“This is weird”

The paper is “Memory that never forgets: Emerging nonvolatile memory and the implication for architecture design”
* Similar to last paper, includes racetrack memory, circuit level considerations and more detailed description of NVM shortcomings, new research integrating NVM in computer system

Emerging NVMs:
* PCM: compatible to CMOS, good scalability, access speeds comparable to DRAM
* RRAM: small cell factor, access speeds comparable to DRAM (not discussed much, until end)
* STT-RAM: good scalability, access speeds comparable to SRAM
  * Qishen: really useful but not mature enough, performance limited ==> read and write times are really slow
* RM: similar to STT-RAM but even higher density, domain shifts needed for access
  * What is the layout of data? How do bit shifts work?
  * Shift-based read and write, shifting slows accesses, seems to serialize reads and writes (makes the process slower?)
  * Semanti: Why is this not being adopted despite all the advantages?

Common ground on emerging NVM
* NVMs have nearly zero standby power
* NVMs allow for higher density (smaller size, multi-level cells)
* NVMs have high write latency and consume more dynamic power with writes :(
* NVMs wear out faster compared to SRAM and DRAM :(
* STT-RAM, RM good candidates for on chip cache and PCM for main memory

Architecture Workarounds for Integrating NVM
* Reducing number of writes: optimize cache replacement policies to reduce writebacks to NVM LLC or main memory
  * Caches currently replace LRU or LOU data, paper suggests other data (not modified, so no rewrite in NVM ⇒ hurts the hit rate of the cache, what level of a cache might this make sense?)
  * NVSim (circuit level simulation) and NVMain (architecture level simulator) both mentioned in paper to study cache replacement policies, etc…
* Partial write schemes that only update modified data (read first, then write)
* Data compression for frequently written data
  * Does the dynamic power from writing used actually benefit the system compared to the static power? It depends on the application (number of reads/writes; not worthwhile in a first level cache, potentially more so in main memory)
* Preemptive write buffers (hold writes until all reads done), setting all bits to 1 prior to a write, then update to a 0-bits when writing to valid data (counter intuitive, because double time and energy, but can

hide this during other operations if everything is going to be reset anyways)
* Wear-leveling to periodically switch data write locations
  * Qishen: this is already being used, moving is nontrivial ⇒ similar to hybrid NVM and VM architectures? Commercially available cache for SSD, specs to be sent to discussion… same issue, so same technique should be used - read and write are different than at other levels of the memory hierarchy
* Hybrid memory workarounds
  * NVM only provide workarounds for multi-level cells versus single level cells
  * Can relax STT-RAM non-volatility: cannot guarantee that entire memory structure is nonvolatile, some parts are volatile (Karpur: spintronics are underlying material of the memory rather than what causes the volatility) ⇒ relaxing the volatility requirement, not the memory itself, to optimize the speed of the memory
    * Interesting to explore loss of data as writability wears out? Losing volatility over time?
    * Ratio of volatility? Ideal amount of volatility to non-volatility for caches? Workload dependent, lots of writes to small fixed area would be advantageous

New research areas with NVM
* Security for NVM-based main memory
  * Counter-mode encryption for secure NVM
    * Compression: doubly advantageous for NVM - smaller write and ensuring that the state of the data isn't preserved after power down
    * Word level counters are nice: reducing the write intensity and helping with wear across, potentially a storage overhead (quantifiable?)
    * New method reduces write intensity: how? Sneak paths as it relates to security? How random is the initial number (fabrication variation? Addresses? Both might impact the level of randomness)?
* Persistent main memory and data consistency
  * Between storage and volatile main memory (totally replace?)
  * Persistent storage is NVM (retains state), flip side of the security issue ⇒ quickly power up and recover data faster without fancy logging mechanisms
* NVM-based storage + file systems
  * What used to happen in disk is now happening in NVM
* Non-volatile processors
  * Every component of the processor can be made non-volatile, processes without power but relatively limited in capability
* Processing-In-Memory (Near Data Processing)
  * RRAM reappears as potential memory candidate
  * Embed computing inside memory: simple but widely required computation (NN computation becomes key candidate), bias towards computing in memory rather than talking about potential 3-d stacking and thermal issues (how sophisticated is the processor core?)

* Approximate computing as a potential area as well to optimize writing? Must be application aware, if it can tolerate that type of approximation

* NVSim: allows users to simulate different memory architectures using different technologies, i.e. in this technology what will be the access time and/or leakage? Read v write access time? (homework coming up next week!) :-)
* NVMain: different architectures can be simulated (potentially used for final project)
    * Other simulation tools as well for analysis of designs