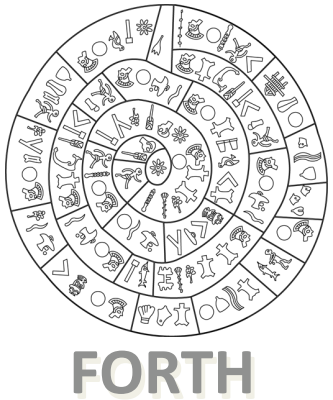
FPGA Implementation of a Configurable Cache/Scratchpad Memory with Virtualized User-Level RDMA Capability

G. Kalokerinos, V. Papaefstathiou, G. Nikiforos,
S. Kavadias, **M. Katevenis**, D. Pnevmatikatos, X. Yang

FORTH-ICS, Crete, Greece



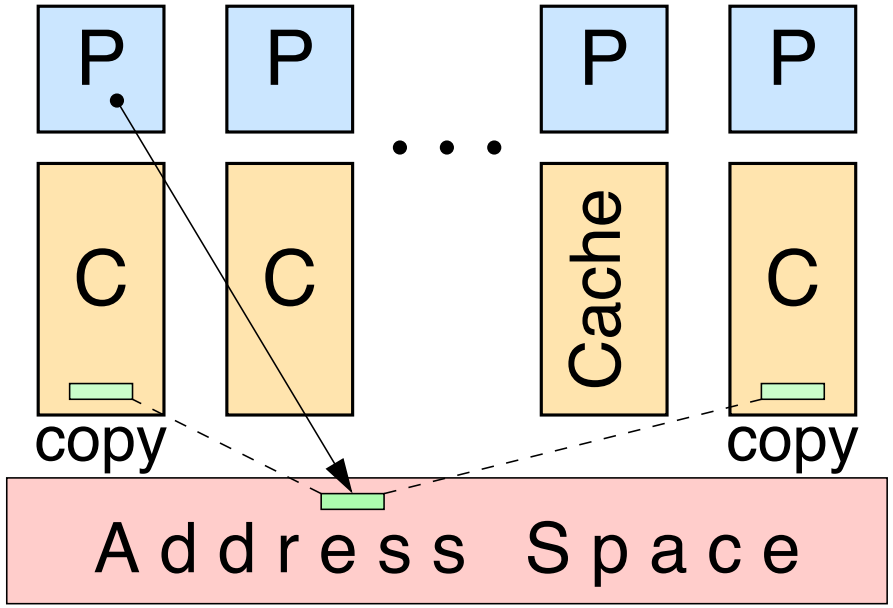
SAMOS – July 2009

Outline

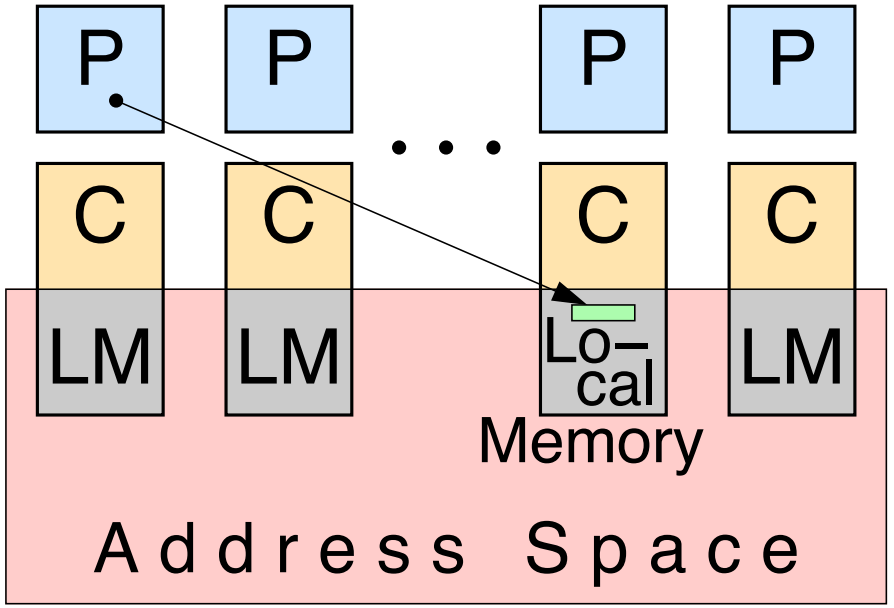
- Merging Implicit and Explicit Communication Support
  - Implicit Communication: through coherent caches
  - Explicit Communication: thru local “scratchpad” memories & RDMA
  - Configurable cache/scratchpad memory

- Virtualized, User-Level Remote DMA (RDMA) Interface
- Implementation: Merged Cache Controller &Network Interface
- Latency
- Conclusions

Implicit versus Explicit Communication



(a) Implicit



(b) Explicit

Address supplied by software does not specify physical locat'n-place'nt

⇒Unable to optimize in software, hence simpler to program…

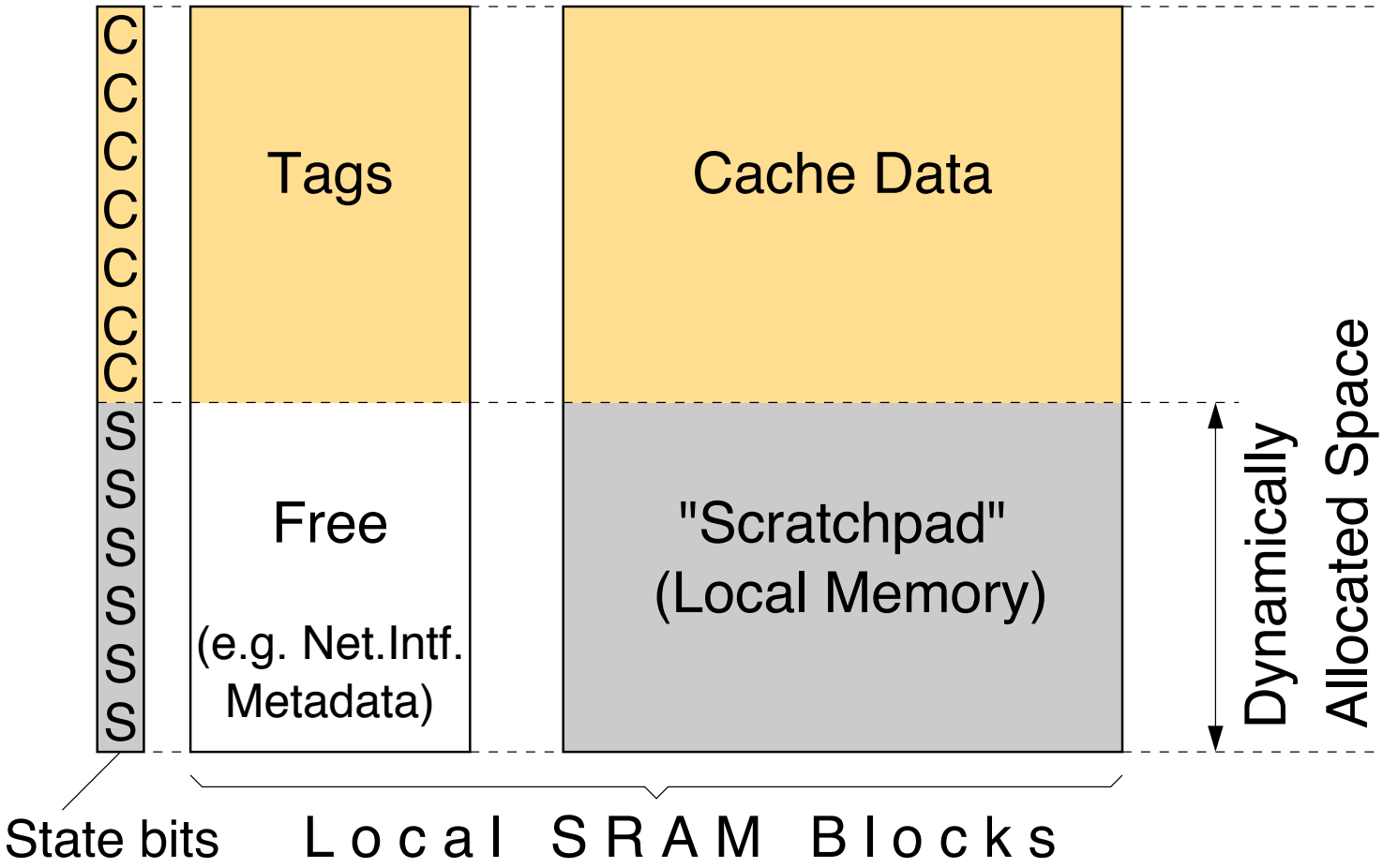
Locality mgt by simplistic/expensive HW

Address can indicate phys. place'nt in local "scratchpad" memories

⇒Software/runtime able to optimize in sophisticated ways whenever possible

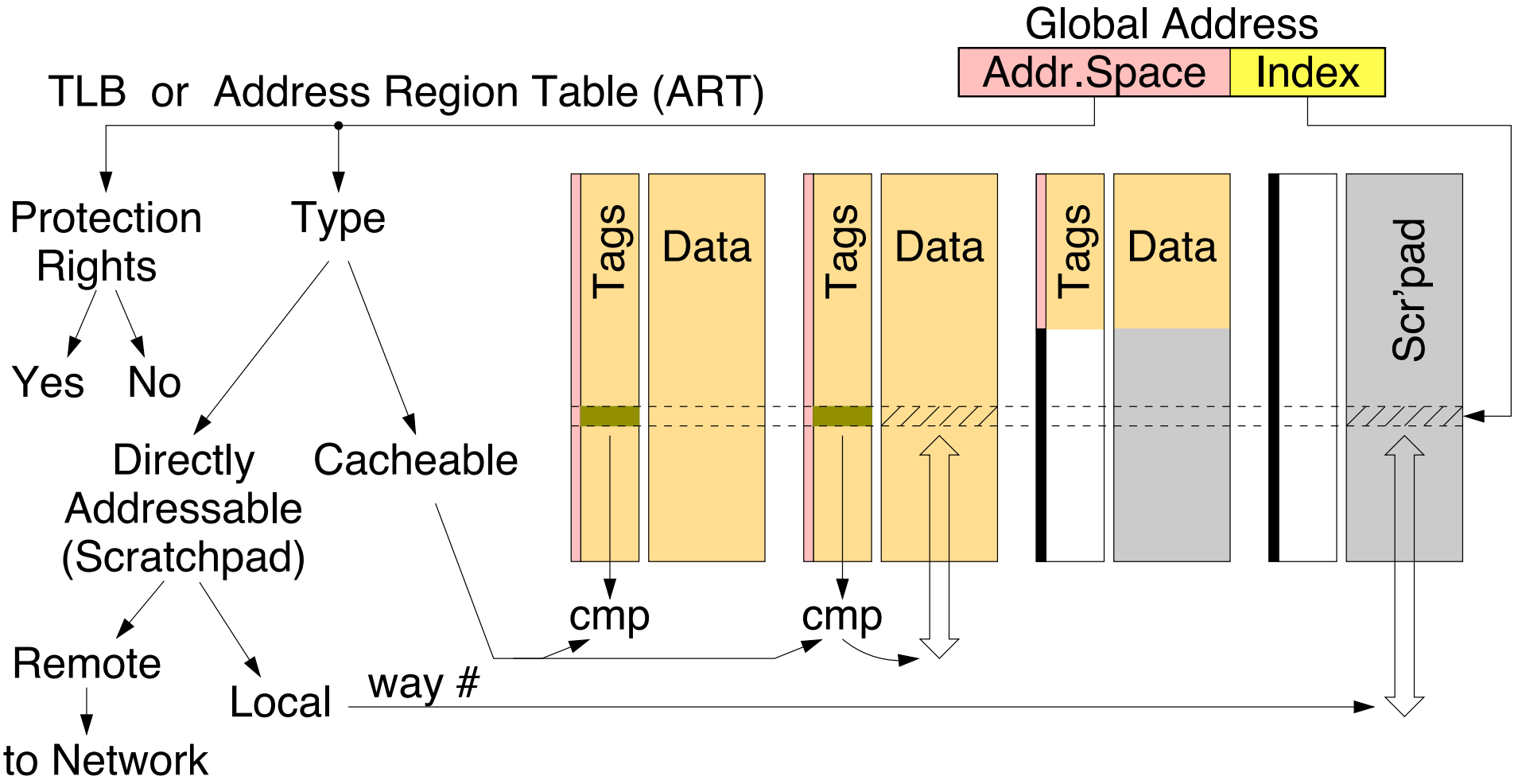
… want both – have both!

Configurable Local SRAM: Cache + Scratchpad



- Runtime configurable spaces: adapt to application requirements

Memory Access Flow



- Store to remote address (with write-combining buffer):
  like RDMA write – fast message send

Outline

- Merging Implicit and Explicit Communication Support

- Virtualized, User-Level Remote DMA (RDMA) Interface
  - Low-latency RDMA initiation
  - Event-Response mechanism for Synchronization support:
    Counters, Queues, and Notifications

- Implementation: Merged Cache Controller &Network Interface
- Latency
- Conclusions

Low-Latency RDMA: the NI as a first-class citizen

- How are Data Communicated?
  - Caches: implicitely, through coherence events, misses, or prefetching
  - Scratchpad: explicitely, through SW-initiated *Remote DMA* (RDMA)
- DMA has been in the domain of *traditional*, *slow* I/O devices
  - Communication, I/O, Networks were slower than the memory-bus
  - DMA controllers were on the *(slow) I/O bus*, far from the processor
  - *Single-set* of DMA-engine *control registers*:

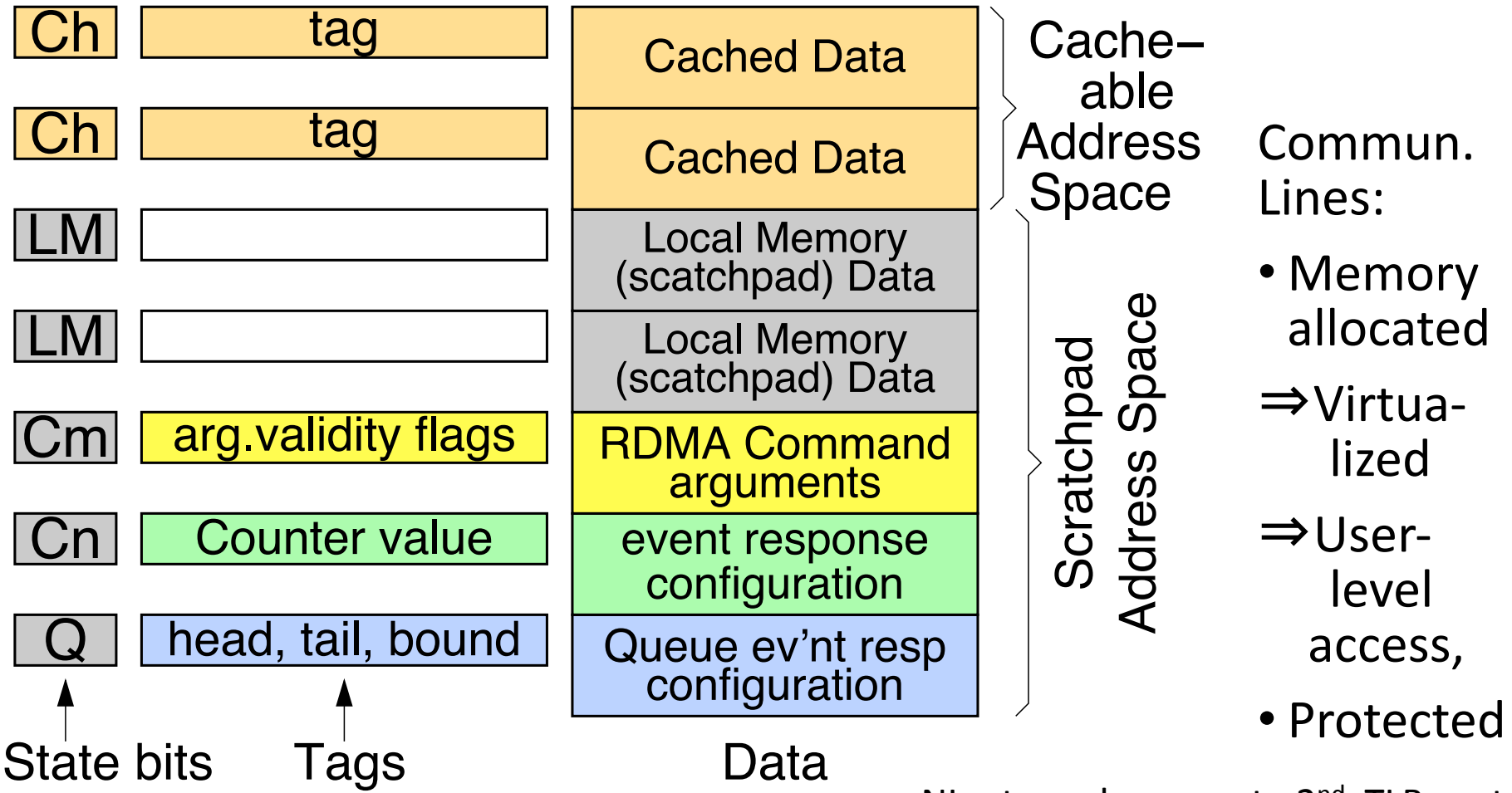
    ⇒DMA engine *shared under OS control* – huge cost of system call

    ⇒no way to issue time-interleaved DMA transfers
- *Fix it!:*
  - Communication registers close to the processor – at cache level
  - Multiple sets of commun. registers, shared/protected like memory

    ⇒Communication Argument Registers inside the Scratchpad

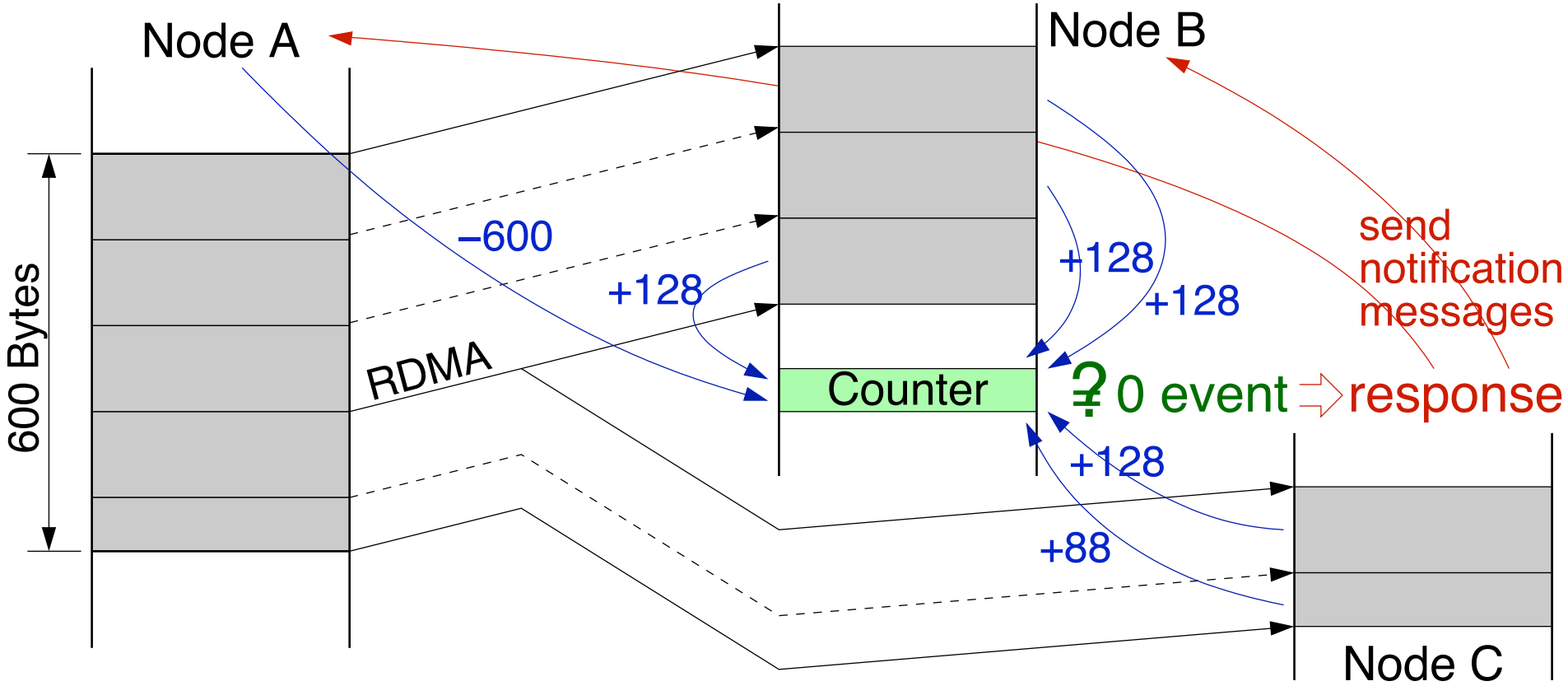
Line Types: Cache, Scratchpad, Communication

- "Special" Communication Lines are equivalent to I/O device Control Registers (memory-mapped locations with side-effects)



- NIout needs access to 2nd TLB port

Event-Response Example: RDMA Completion Notification



- Write packets (incl. remote stores) generate ack
- Works with multiple and split RDMA, w. multipath routing, w. store fences
- Counters also used for Barrier Synchronization
- Queues used for: Request Multiplexing; Task Scheduling; Locks

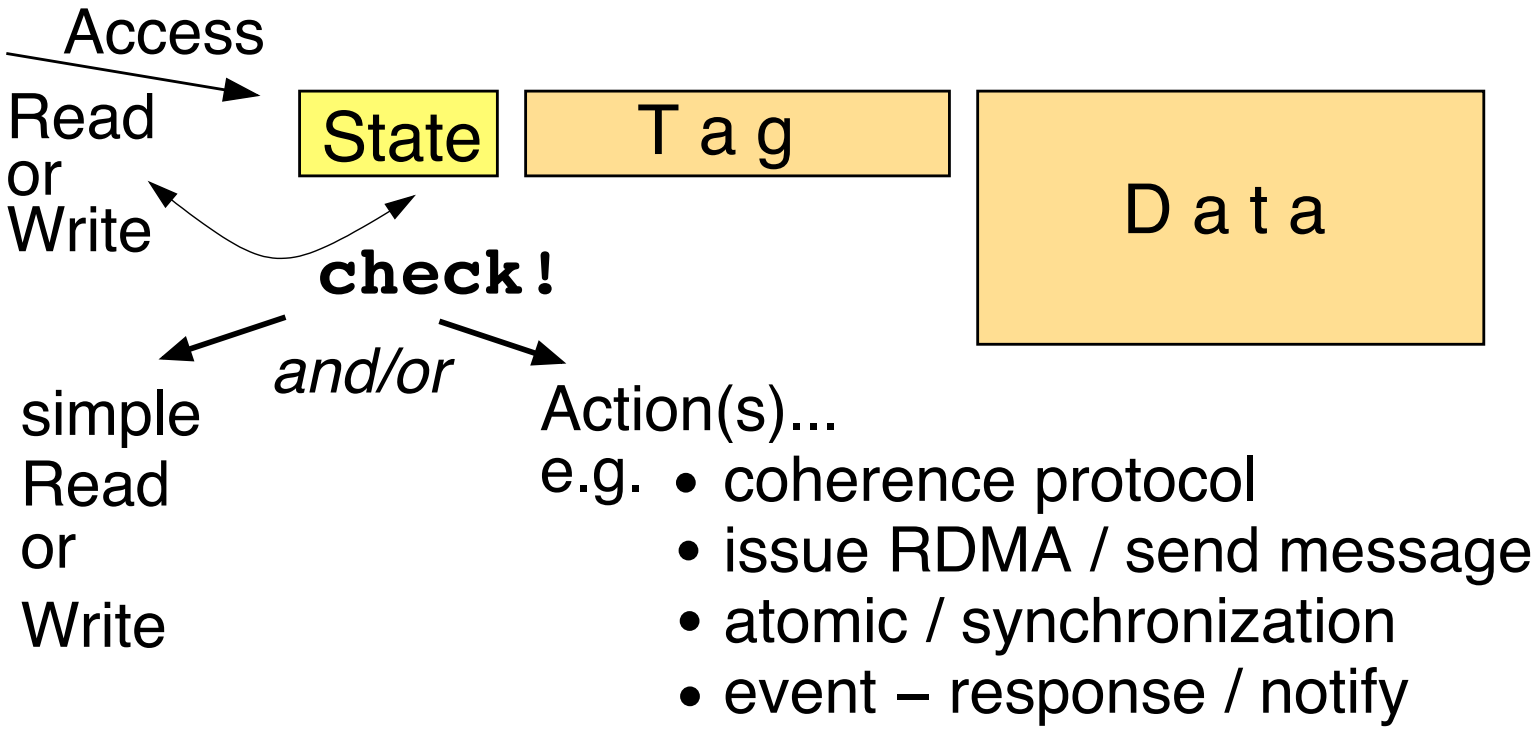
Outline

- Merging Implicit and Explicit Communication Support
- Virtualized, User-Level Remote DMA (RDMA) Interface

- Implementation: Merged Cache Controller & Network Interface
  - Common underlying hardware
  - Block diagram
  - FPGA prototype and gate counts

- Latency
- Conclusions

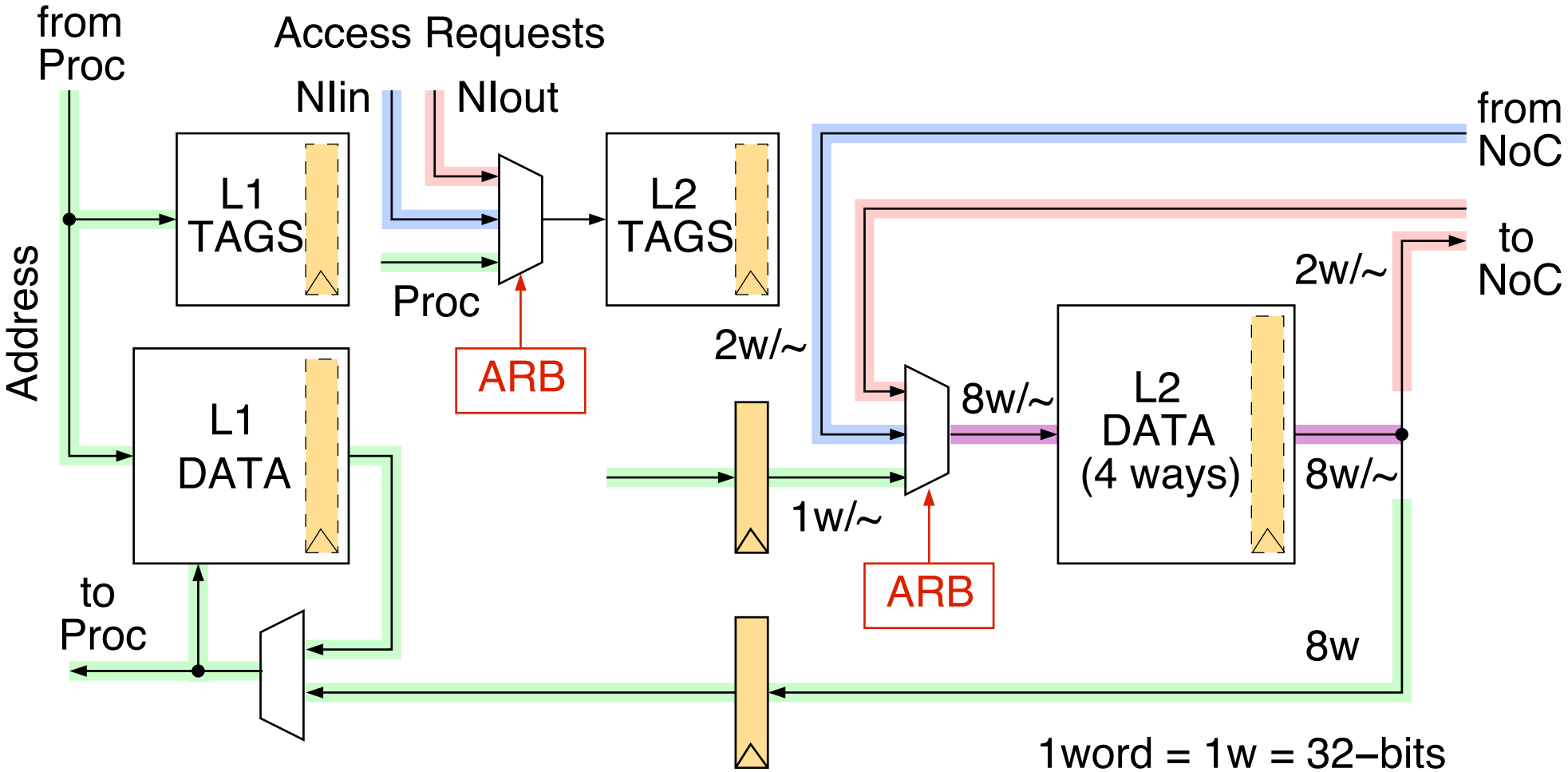
Caches, RDMA, Synchronization: Common Underlying HW



- Cache write-back's are like Remote Write DMA's
- Cache misses are like Remote Read DMA's
- Control uses event-responses as common underlying HW mech'sm

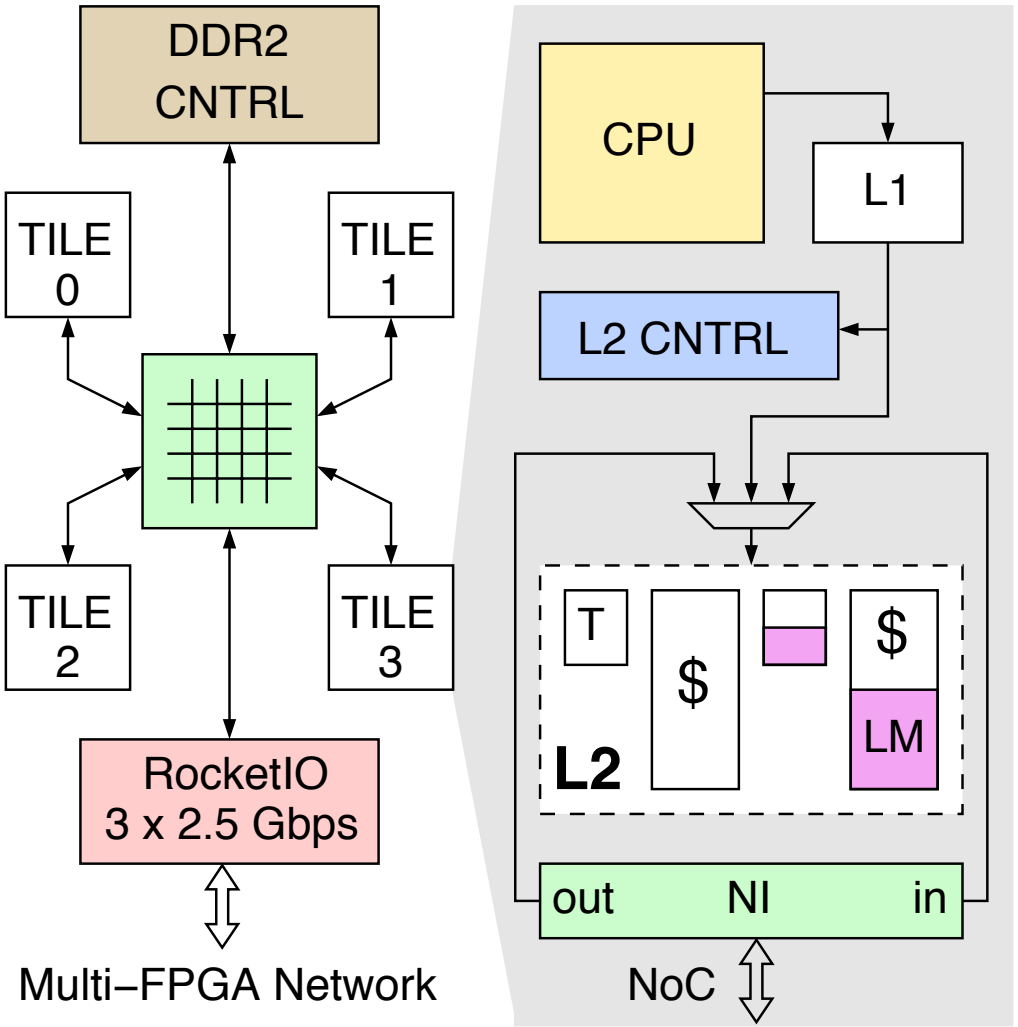
⇒Merge traditional cache controller & network interface functions

Block Diagram: Clock Cycles, Paths, Widths, Arbitration



- L1 cache is usually too small and too tightly integrated into the processor for scratchpad & RDMA to be added to it ➔ use L2

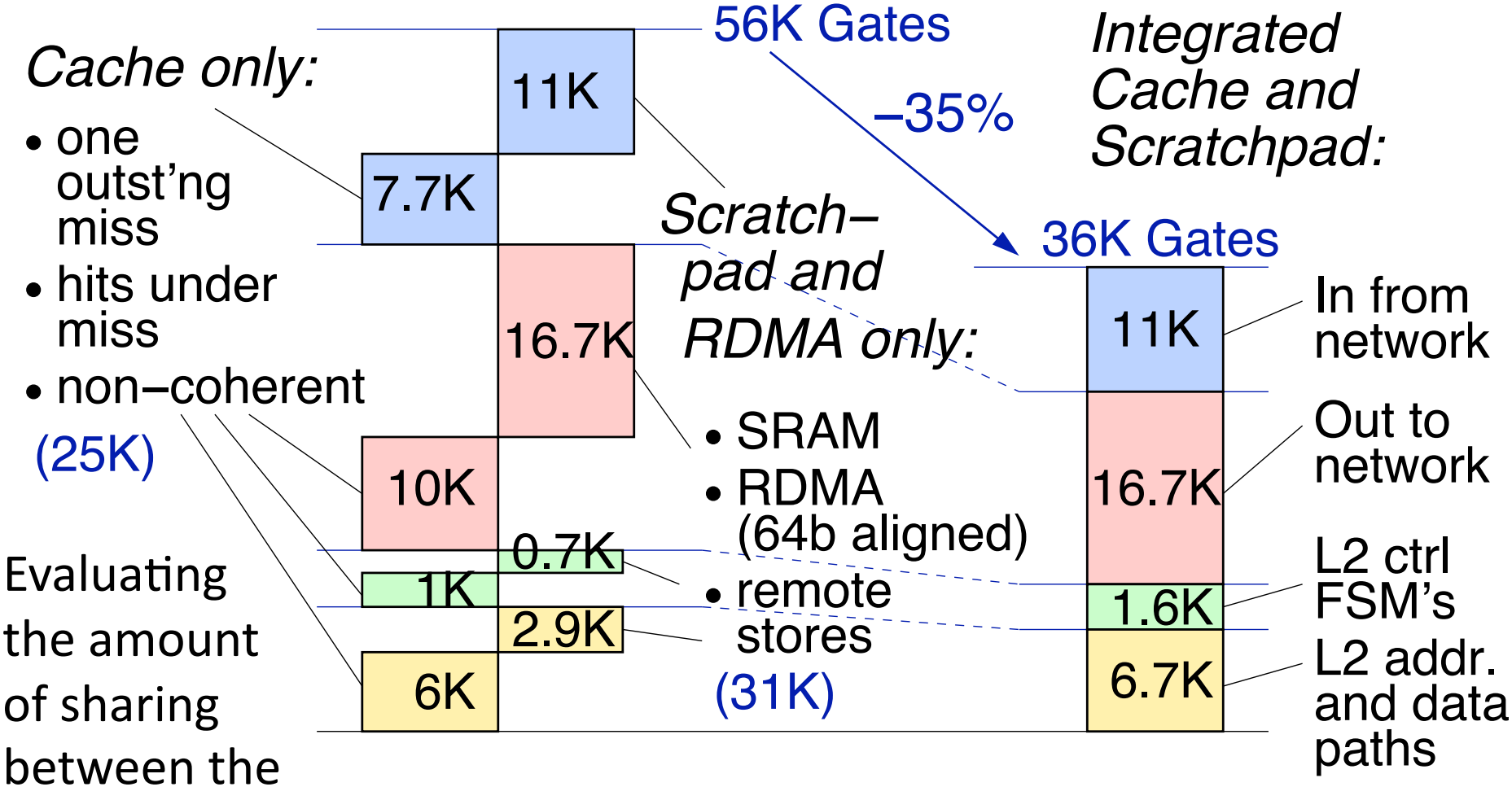
Prototype: 4 nodes per FPGA



- Xilinx Virtex-5
- Four 32-bit in-order MicroBlaze proc's
- 64-bit Crossbar NoC
- 256 Mbyte, off-chip DDR SDRAM
- Per Processor:
  - 4 KB write-thru L1
  - 64 KB write-back L2, L1-inclusive, hits under miss
  - 100 MHz
  - No coherence yet
  - No Addr. Transl. yet

- Read hit: 1 clock (L1), or 4 clocks (L2)
- Write hits: *Pipelined*, 1 w/ck, both L1 and L2

Gates Cost: Integrated versus Separate Cache - Scratchpad



- Evaluating the amount of sharing between the cache and the scratch-pad / RDMA

Gates, excluding SRAM arrays and L1 Cache, assuming: 1FF ~= 1LUT ~= 8 gates

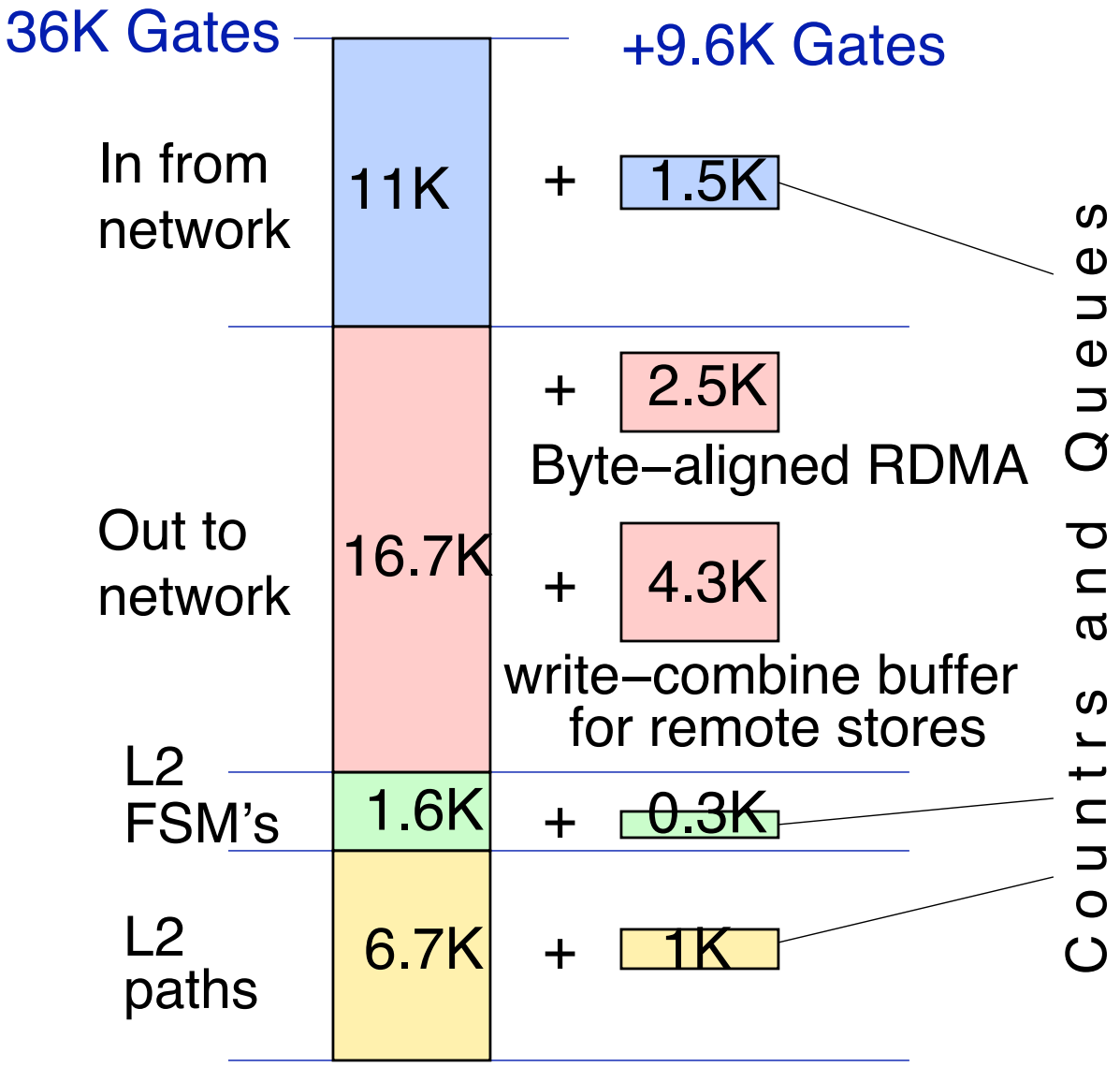
- Version 2 design – newer than in paper (optimized)

Extra features Cost

Left side is baseline:

Integrated Cache and Scratchpad, with:

- SRAM blocks
- 64-bit-aligned RDMA
- remote stores



Gates, excluding SRAM arrays & L1 Cache
assuming: 1FF ~= 1LUT ~= 8 gates

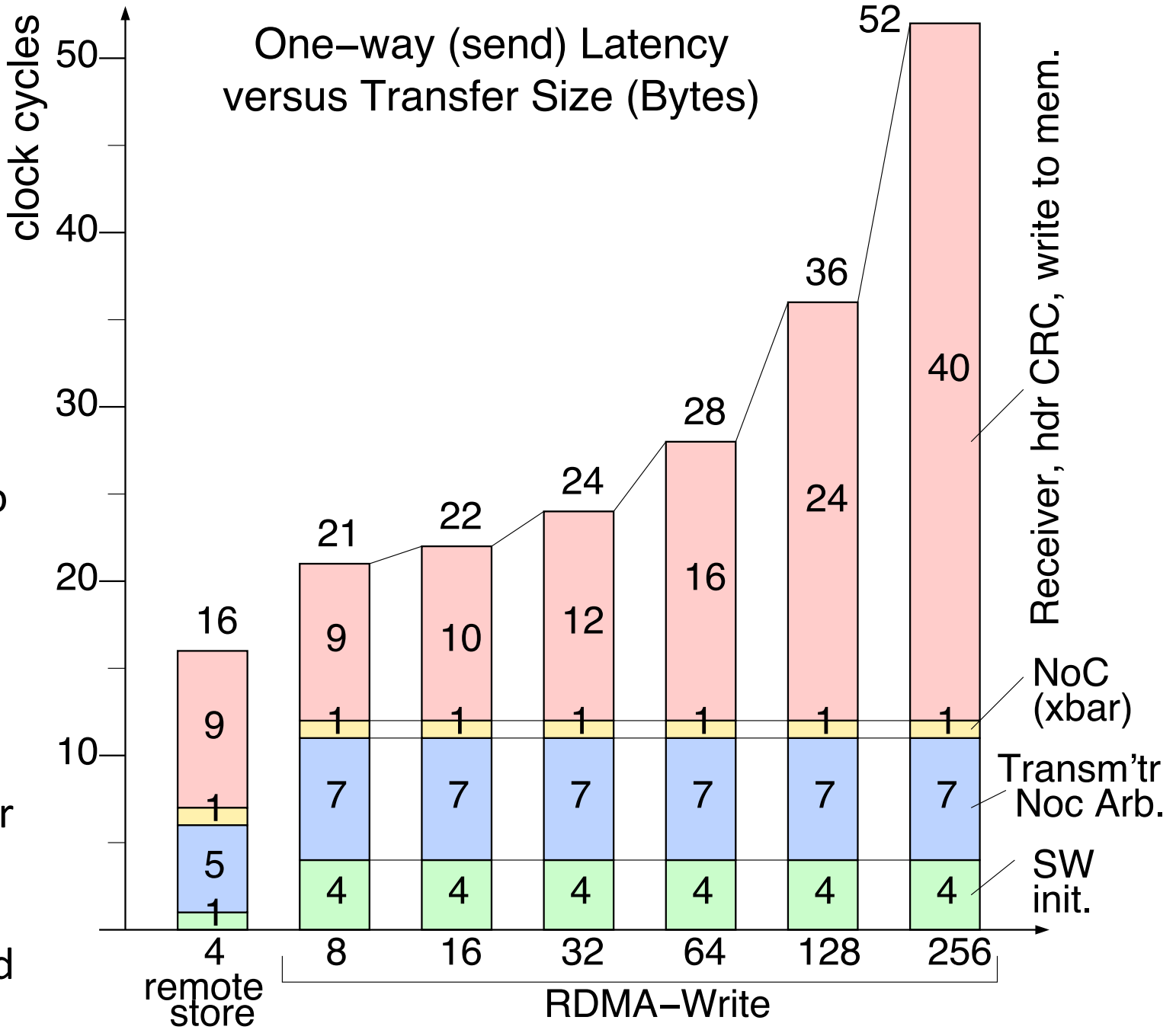
Outline

- Merging Implicit and Explicit Communication Support
- Virtualized, User-Level Remote DMA (RDMA) Interface
- Implementation: Merged Cache Controller & Network Interface

- Latency
- Conclusions

Latency under light load

- RDMA initiated by four stores; back2back
- Scr'tchpd also used for xbar input queues
- Cut-thru crossbar
- Receiver first checks header CRC, then starts writing in cut-thru md

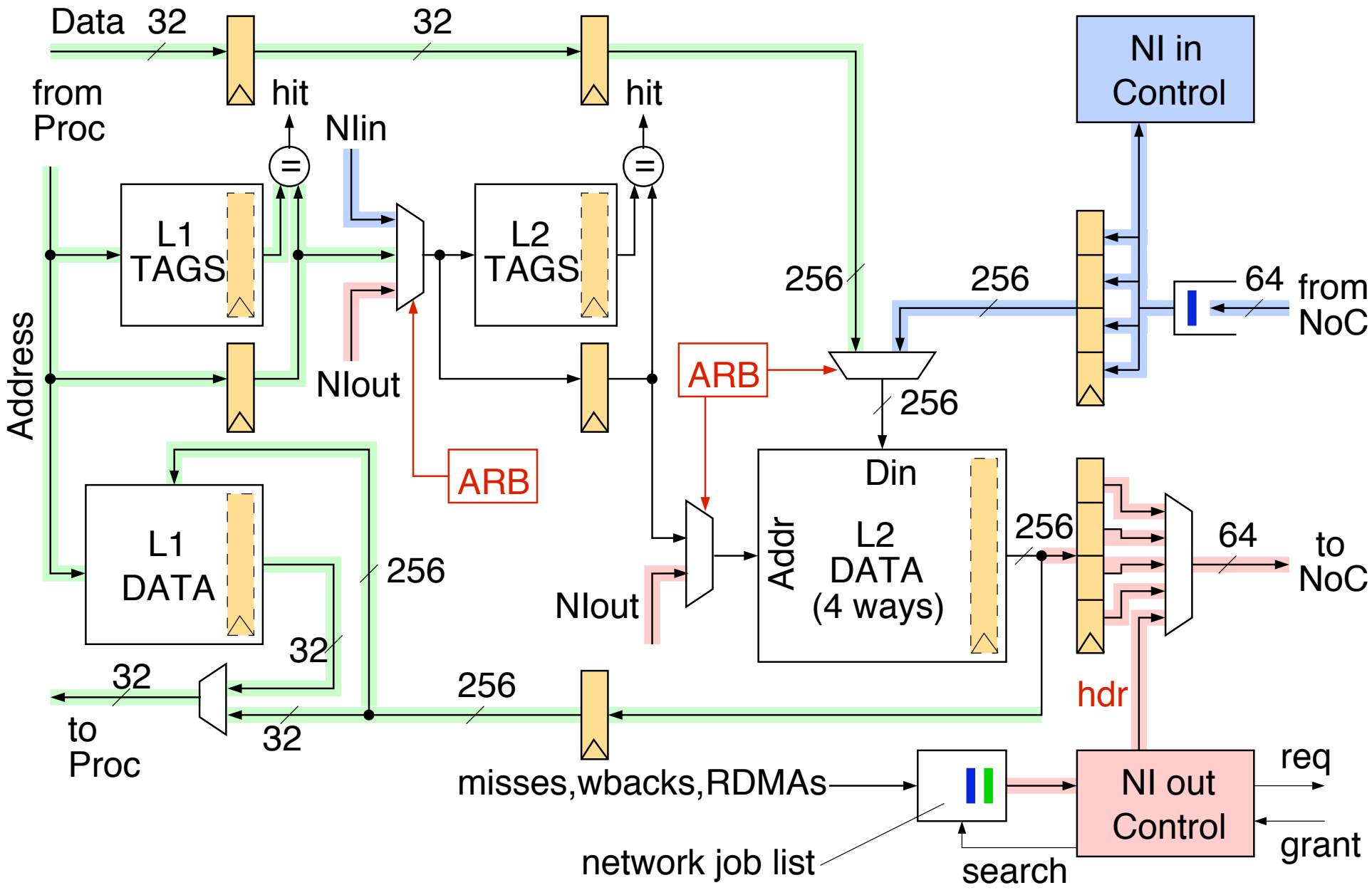


Conclusions:

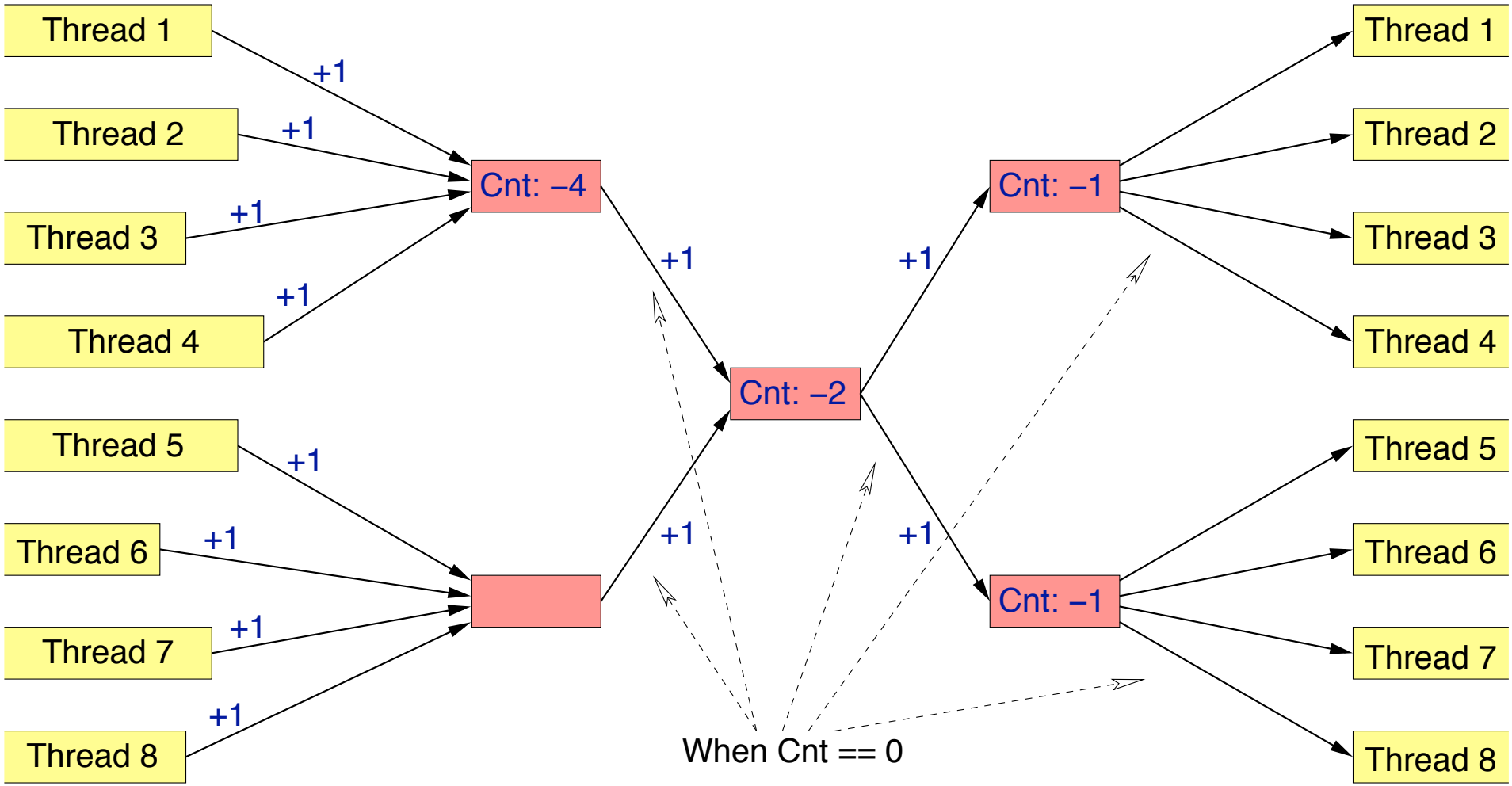
- Explicit Communication (scratchpad, RDMA)
  - whenever software knows how to orchestrate the data transfers
- Implicit Communication (coherent caches)
  - otherwise, for ease of use
- Low-latency RDMA
  - control “registers” in scratchpad: virtualized, user-level, protected
- Event – Response mechanism
  - unify cache / scratchpad
  - synchronization support: counters, queues, notifications
- FPGA Prototype
  - 35 % sharing when cache controller merged with RDMA controller
  - 20 clock cycle one-way small message latency

Backup Slides

Detailed Datapath

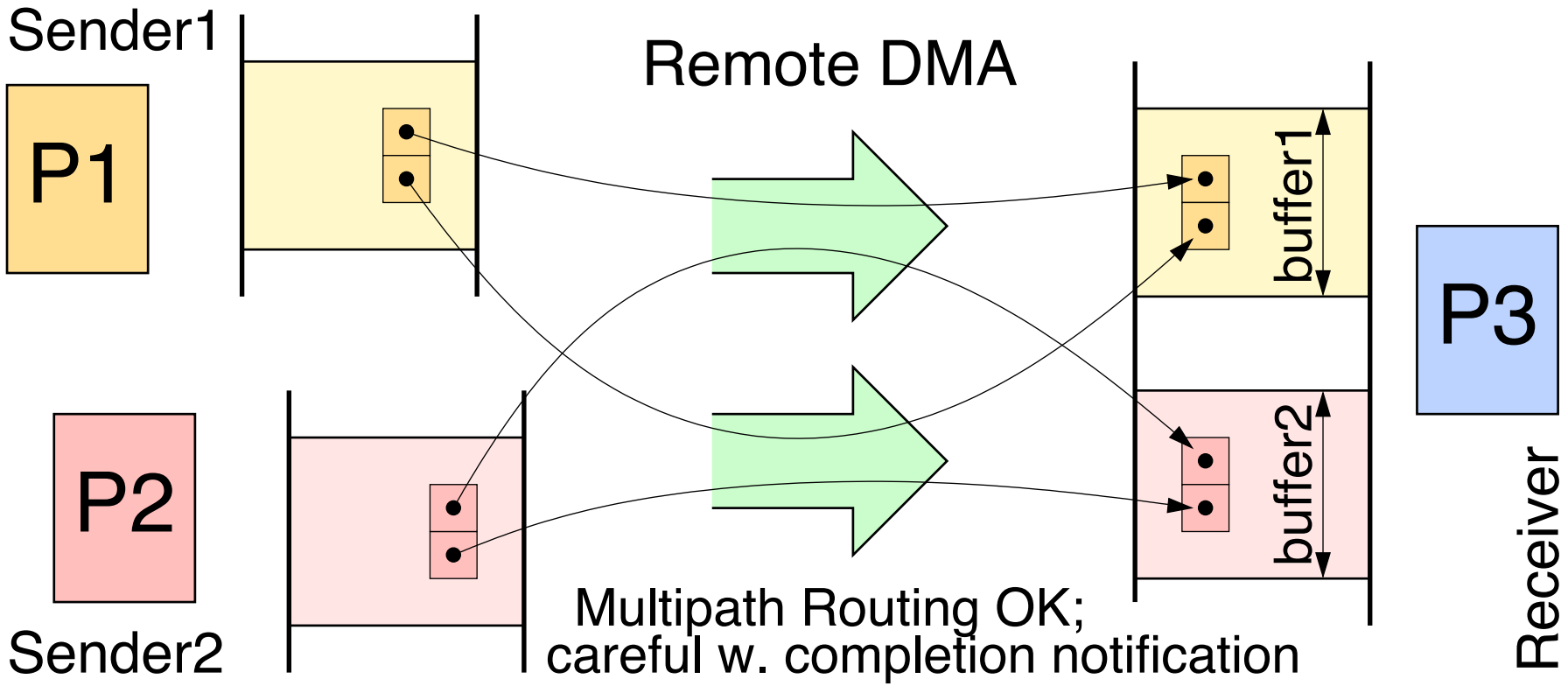


Barrier Synchronization using Counters



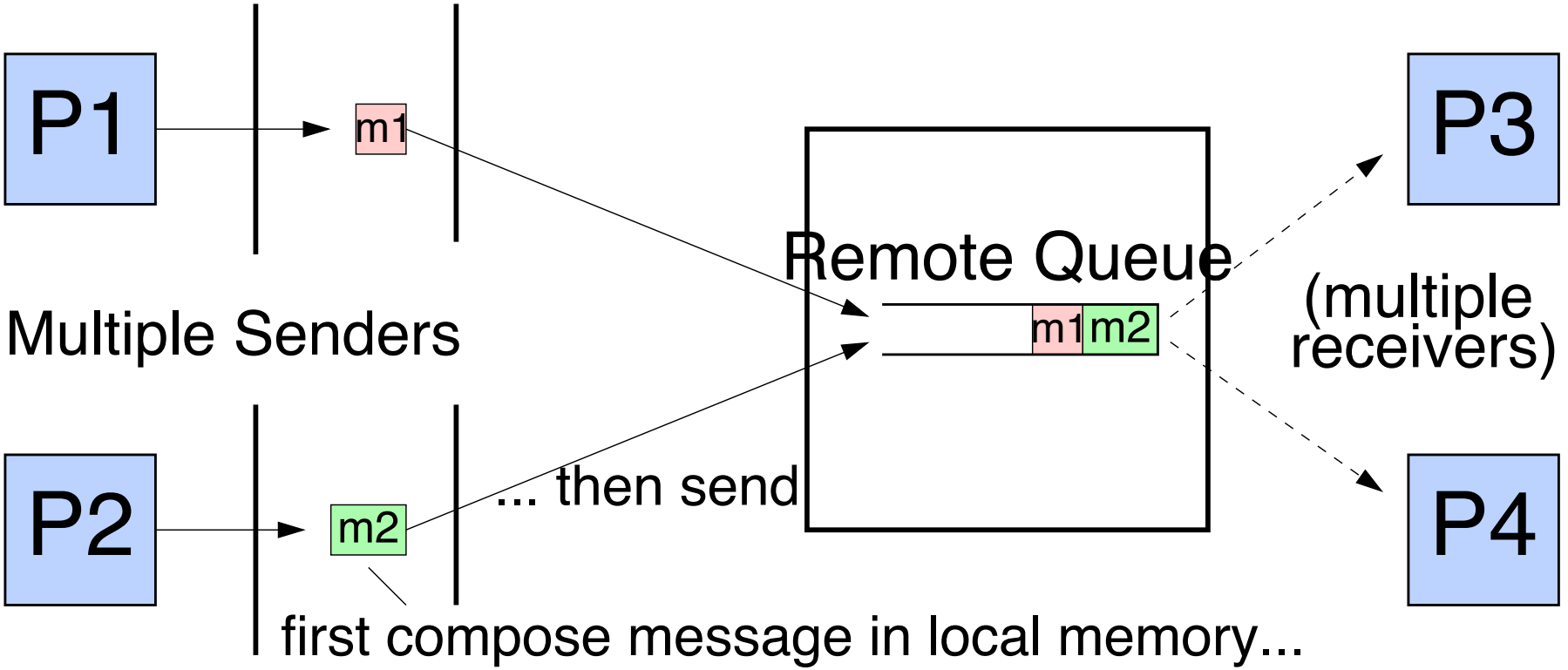
- Each counter can be configured with up to 4 notification addresses

Remote DMA: in-place Data Delivery



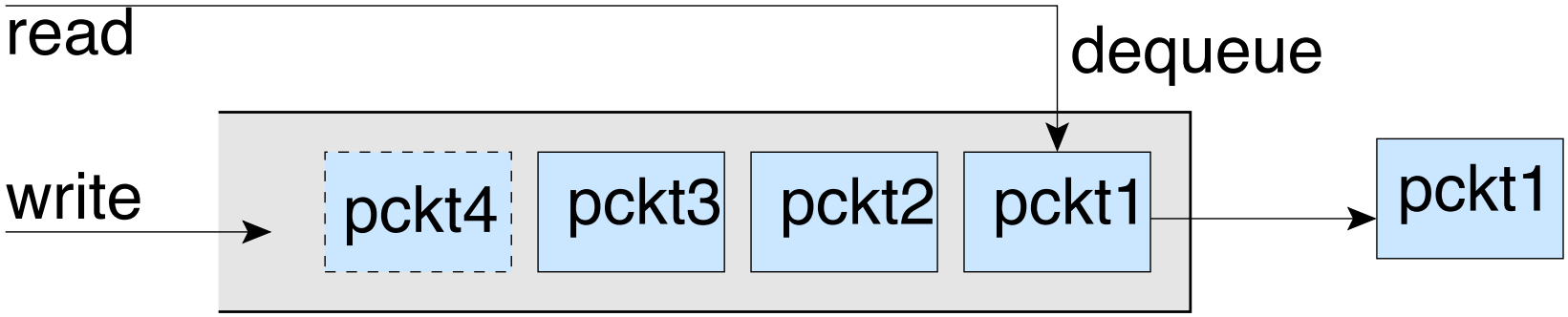
- Global (shared) Address Space
- Allows zero-copy communication, adaptive (multipath) routing
- Requires buffer space allocation per producer-consumer pair

Remote Queues: Multi-Party Synchronization

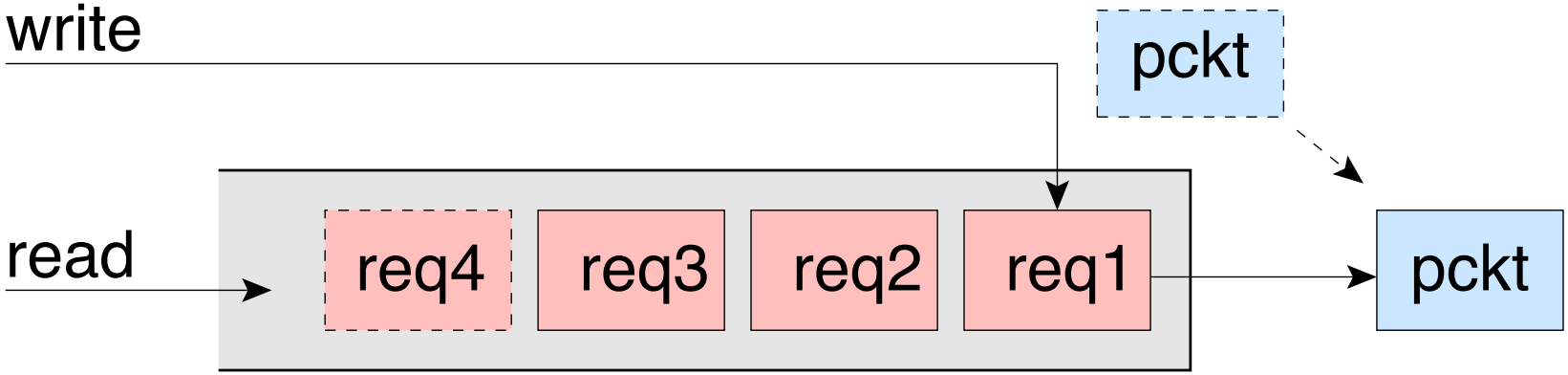


- Remote Queues differ from RDMA as follows:
  - receive buffer space shared among many senders
  - speeds up polling of multiple receive channels
- Atomicity of multiplexing/demultiplexing: Synchronization Primitive

Multiple-Reader Queues



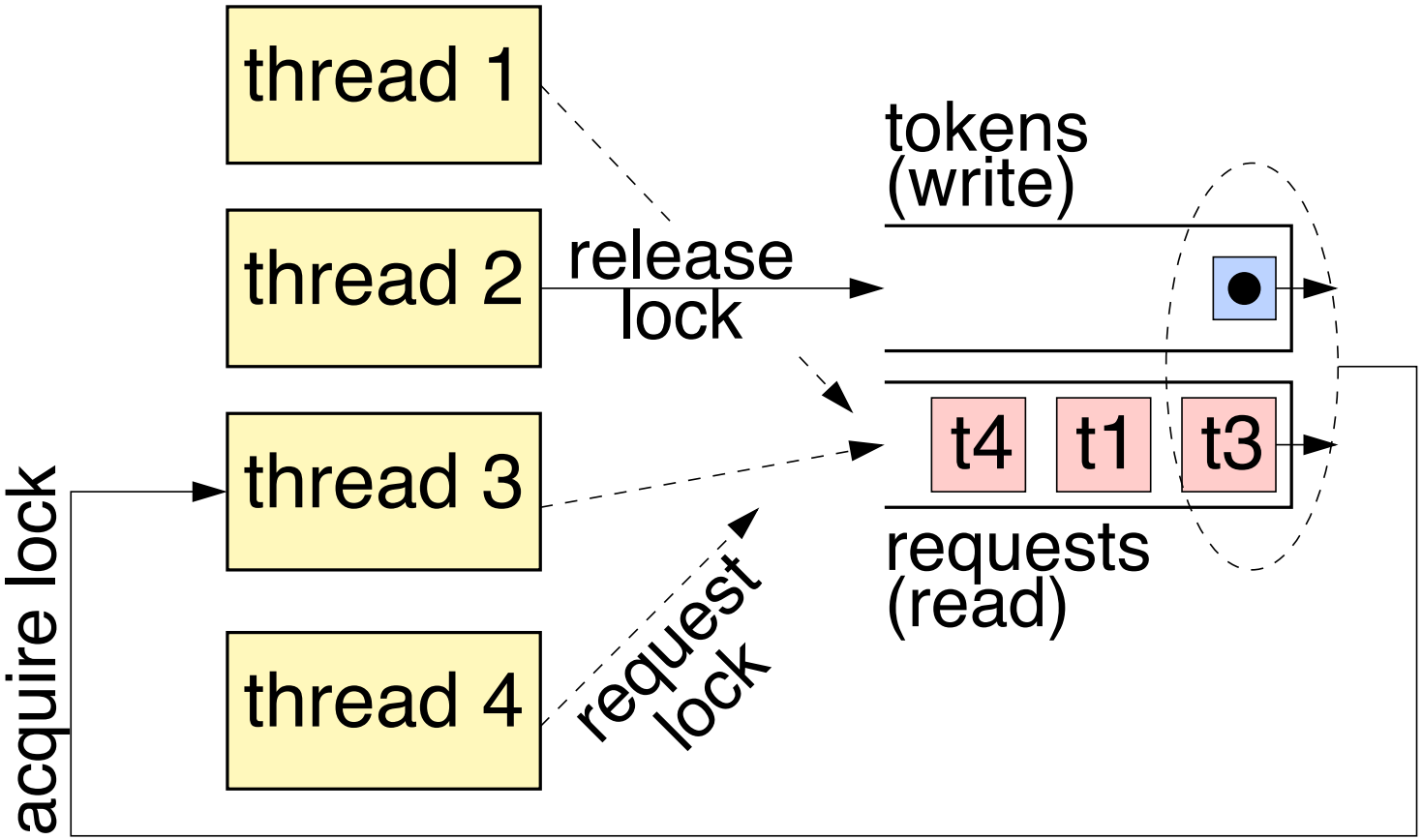
(a) data arrive before read requests



(b) data arrive after read requests

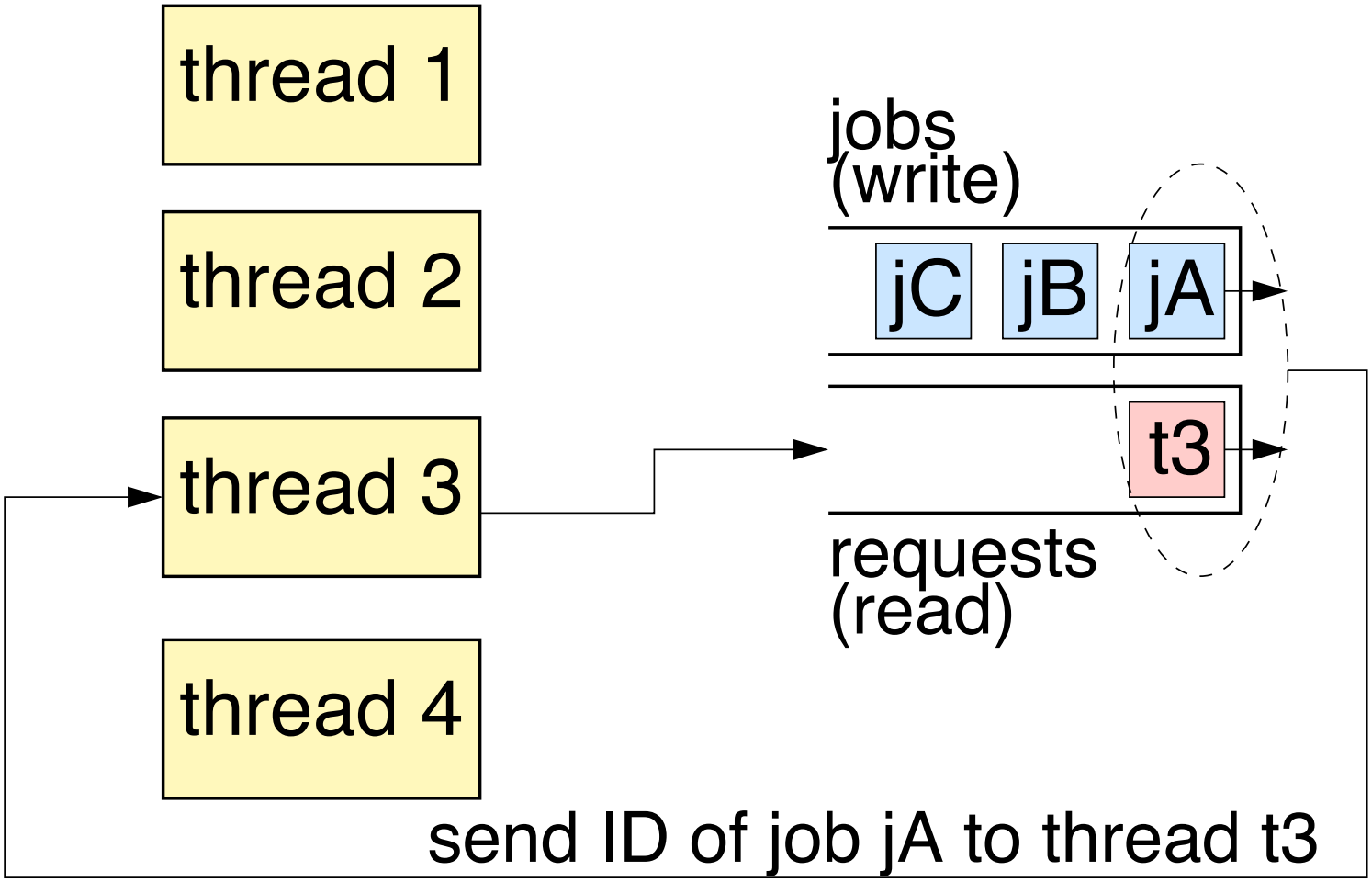
- Queue provides *pairwise matching* between write and read packets, *independent of arrival order*

Pairwise Match Application 1: Traditional Locks



- Generalization: multi-token locks (semaphores – place an upper bound on the amount of parallelism)

Application 2: Task Queues – Remote Job Dispatching



- (Remotely) Read from the job (task) queue, to get the next job ID