My Memory Ain't What is Used to Be
Directions in Memory Interfaces

- **Memory Wall**: computation speed increases faster than memory access time

- Uniprocessor
  - Load and Store operations guaranteed to complete in program order
  - Parallelism (at runtime): consecutive loads, access to different addresses

- Multiprocessor
  - Consistency model defines in what order operations are applied to memory and in what order updates are seen by other processors
    - Processors use coherency protocols (e.g. MESI, MOESI)
    - Sequential consistency (SC): total ordering and sequentialization
    - Relaxed models provide higher performance but are harder to program
      - Use memory barriers to prevent reordering
    - Implicit vs. explicit coherence models
      - Explicit models invoke coherence mechanism explicitly
My Memory Ain't What is Used to Be
Directions in Memory Interfaces

• Transactional Memory
  – Atomically execute a sequence of instructions and rollback if necessary
    • Pros: optimistic, less deadlocks, flexible synchronization granularity
  – Transactions in Hardware
    • Compare-and-Swap less useful for complex data structures
  – Transactions in Software
    • $atomic\ (predicate)\ \{\ ...\ \}$

• Functional languages
  – Only have read-after write dependencies -> consistency and coherence easier but hard to have mutable states
    • I-structures, m-structures and monads
My Memory Ain't What is Used to Be Directions in Memory Interfaces

• Streams and Vectors
  – Stream programming: *kernel* and *streams*
  – Continuous streaming or chunked streaming are possible implementation in hardware
  – Vector processors provide instructions that operate on entire vectors at once (also LOAD and STORES).

• Memory aliasing
  – Hard to determine at compile time if pointers reference same memory location, this limits compiler optimizations
  – *restricted* pointers in C99, *no-middle-pointers* in Java
  – memory dependency predictors (in hardware) are good at the moment but might be insufficient if instruction windows are 100s or 1000s of instructions

• Spatial Architectures, e.g. WaveScalar

• Processor in Memory: integrate computing resources and large memory on the same device
Questions

- Can functional languages exploit their “only read-after-write” property on current hardware (e.g. x86) or is special hardware support needed?
- Is the memory wall still getting “higher” or is it flattening out because clock frequencies also flattened out as well and we have parallelism due to multiple execution contexts?
- How do Intel’s upcoming Transactional Synchronization Extensions (TSX) work?
  - What adaption if any is necessary to languages to exploit this?