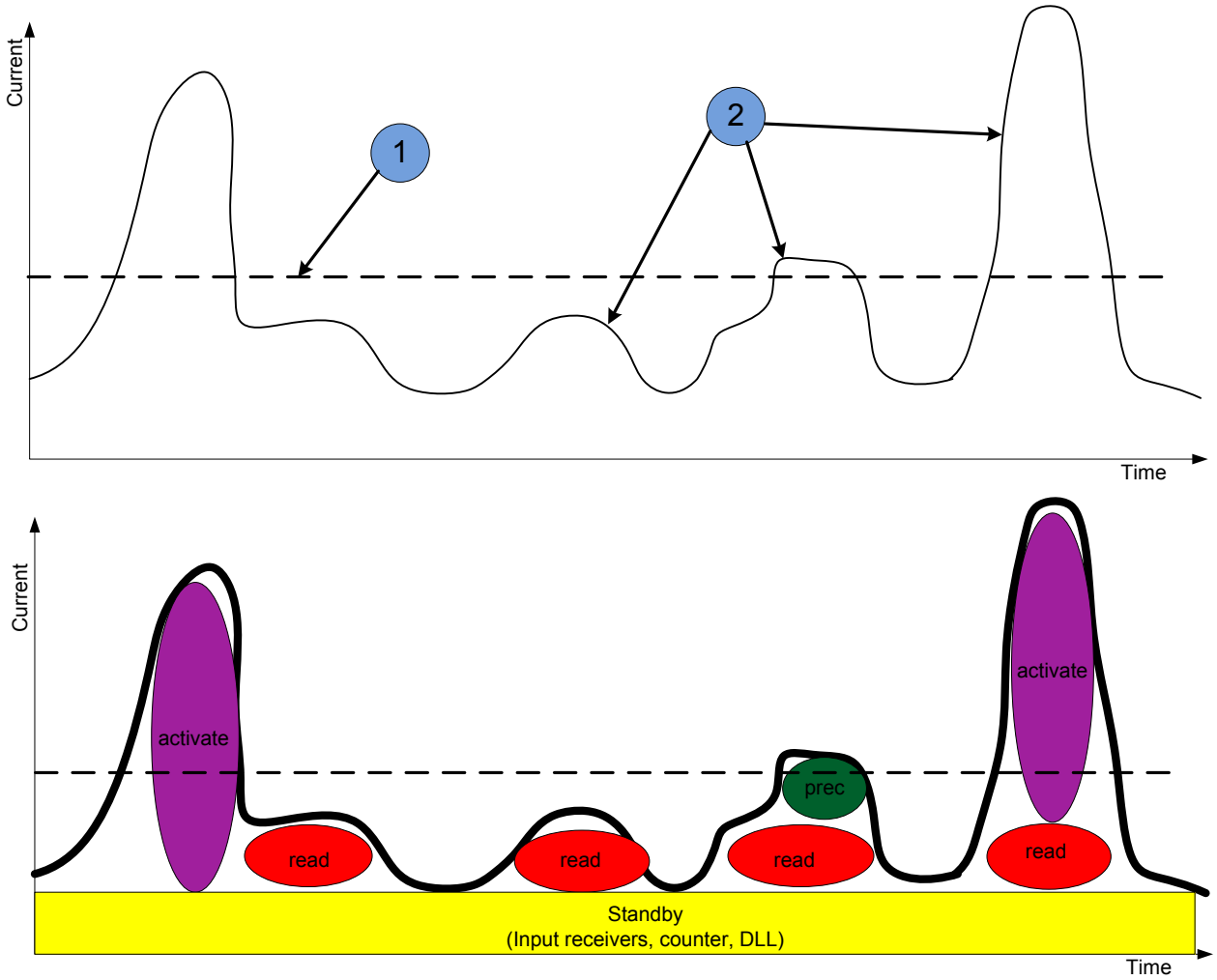
Queuing Delays



- Lot of queuing in memory system
- Queuing delays add significant to latency for high sustained bandwidth
- “better” memory system also depends on expected sustainable bandwidth utilization

DRAM Power Management
Relationship of Power, Latency and Bandwidth

Abstract Illustration of DRAM Power



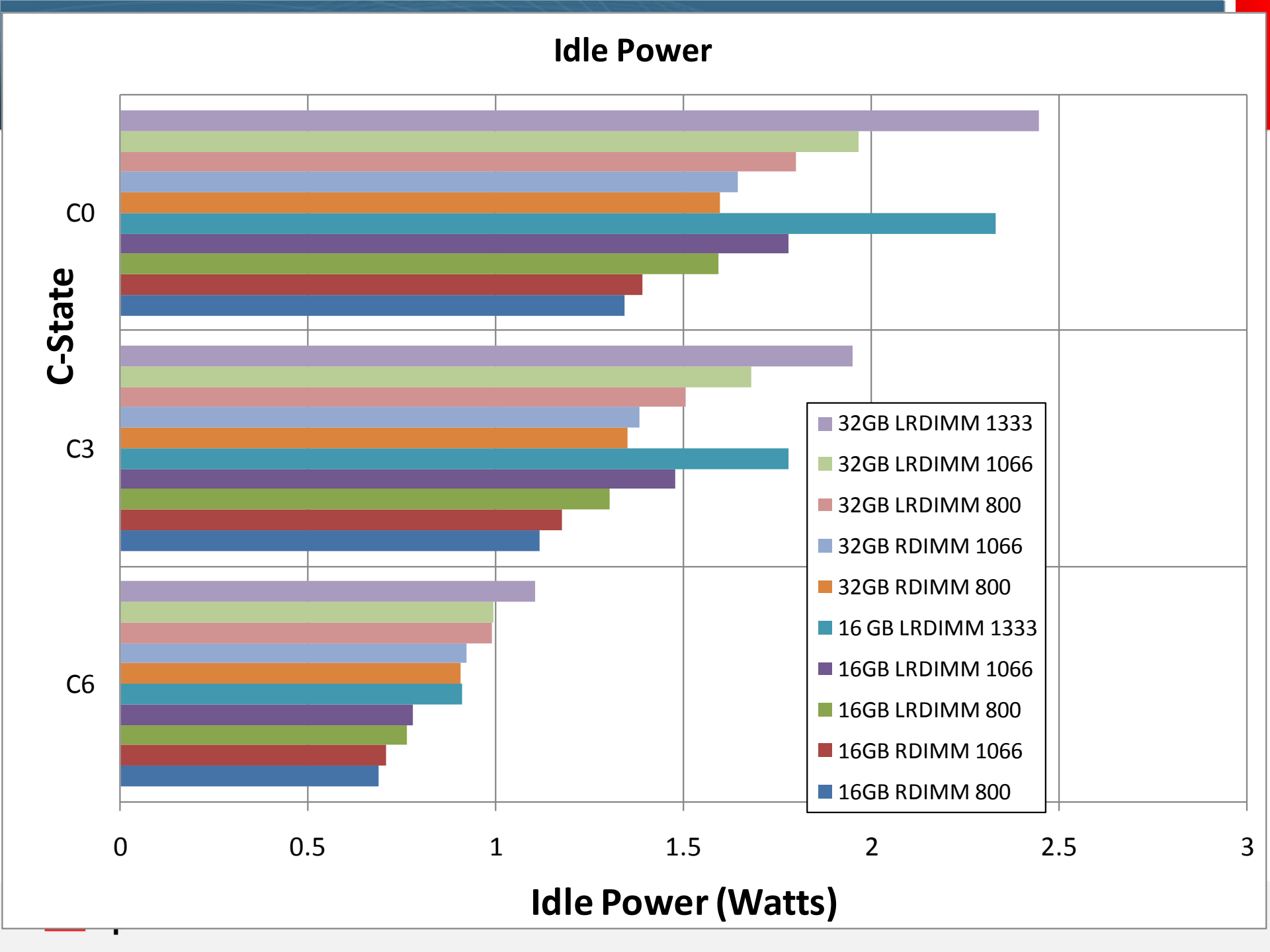
- Substantial portion of power is standby (DLL/receiver) power

DRAM Power

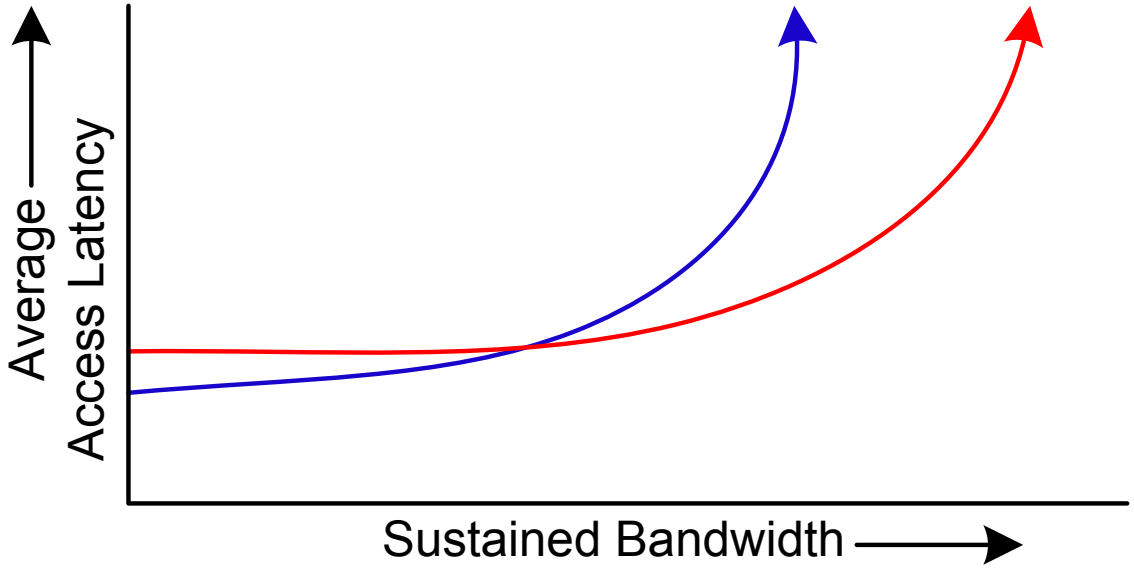
- In large (capacity) server memory systems, substantial amounts of power consumed by DRAM devices in standby
  - Active DLL, input receivers, FIFO
  - Algorithms designed to minimize standby power
- Mobile systems, substantial amount of power consumed by refresh activity, even during self-refresh
  - Partial array self-refresh

Reducing System Power with Race to Sleep

- Hurry up and sleep power management algorithm
- If application is neither latency critical nor bandwidth critical, queue up read/write requests, then perform burst read/write, interleaved with “idle” in self-refresh states
  - Need to understand system-level QoS (e.g. bounded latency) guarantees
- Example:
  - 20% bandwidth utilization ~= 1 command, turn into
  - 20 us in ~100% bandwidth, ~60 us in self refresh
- Eliminates
  - DLL/receiver power from DRAM during “idle” periods
  - Memory controller interface power (controller interface also powers down)
- Sounds good, but some systems and events have latency guarantees that must be met
  - Self-refresh exit times may be too long – need careful analysis and management


Idle Power
C-State
C0
C3
C6
32GB LRDIMM 1333
32GB LRDIMM 1066
32GB LRDIMM 800
32GB RDIMM 1066
32GB RDIMM 800
16 GB LRDIMM 1333
16GB LRDIMM 1066
16GB LRDIMM 800
16GB RDIMM 1066
16GB RDIMM 800
0
0.5
1
1.5
2
2.5
3
Idle Power (Watts)

Latency vs. Bandwidth vs. Power Curves



- Idle (unloaded) latency also depends on power state
- There are multiple curves, one for each power management scheme, for a given memory system
- Should have Latency vs. Bandwidth vs. Power “surfaces” as opposed to just curves

Reliability, Availability, Serviceability

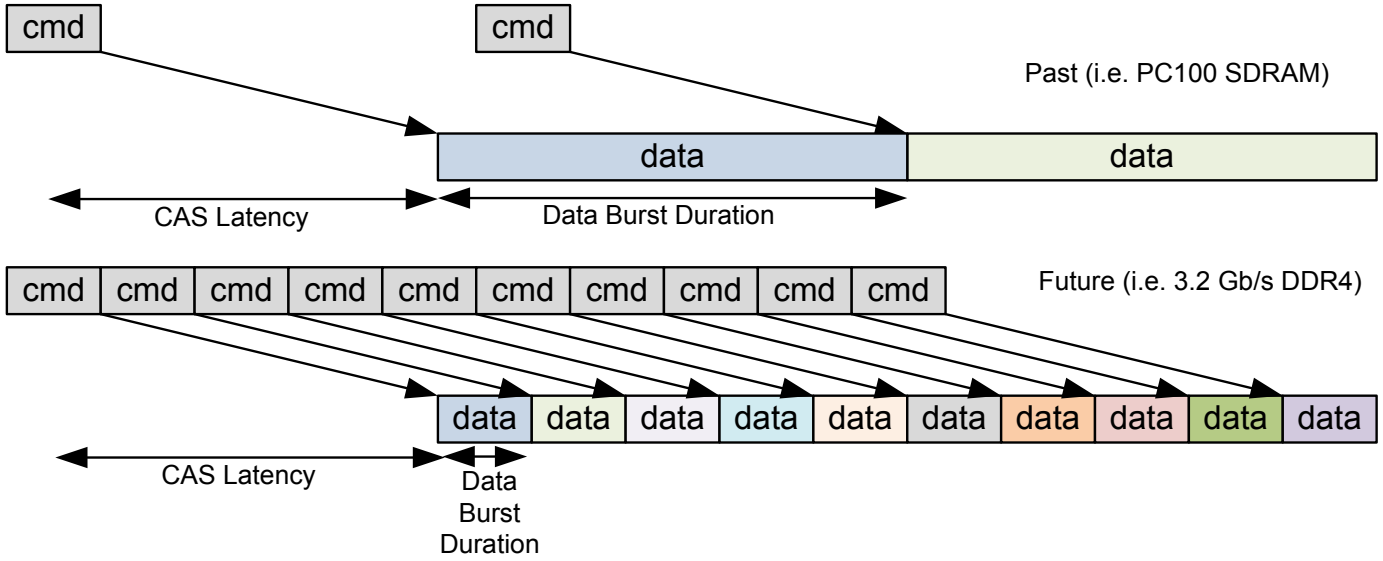
DRAM RAS

- Signaling Exception
  - Something went bad, how is it handled?
- Protecting Command Transport
  - Command and Address Parity
- Protecting Data Transport
  - CRC
- Protecting Data Storage
  - ECC
  - Chipkill
  - Scrubbing

Signaling Exception in Memory Systems

- Existing DRAM memory systems are non-intelligent entities
  - Host controller responsible for maintaining coherency, consistency and correctness
- No mechanism from DRAM to generate “back pressure”
  - CellularRAM is an exception
  - LPDDR2-NVM uses DNV signal
- Future memory systems such as HMC may use transaction based protocol
  - More efficient handling of errors and exceptions
  - Not available in present systems
  - May incur additional latency penalty
- Existing high end servers utilize exception handling for error recovery
  - Instant (nanosecond scale) response to “alert”
  - Command Parity Error and CRC Error
- Also, event_n signal maybe used to handle slower events
  - Slow (microsecond scale) to respond to event_n trigger
  - E.g. Thermal trigger (CLTT)

Exception Handling for High Speed Memory Systems



- High data rate controllers need deep pipelines to schedule commands
- Upon exception - need to stall long pipeline, clean up and restart
  - Large performance hit at high frequencies
- More common – keep track of “replay pipeline”, then restart from point of fault

Alert_n Signal Usage in DDR4

- Wired-Or signal from many devices
  - Which command or data faulted?
  - Multiple nanoseconds of “uncertainty window”
- Halt and restart from point of (address) fault
  - Controller to read error log via MPR mechanism
- Force sequential re-execution of DRAM command stream from point of address fault
- Alert_n may also signal DRAM write CRC error
  - Faulting data may also require replay of command sequence to avoid ordering violation

Address/Command Parity Error Log Register

2.18.28 RCC0 .. RCFF - Error Log Register

The control word locations C0 .. FF function as a 32-bit error log register. Upon occurrence of a CA parity error the device logs the following sampled command and address bits in the Error Log Register, which can be transferred by the memory controller to page 0 of the DRAM MPR, where it can be read by the host.

Table 52 — RCC0 .. RCFF: Error Log Register

Control Word	Setting (DA[7:0])							
RCCx	A7	A6	A5	A4	A3	A2	A1	A0
RCDx	A15/ CAS_n	A14/ WE_n	A13	A12	A11	A10	A9	A8
RCEx	DPAR	ACT_n	BG1	BG0	BA1	BA0	A17	A16/ RAS_n
RCFx	> 1 Error[1,2]	CA Parity Error Status[1,3]	Reserved			C2	C1	C0

1.This bit will get reset to '0' when 'Clear CA Parity Error' command is sent.
2.With RC0E DA3=1 and DA2=1 the device will forward commands and re-enable parity after the ALERT_n pulse. The 'CA Parity Error Status' bit will remain set. If a subsequent parity error is detected the device will re-enter the parity error state and set the '> 1 Error' bit.
3.The DDR4RCD01 will set this bit upon occurrence of a CA parity error.

- Parity error log register consists of four 8-bit wide registers
- Need 4 Control word writes -> DRAM MRS to transfer log

Exception Signaling Alternative: DNV Signal Usage in LPDDR2-NVM

- LPDDR2-NVM device drives data status with data
  - Good data
  - Retry immediately
  - Wait a while before retry
- Minimizes impact on low latency (DRAM) command scheduling pipeline
  - Data returned to requestor with additional status information
  - Does not have to immediately retry or reissue command
- May be used for LPDDR2-NVM
  - Not supported by LPDDR2-DRAM devices

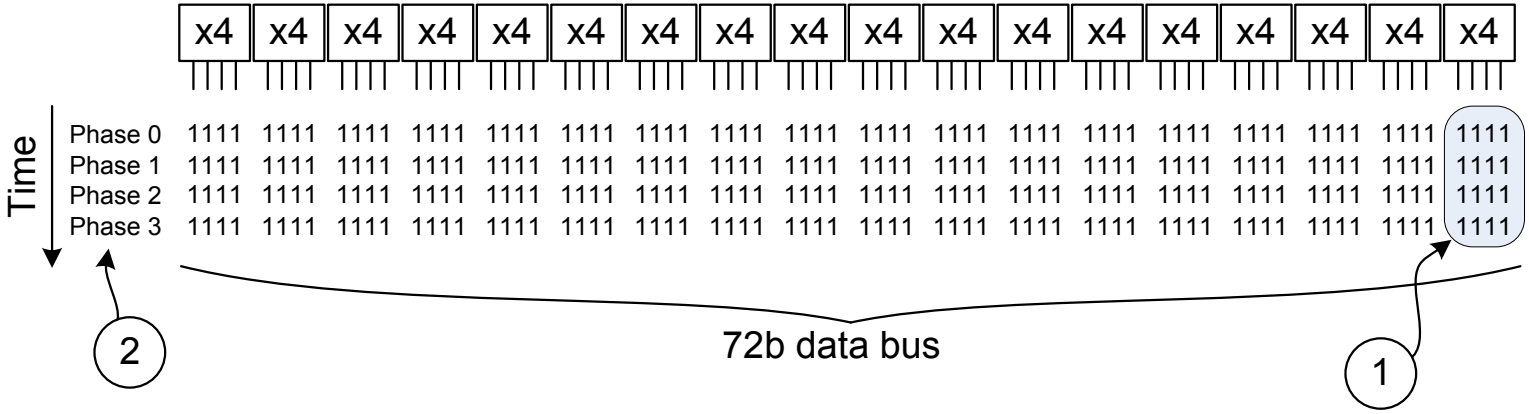
Cyclic Redundant Code (CRC)

- Protects against transmission errors (signaling fault)
- Detect any number of errors, adjacent or not
- Optional Write CRC for DDR4
  - Assume read data protected during transport by ECC/Chipkill
  - If enabled: Memory controller computes CRC based on data and sends along with transmitted data
  - Increases burst length from 8 to 10
  - DRAM device looks at data and computes CRC for comparison against CRC received along with data
  - If computed CRC differ from CRC received along with Data, signal data fault via Alert_n
  - In case of CRC error, re-transmission is typically the solution
- High speed memory system solutions use CRC to protect packet transport
  - Fully Buffered DIMMs
  - Hybrid Memory Cube

SECDED ECC

- Single Error Correct, Double Error Detect (SECDED)
- Typically used to protect against storage errors
- Detect and correct 1 (or more, in case of Chipkill ECC) of adjacent error bit(s)
- Check bits are computed by memory controller, sent along with data and stored with data.
- Memory device(s) - no knowledge or understanding of ECC.
- Upon memory access, data and check bits are retrieved together
- Memory controller re-computes check bits based on returned data
- If re-computed check bits differ from retrieved check bits, error has occurred.
- Difference vector used to distinguish between correctable single bit error or uncorrectable multi-bit error

x4 Chipkill Example



1. Each symbol is 16 bits, 4 bit width
2. 4 consecutive data phases

- Chipkill symbol size is 16 (16-bit) symbols
- Two (16-bit) check symbols
- Failure of asingle symbol can be detected and corrected
- Failure of second symbol (chipkill + SEU) may have small coverage gap
  - e.g. Intel Seaburg chipset (5400) capable of detection of 99.986% of all single bit failures that occur in addition to a x8 device failure.

Scrubbing

- "Scrub out" SEU-induced soft errors
- Prevent build-up of multi-bit errors
- Demand Scrubbing
  - ECC error? Write back data to location of error so you don't get the bad bit again
  - Did the error bit go away?
- Patrol Scrubbing
  - Constant, background read of data to ensure single bit errors are "scrubbed" out of memory device
  - Minimizes chance that single bit errors will exist if and when chipkill/component failure occurs
  - Hardware injected DRAM read commands that walk through physical memory once every XX hours

Summary

Summary

- Memory controllers are the glue that binds memory devices to host logic
- Many different varieties to support wide ranging set of requirements
- Attempted herein to describe
  - Commonalities where they exist
  - Design points or design choices depending on requirements