why memory is a hard problem in modern computer architectures

Felix Held

what this talk is (not) about

- very brief and high-level overview
- challenges to scale memory subsystem
- a lot of details omitted
- not the physical memory interface or its training

memory subsystem: challenges

- throughput
- latency
- physical memory interfaces

throughput

- relatively easy to increase
  - more data lanes
  - higher symbol rate on lanes
  - multiple bits per symbol (not used for DRAM yet)

latency

- hard to decrease, involves fighting physics:
  - speed of light in a medium
    - interconnect
  - RC time constants
    - C: MOSFET gates, storage capacitors
    - R: interconnect
    - doesn’t really decrease with die shrinks

memory subsystem: challenges

- data throughput steadily increases
- time to get a data word stays about the same

DDRx DRAM chips

- optimized for
  - capacity
  - cost
- architecture:
  - command bus
  - bidirectional data bus
  - multiple banks
  - organized in rows and columns

DDR3 DRAM

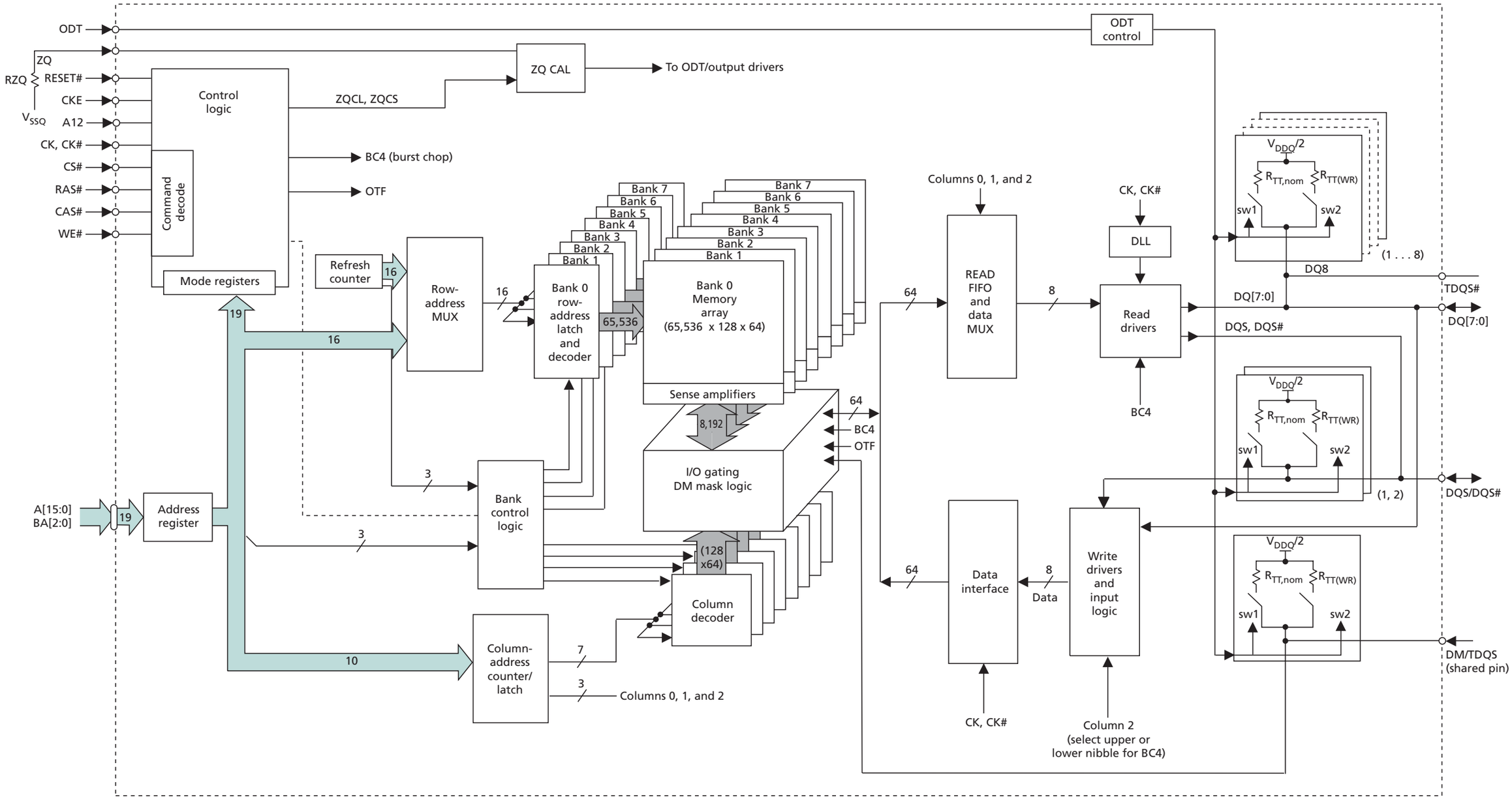
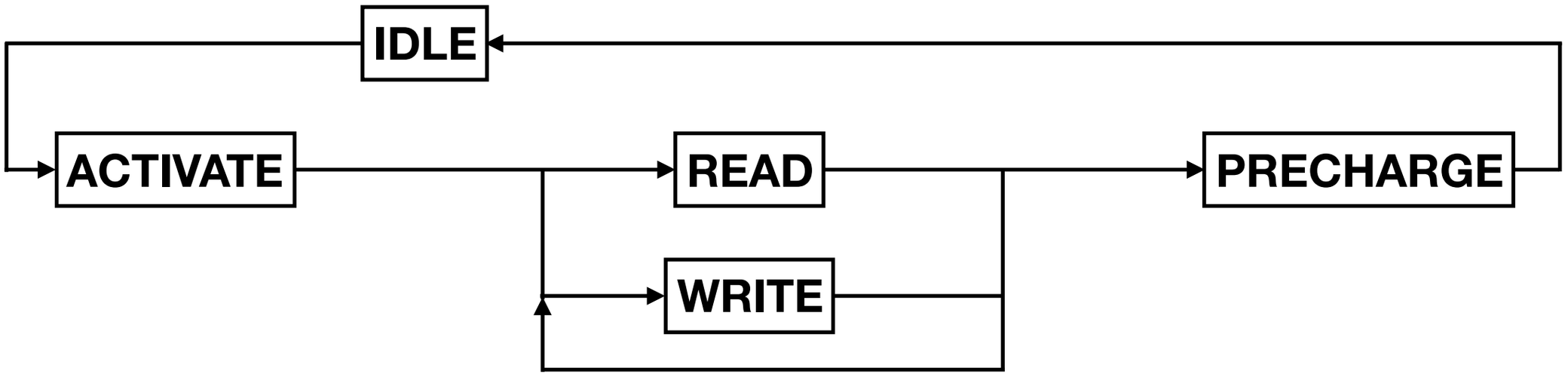


Image Source: https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/4gb_ddr3_sdram.pdf

DDRx DRAM chips

- data rate higher than command rate
  - double data rate only for data
  - two times for 1T command rate (usually used)
  - four times for 2T command rate
- burst data transfer with length of typically 8 data words

DDR3 DRAM



- open row
- tRCD delay to read/write

- access column
- READ: CL delay until first data word on bus
- WRITE: CWL delay until first data word is read from bus

- close row
- tRP delay to next ACTIVATE on same bank

memory controller

- process load/store requests from CPU/GPU/DMA/…
- preempt collisions on data bus
- RAM refresh
- make sure RAM stays in thermal budget
- map physical addresses to memory locations
  - channel, rank, bank, row addresses
- schedule memory accesses
  - higher memory bandwidth utilization

memory controller: address mapping

- channels
  - typically interleaved
    - higher bandwidth in one memory region
    - bit more power consumption
- ranks
  - command and data bus shared
  - internal state independent
  - mostly for more memory capacity

memory controller: address mapping

- banks
  - typically interleaved
    - work around tRP restrictions
      - delay to open new row in same bank
    - bit more power consumption
- row addresses
  - often not continuous, instead bank interleaving

memory controller: address mapping

- column addresses
  - direct mapping of least significant bits to column address
  - accessing different columns in one row: low latency
  - memory access patterns: typically continuous blocks

caches

- help solving part of the latency problem
- physics TL;DR:
  - large memory → high latency
  - small memory → lower latency
- memory accesses: often temporal and spacial locality
- keep a small region of memory in a fast memory
- (more or less) transparent

caches

- address bits except offset in cache line get hashed
- hash used as address for cache line in cache
- higher address bits stored as tag
- associativity: number of possible locations in cache for one hash value
- replacement strategies:
  - last recently used
  - last frequently used

caches

- common cache line size on modern x86: 64 bytes
  - DDRx DIMMs have 64 useable data bits
  - DDRx burst length is usually set to 8

  → 64 bytes, matches cache line size

cache hierarchy

- bigger cache → slower cache
- combination of bigger, slower caches and smaller, but faster caches
- inclusive or exclusive

cache write strategies

- write through:
  - every write is propagated to the next cache level or main memory
  - less state tracking needed
  - more write bandwidth used
- write back:
  - contents only written back when line gets evicted
  - less writes to higher cache level or main memory
  - more state needs to be tracked and propagated

CPU microarchitecture: load buffer, out-of-order exec

- load buffers:
  - memory reads have not very well predictable latency
  - wait for data shouldn’t stall the pipeline
- single thread performance: out-of-order execution
  - execute instructions that don’t depend on this read
  - try to keep the processor busy
- multi thread performance: (possibly) hyper-threading

cache prefetching

- speculative loading of memory contents into cache lines
  - minimize read latency
- instruction fetch
  - branch prediction
- data reads
  - access pattern prediction
  - cache hinting instructions

cache prefetching

- right data prefetched
  - latency reduction
- wrong data prefetched
  - unnecessary memory bandwidth usage
  - eviction of other cache lines
- prefetching instructions to give processor hints

cache coherency

- problem in Symmetric MultiProcessing systems
- different cores should always see same data in one location at a time
- programming models for SMP assume cache coherency
- doesn’t scale well with the number of cores
- bus snooping vs directory-based
- write-invalidate vs write-update
- different cache coherency protocols

bonus slide: memory ordering

- under certain conditions reads and writes can be reordered
- specified in the instruction set architecture
- needs to be taken into account when building synchronization primitives
  - insert (expensive) memory barrier if necessary

Thank you for your attention.

Questions?