

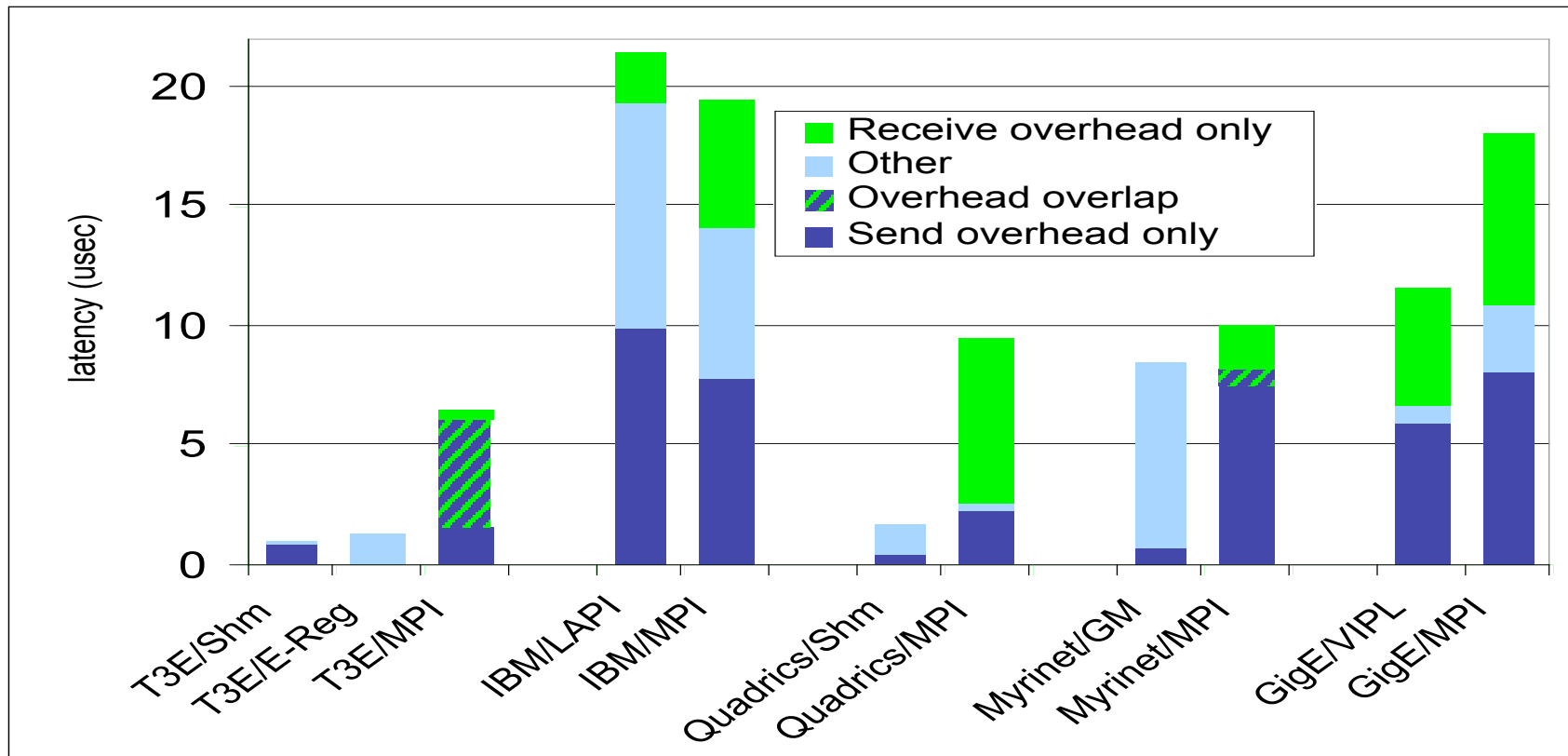
The slide features a decorative arrangement of seven circles. Three circles are positioned in a top row, and four circles are in a bottom row. The top row consists of an empty circle on the left, followed by two solid light-purple circles. The bottom row consists of two solid light-purple circles on the left and one empty circle on the right. The text is centered horizontally across the slide.

Software Caching for UPC

Jason Duell

Motivation – Network Latencies

UPC accesses	Private	Local shared	Remote
No. Cycles	1-2	~10	> 2K

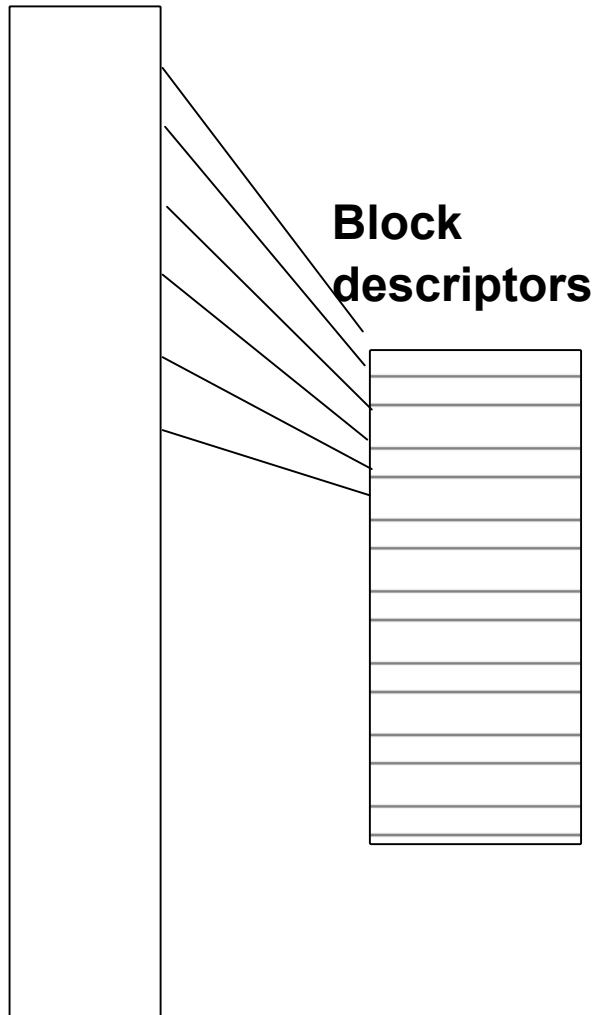


Caching and the New Memory Model

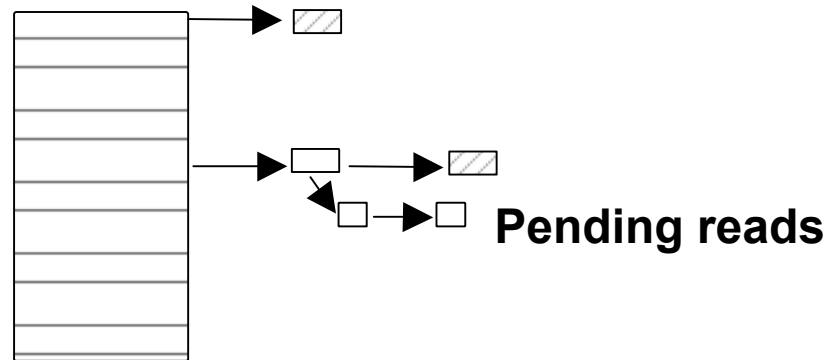
- Model forbids data races on relaxed accesses
 - So it's OK for cache to not handle them intuitively.
- Relaxed reads no longer required to observe remote strict writes (in absence of synchronization)
 - A “local knowledge” scheme, so no need for coherence/invalidation messages.
- Permits an efficient caching implementation
 - Cache remote relaxed accesses
 - Possibly coalesce relaxed (or even strict!) writes
 - Flush the cache (and write buffer) at synchronization points (barriers, locks, fences, strict accesses)

Cache Organization

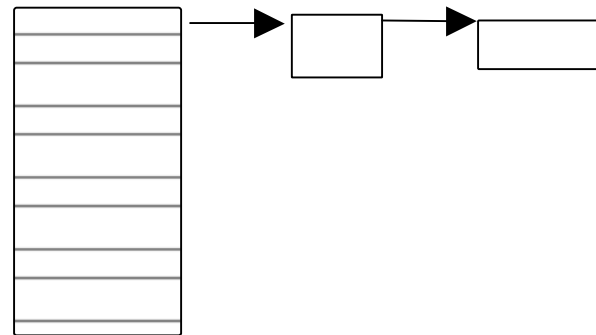
Cache Blocks



Hashtable (shared address -> block descriptor)



Pending writes (per UPC target thread)



Cache Operations



- Relaxed Reads:
 - service from cache, if resident. DONE.
 - allocate cache block (evicting if necessary)
 - find and sync any conflicting pending writes
 - initiate remote cache block read
 - chain pending request off of descriptor
- Relaxed Writes:
 - find and sync any conflicting pending block reads
 - initiate network message for the write
 - write to cache block (if resident)
 - store write request in list of pending writes
- Cache flush: at any strict read/write, barrier, etc.,
 - Just bzero hashtable and block status bits

Cache Operations

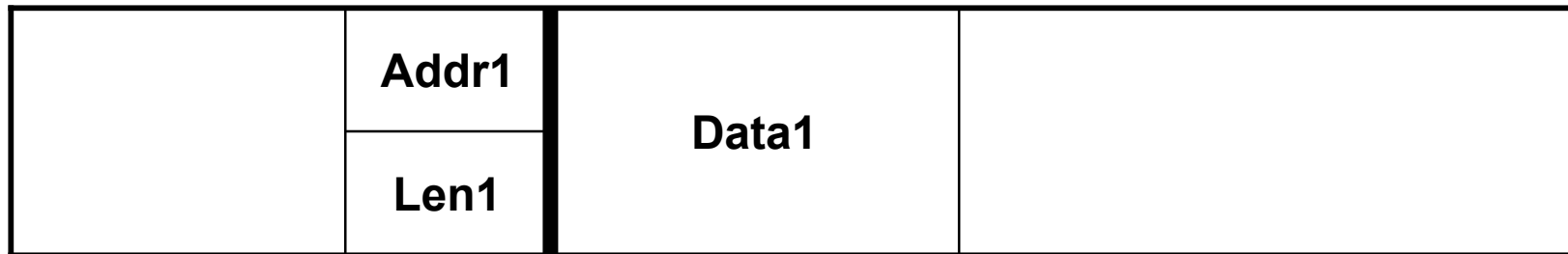


- Bulk and unaligned accesses
 - Unlike hardware cache, accesses of arbitrary length and start address
 - Fastpath assumes all accesses fit into a single cache block
 - Bulk operations usually imply hand-optimization: bypass cache
- Bulk writes: (same as small, but affect multiple blocks)
 - find and sync any conflicting pending block reads
 - initiate network message for the write
 - write to cache block (if resident)
 - store write request in list of pending writes
- Bulk reads:
 - Need to check for pending conflicting writes: flush them
 - But can skip storing read memory in cache
 - New memory model allows older values to remain in cache

Write-packing buffer

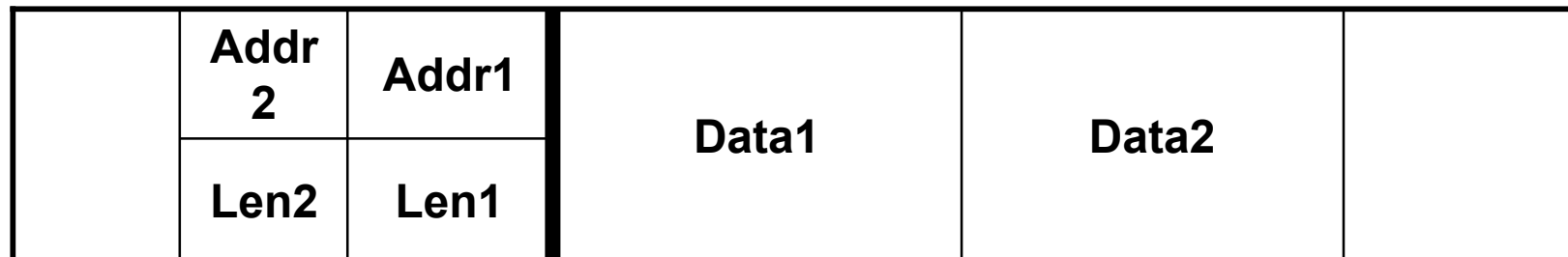
Delay writes, and place into per-node write-packing buffer.

-store addr/len info separately from the data



Pack later messages around earlier ones .

-results in fewer, larger messages



Write-packing buffer (cont.)

Contiguous writes can be coalesced into a single message
- just change the length field



- Strict writes can be packed into buffer, as long as remote side unpacks writes in correct order
- Min/max address can be kept to speed up conflict checks
- Also, trivially handles write-after-write conflicts: will be useful if conflict detection moved from compile-time to run-time

Preliminary Results

	Local	Remote	Cache hits
Time (ns)	20	27000	350

