Part III. Embedded Memory Systems

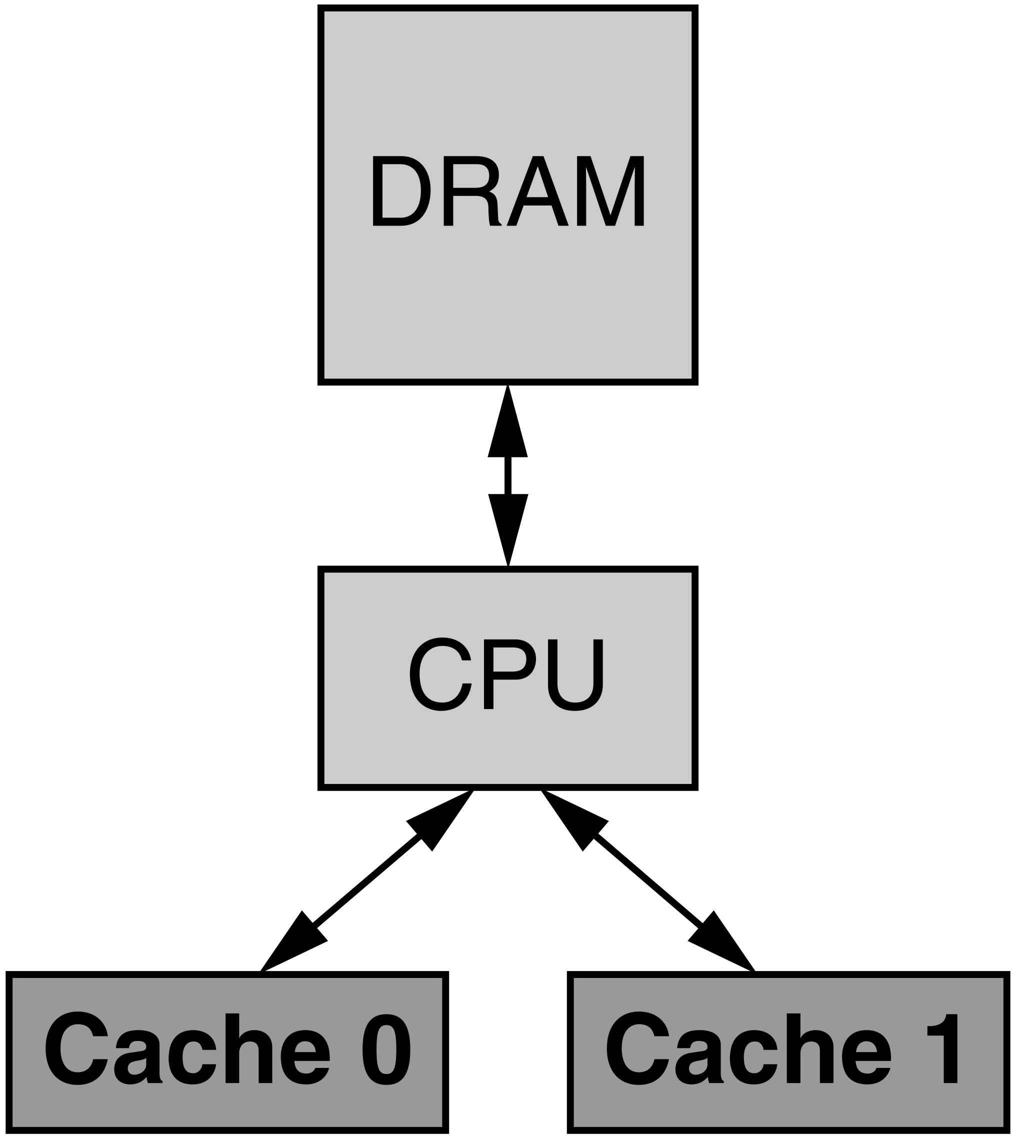
University of Maryland
1856

Today’s Story

- **The memory system now dominates performance & power. Embedded systems “solve” issues now confronting gen-purpose. => Take a few notes from the embedded playbook:**
  - DSP & embedded-processor memory systems
  - Better Cache designs for power and performance
  - Better DRAM designs for power and performance
  - High-performance systems as embedded systems
  - Treat DRAM/main-memory as cache, larger block size
  - **Issue:** Software management of memory systems
  - **Issue:** Parallelism & non-conflicting assignment of resources

DSP/Embedded Memory Systems

DSP “cache”

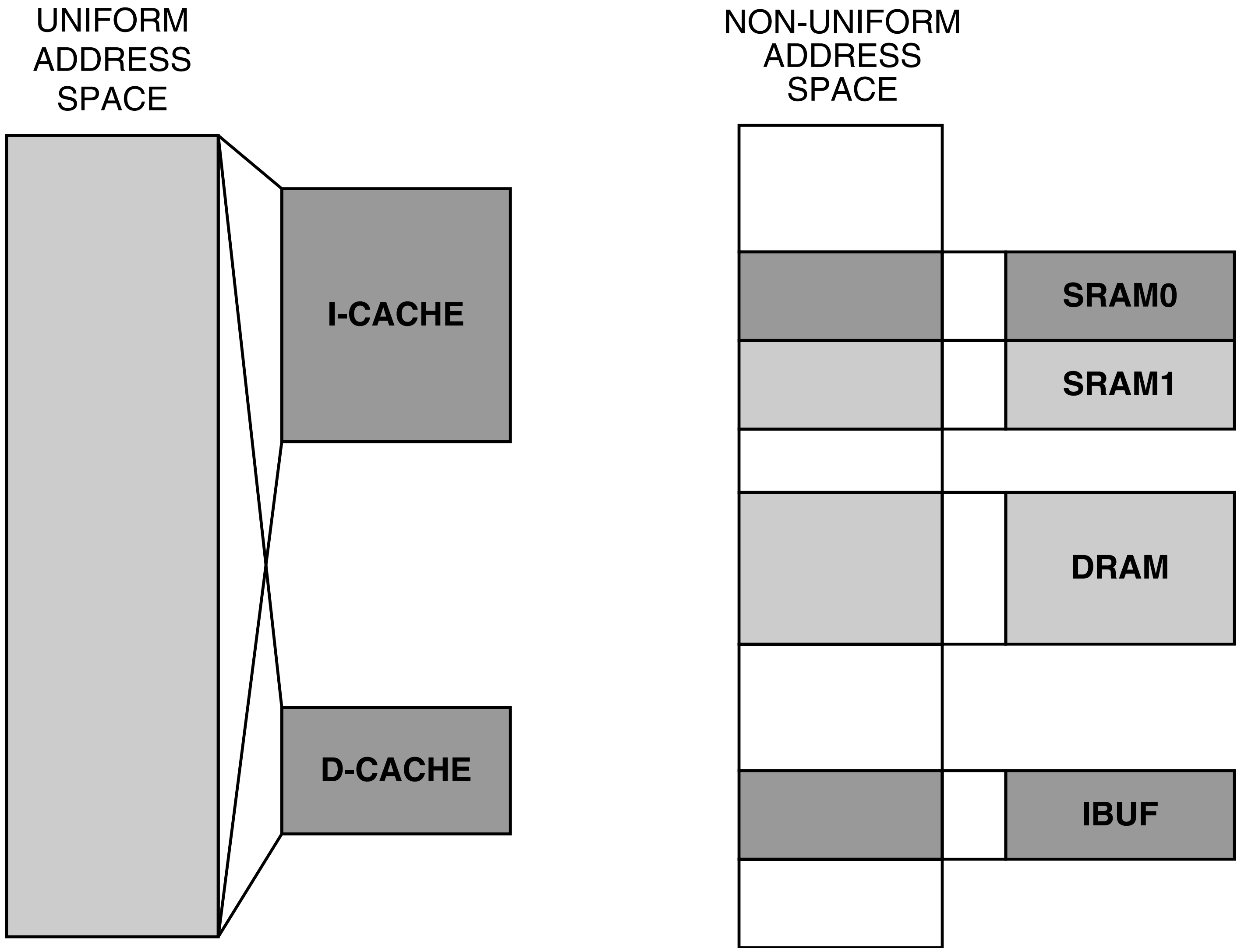


DRAM

CPU

Cache 0

Cache 1



UNIFORM
ADDRESS
SPACE

I-CACHE

D-CACHE

NON-UNIFORM
ADDRESS
SPACE

SRAM0

SRAM1

DRAM

IBUF

Software-visible view

DSP/Embedded Memory Systems

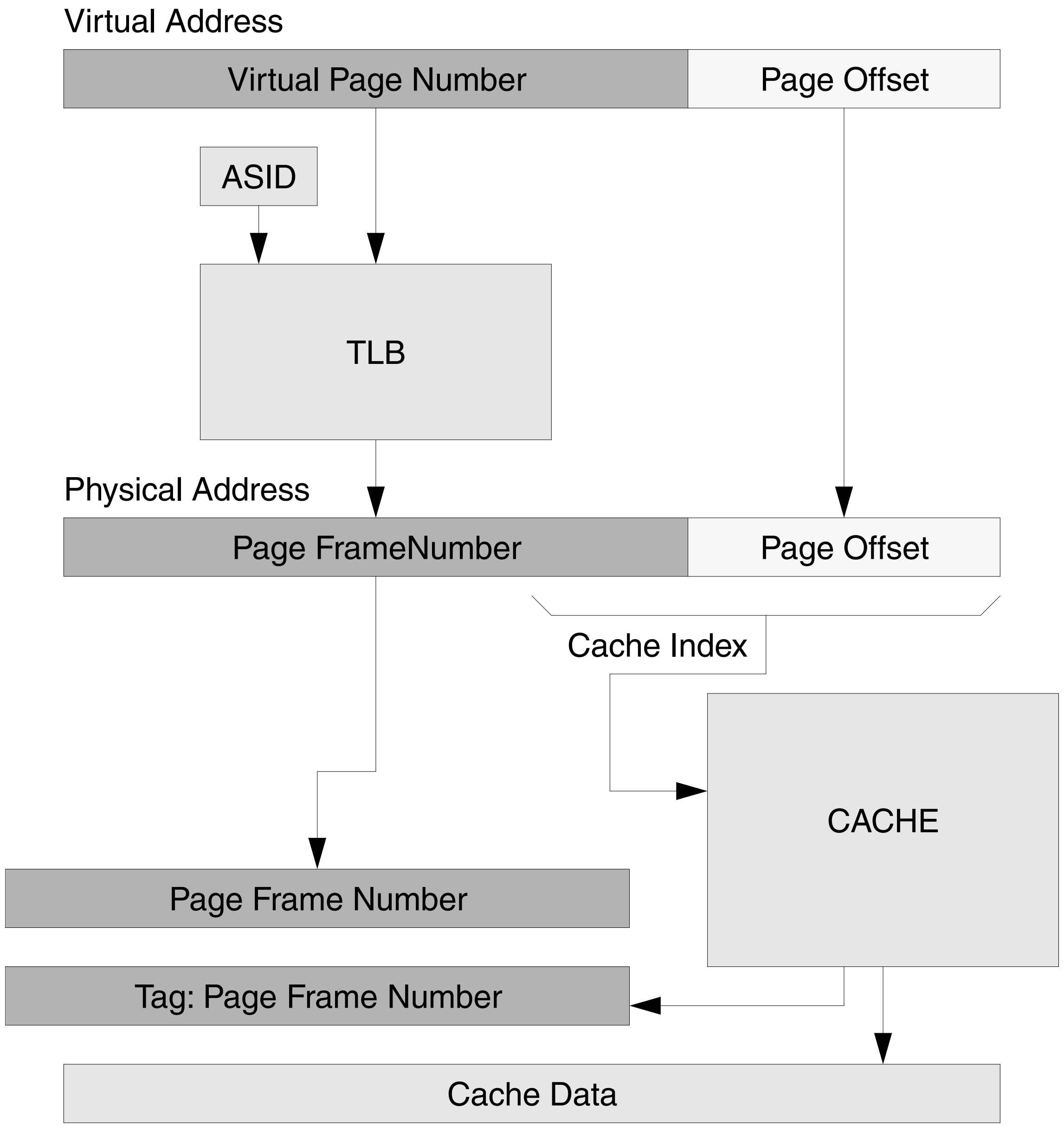
- Software schedules accesses to different technologies;
  this breaks abstraction, but it improves efficiency
  (e.g. studies show scratch pad beats transparent cache)
- Multiple busses to memory => *much* better streaming performance
- Issue of compilation: transparent cache is a much easier compiler target
- Interesting concept, not fully explored: an item’s name indicates its properties as well as its location:
    - Read-only/read-write/executable/non-executable
      Volatile/non-volatile
      Cacheable/non-cacheable … etc.

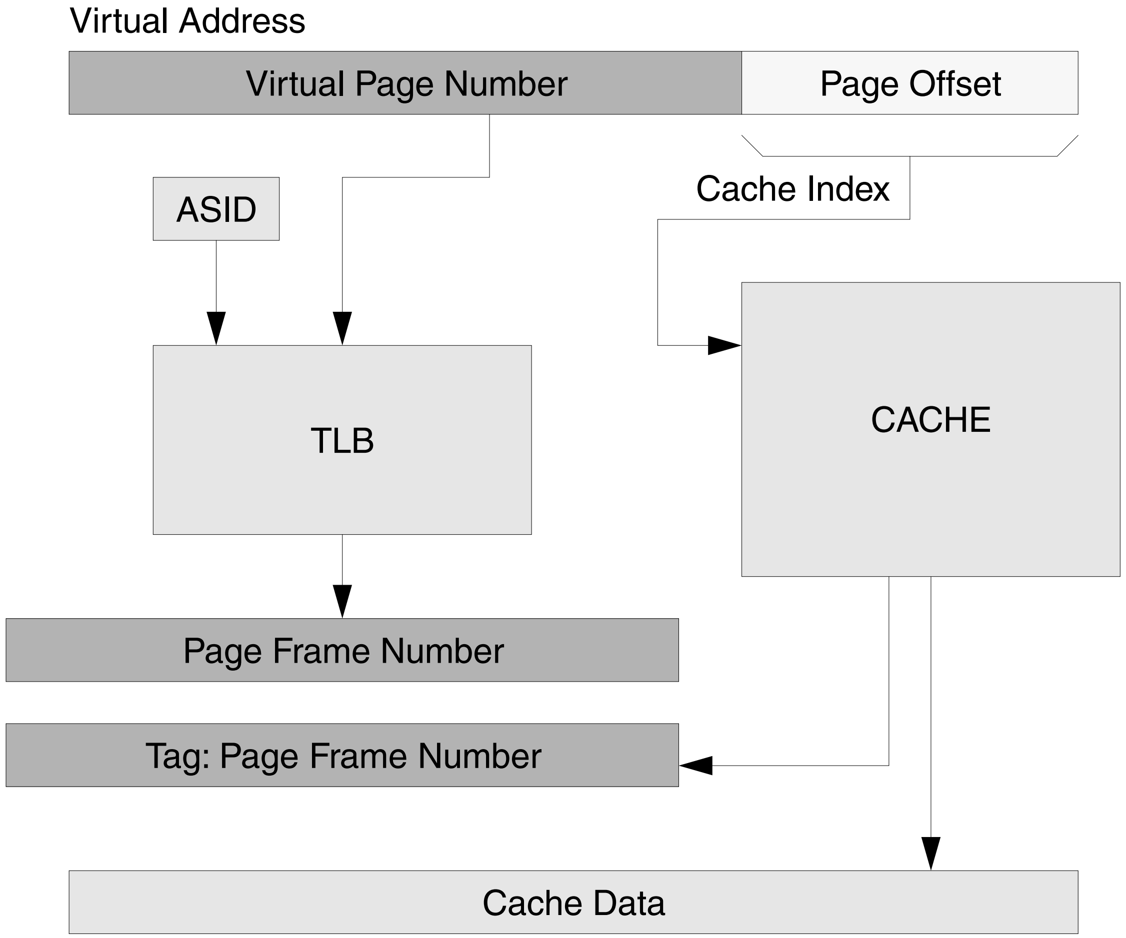
Why That Last Bit Might Matter

- POTENTIAL REGIONS:
    - SRAM (0, 1, 2, etc. … also L1, L2, L3, etc.)
    - DRAM
    - Flash/PCM/whatever solid-state non-volatile memory you choose
    - Disk
    - Network?
- Back to the SASOS Concept …

[SASOS Discussion]

Cache for Power & Performance

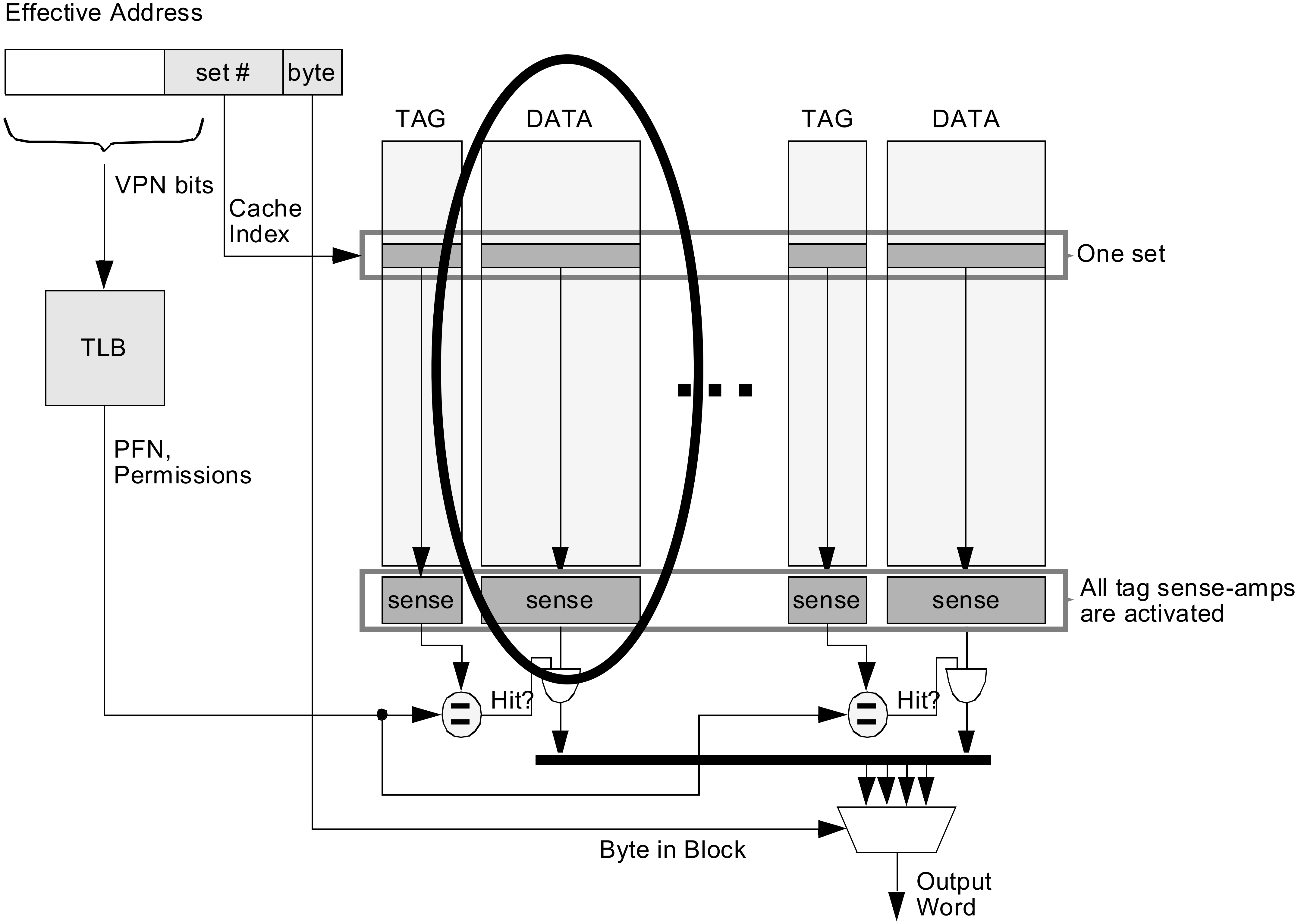

Virtual Address
Virtual Page Number
Page Offset
ASID
TLB
Physical Address
Page FrameNumber
Page Offset
Cache Index
CACHE
Page Frame Number
Tag: Page Frame Number
Cache Data


Virtual Address
Virtual Page Number
Page Offset
Cache Index
ASID
TLB
CACHE
Page Frame Number
Tag: Page Frame Number
Cache Data

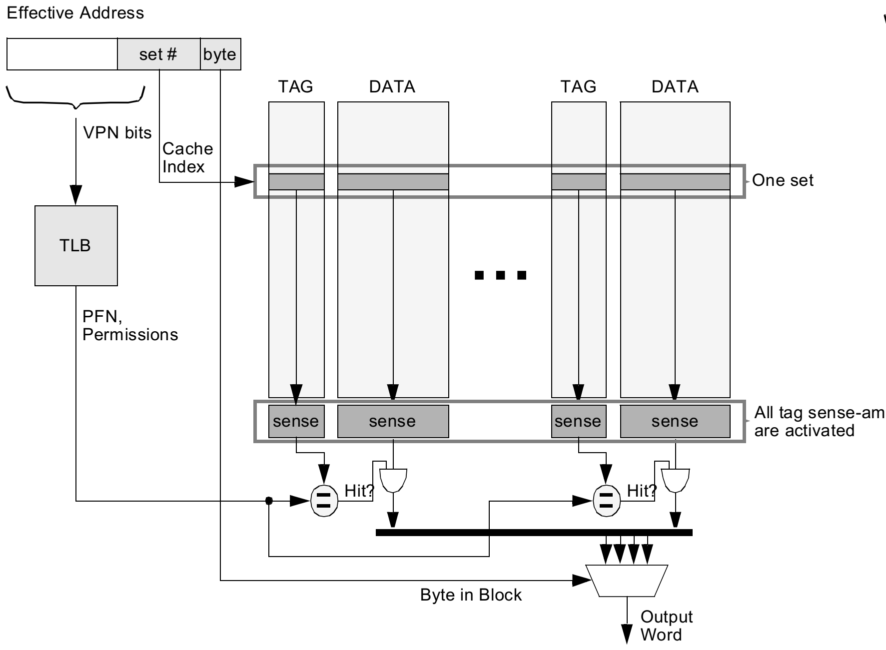
Windows assumes physical cache (left)
to solve aliasing problem.

[Aliasing Discussion]

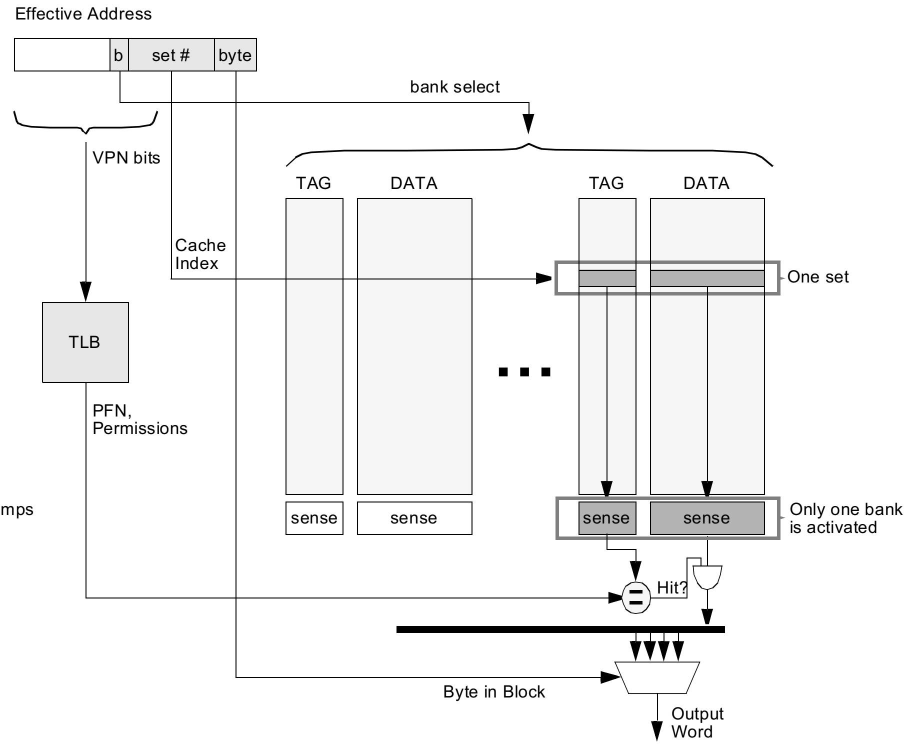
Main Issue: This Cannot Exceed 4KB Page Size



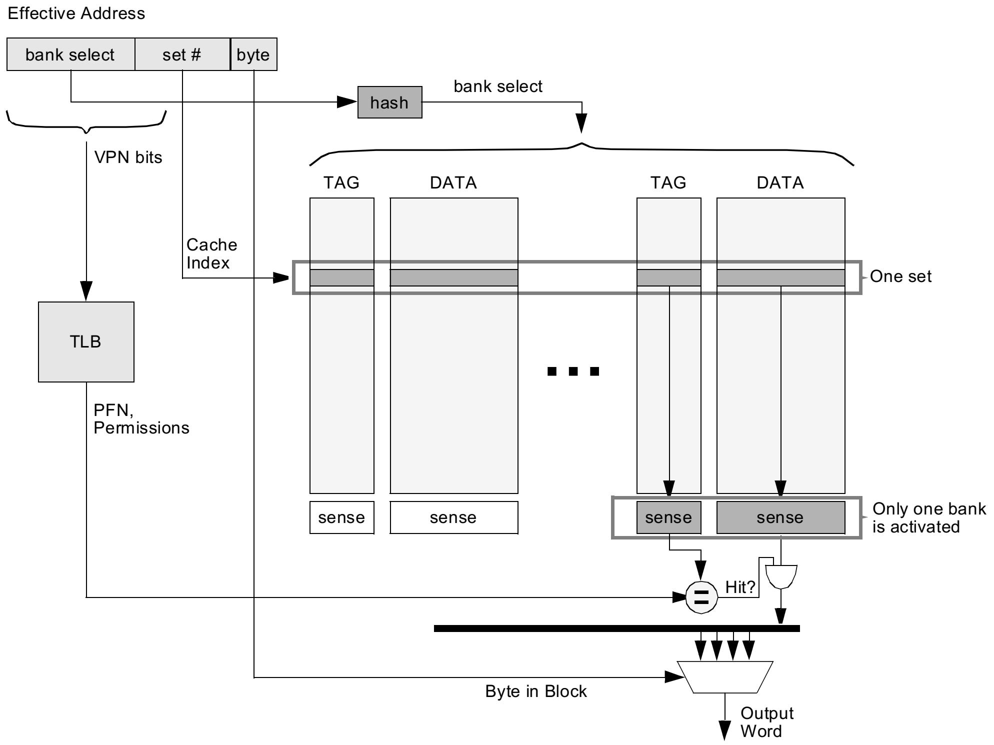
A Solution: Hash-Associative Cache



(a) Traditional n-way set-associative cache

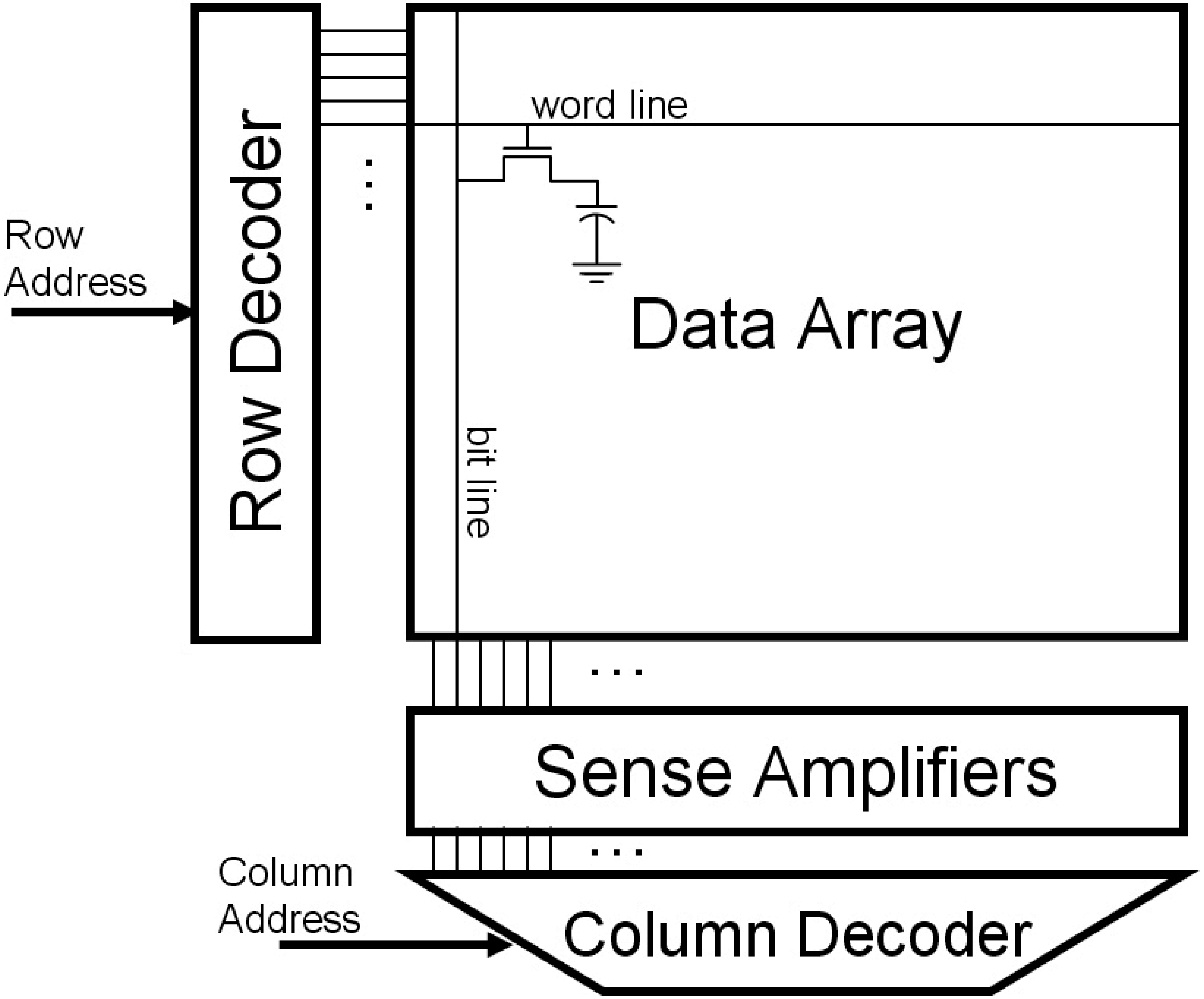


(b) Traditional direct-mapped cache, n banks



(c) Hash-associative cache

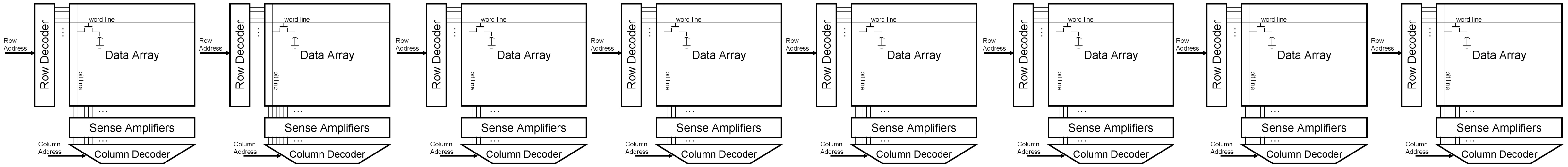
DRAM Designs for Low Power: One DRAM



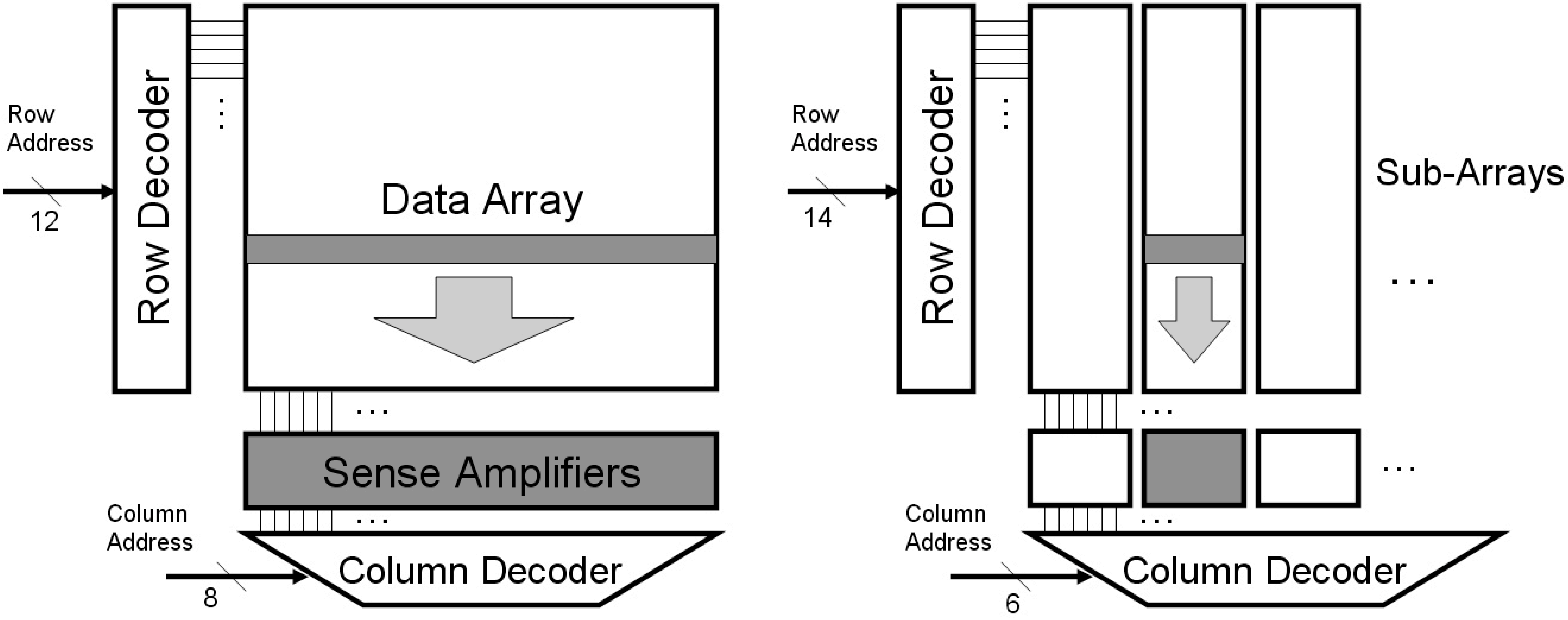
problem: lots of bits are read per bank activation

Per Rank (at the DIMM Level)

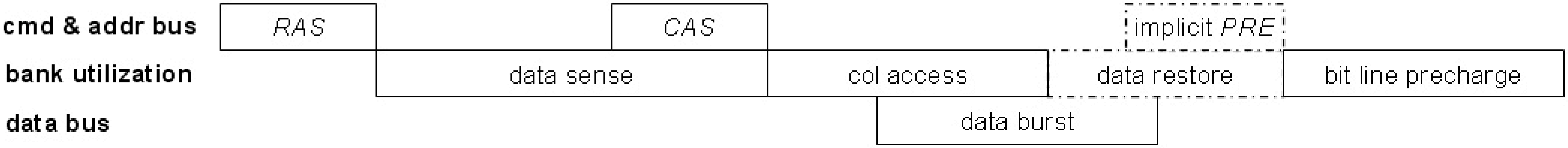
- Each DRAM device drives & senses ~8K capacitors, sense amps
- Eight devices per rank => 65,536 such discharge/sense cycles, all to read 512 bits of data.
- This is somewhat inefficient



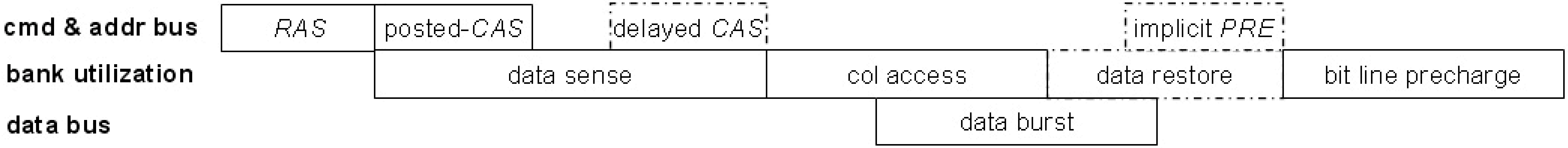
A Better Approach: FCRAM



Posted CAS



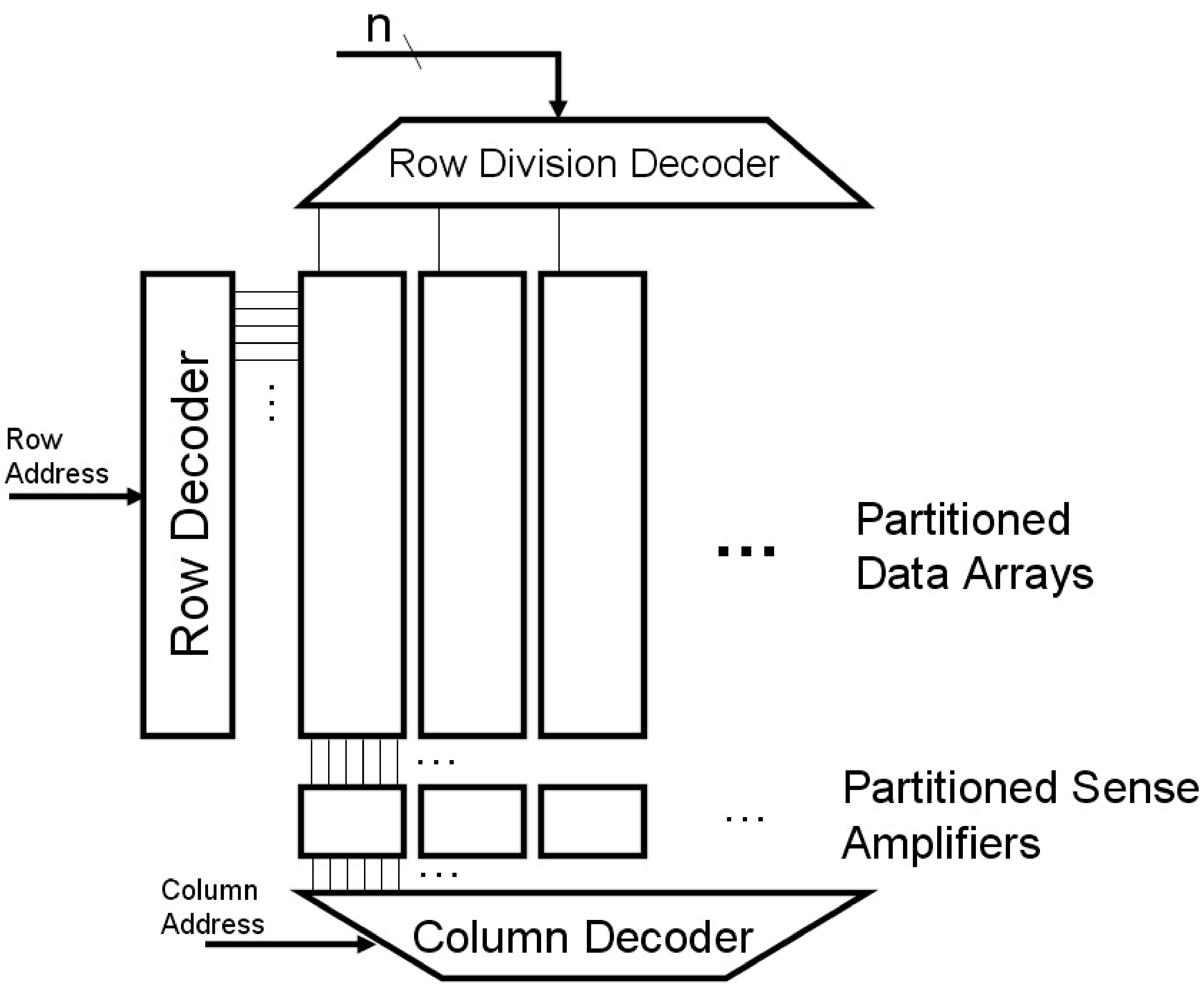
(a)



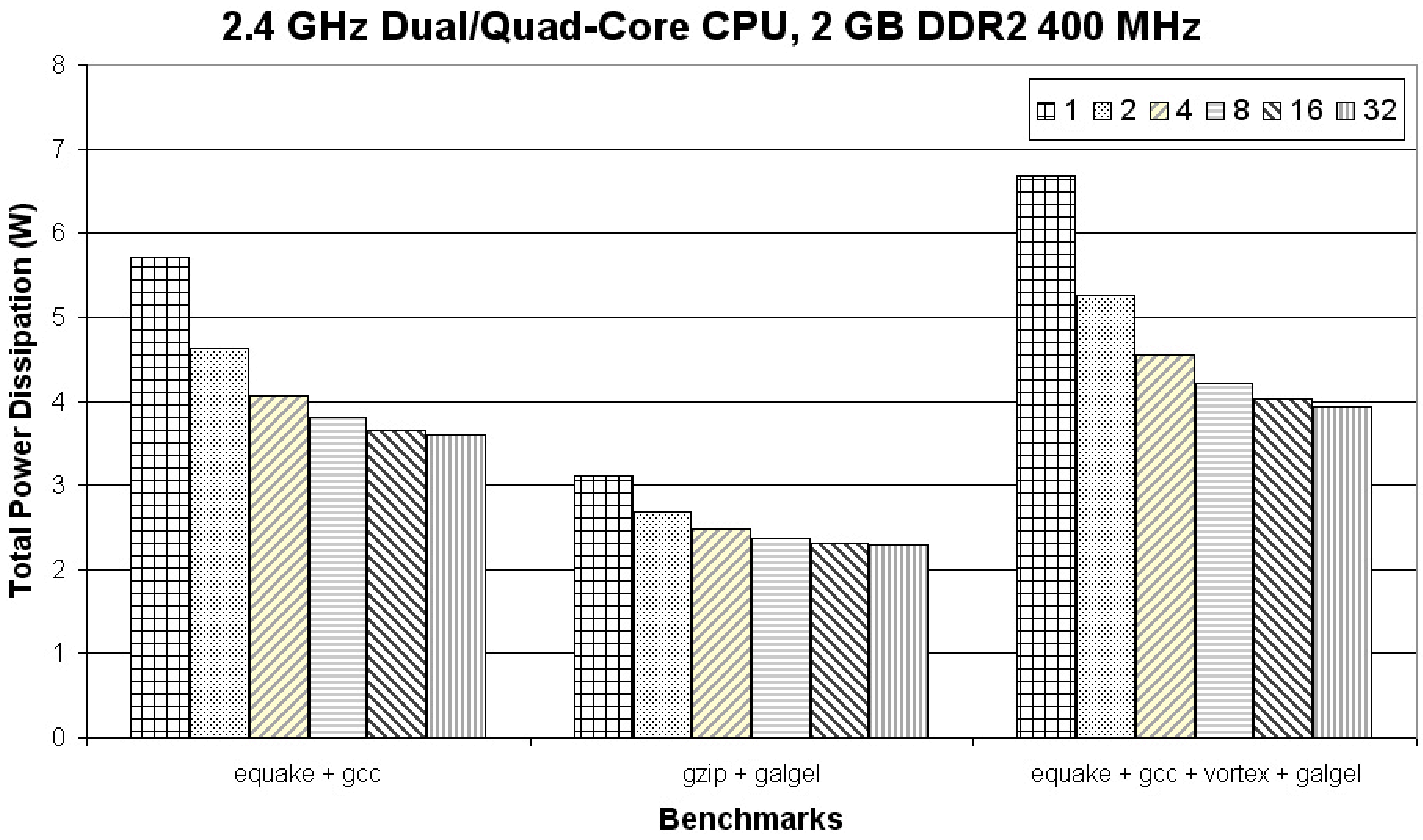
(b)

Main point: Column address is seen by DRAM *early*

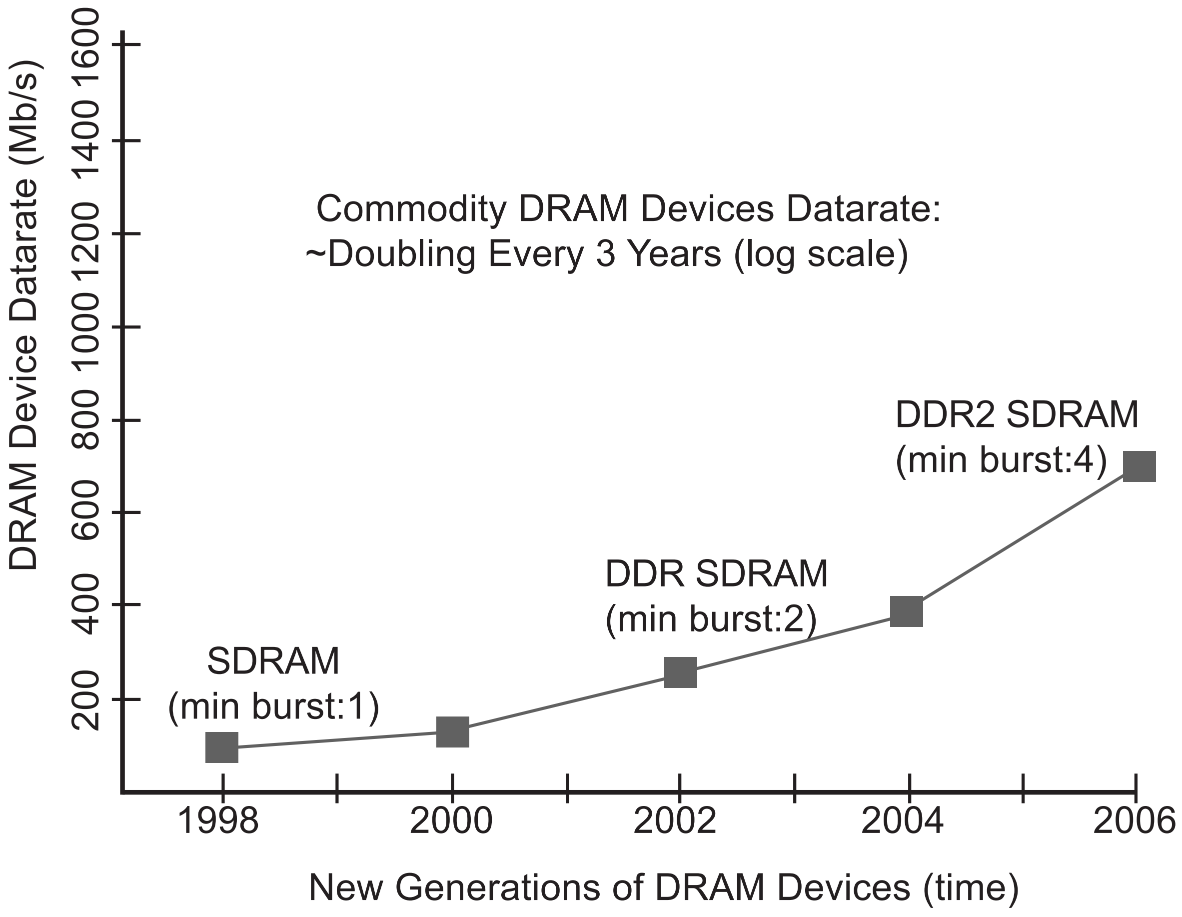
Combine the Two: Fine-Grained Activation

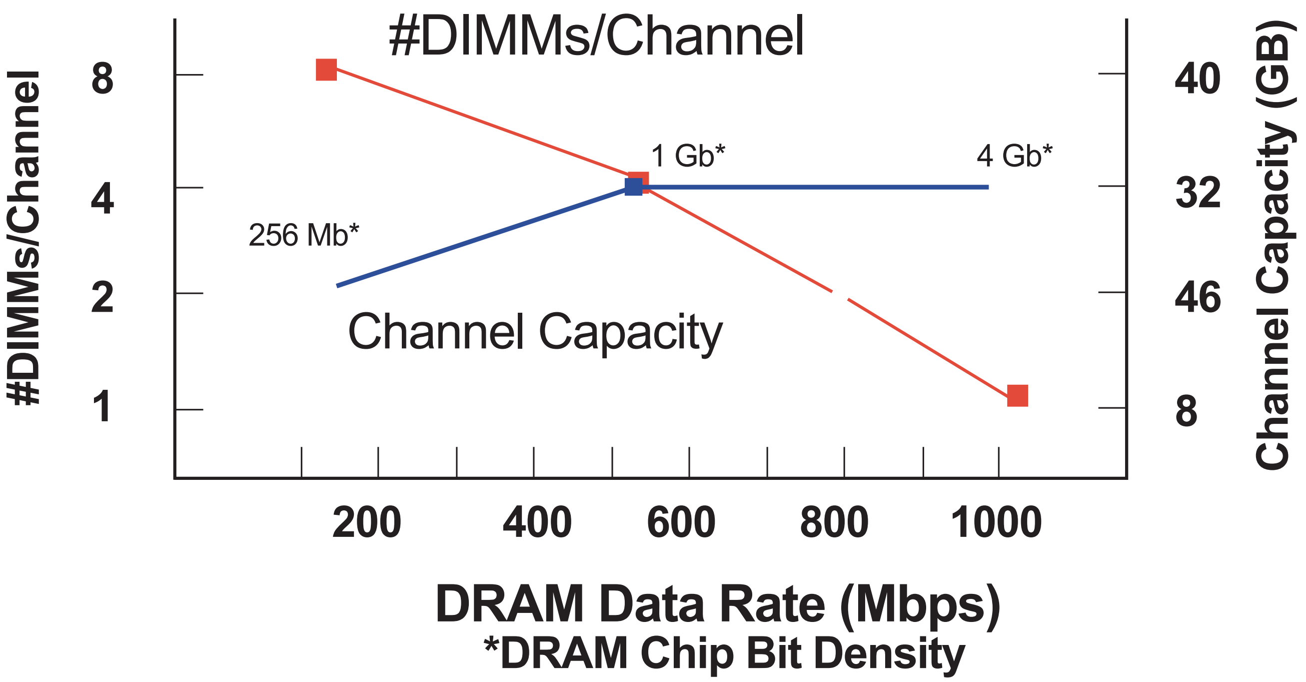


Power Savings



DRAM Designs for High Performance





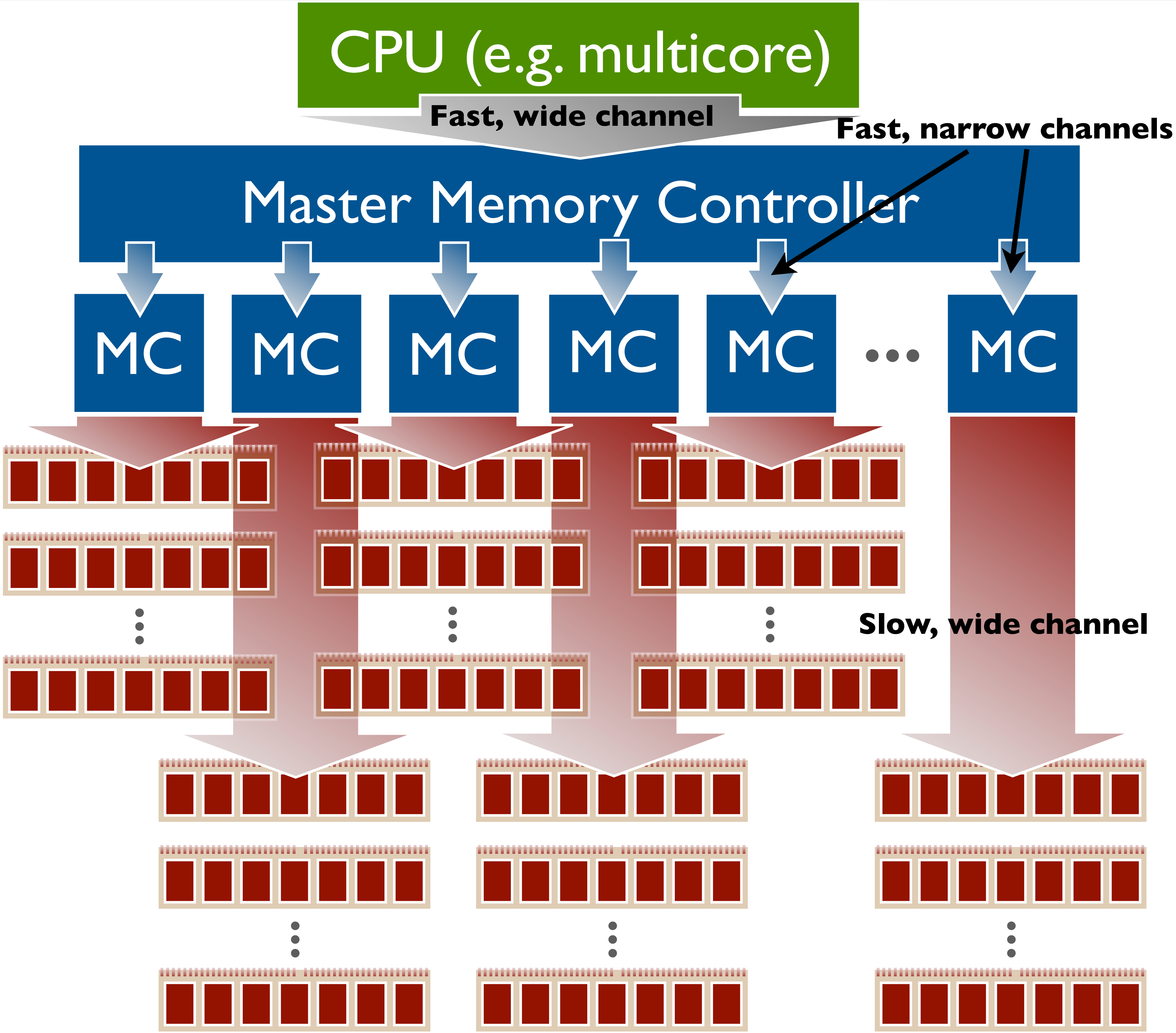
The capacity problem

Fully Buffered DIMM

JEDEC DDRx
~10W/DIMM, 20 total

FB-DIMM
~10W/DIMM, ~400W total

A Solution: BOMB



Additional Issue: Granularity

- Assertion: 4KB page has outlived its usefulness
- e.g., Google File System: 64MB pages
    - reduces transfer overhead
    - reduces mapping overhead
    - increases sequential benefits
    - etc.

Enterprise & Super- Computing

- Run same app (set of apps) 24x7
- Developers spend significant time/energy optimizing apps
- Frequently run a custom (or at least fine-tune the existing) OS
- Have significant, pressing correctness/failure/dependability issues
  => *not intrinsic to application area, but because of large-scale multipliers*
- Care very deeply about energy consumption
  => *not intrinsic to application area, but because of large-scale multipliers*
- Sounds a lot like embedded systems, no?

Some Thoughts & Discussion

- **Use embedded processors**
  (power & heat problems reduce)

- **Use software management of memory hierarchy**
  (performance can increase, scheduling problems are reduced, power can decrease, checkpoint & restore becomes trivial, etc.)

- **Need to pay close attention to resource-mapping issues**
  (10x performance degradation for poor resource utilization in parallel systems)

- **As long as we're rewriting the OS, incorporate solid-state non-volatiles**
  (e.g., to support distributed & memory-mapped file/object system,
  to divide up read-mostly versus write-often data,
  to reduce network I/O traffic, etc.)