Cluster Communication Latency:

towards approaching its Minimum Hardware Limits, on Low-Power Platforms

Manolis Katevenis

FORTH, Heraklion, Crete, Greece (in collab. with Univ. of Crete)

http://www.ics.forth.gr/carv/

FORTH-ICS
COMPUTER ARCHITECTURE &
VLSI SYSTEMS LABORATORY

EURO
SERVER

ExaNeSt

ExaNoDe

Acknowledgements

- Iakovos Mavroidis
- John Goodacre (ARM, KALEAO)
- Vassilis Papaefstathiou
- Manolis Marazakis
- Nikolaos Chrysos
- *and many many more!...*

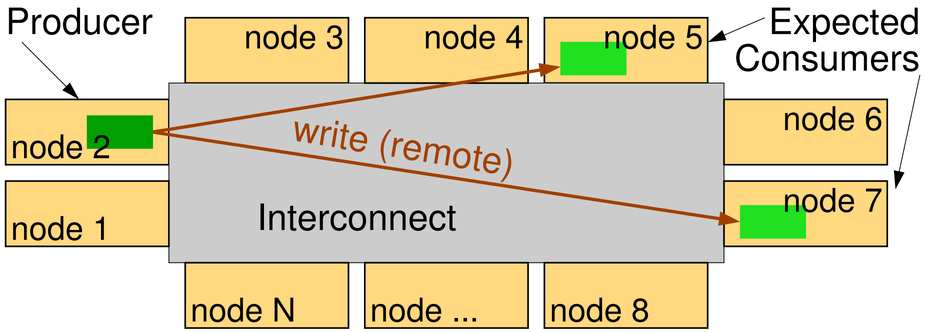
HiPEAC COMPILATION ARCHITECTURE

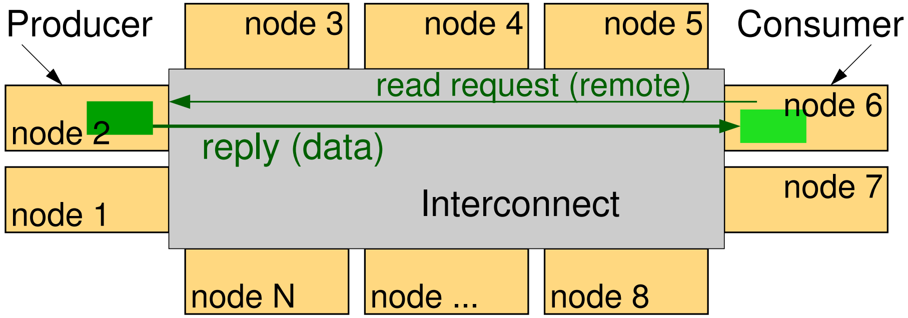
EURO SERVER

ExaNeSt

ExaNoDe

The Essence of Communication

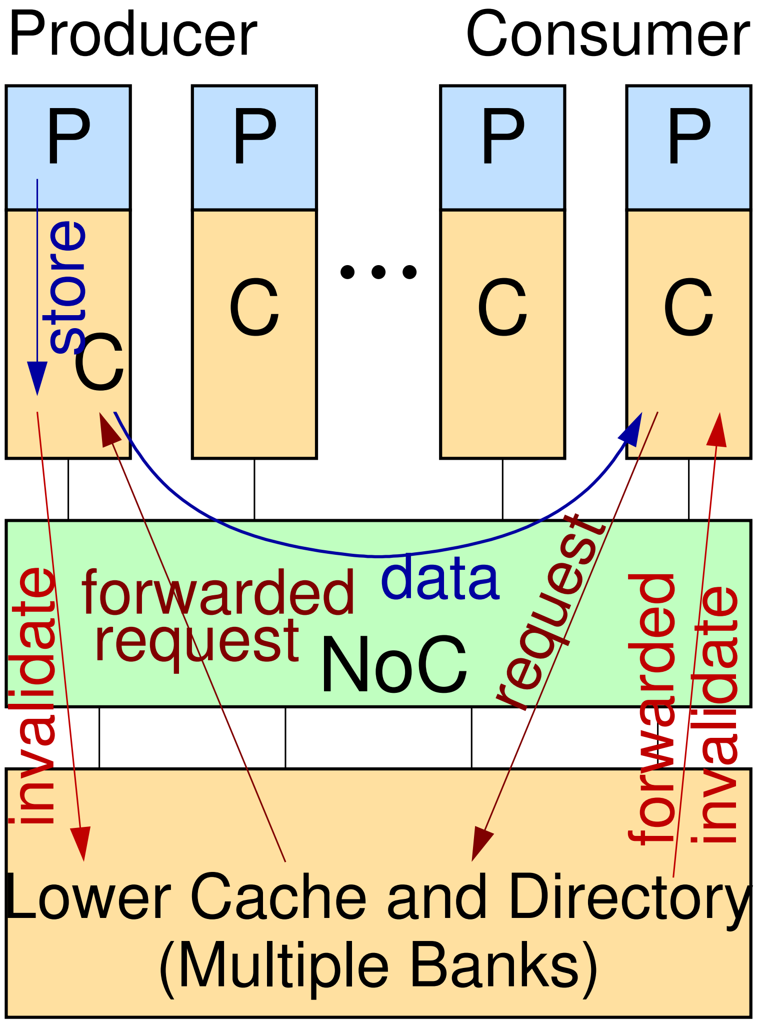


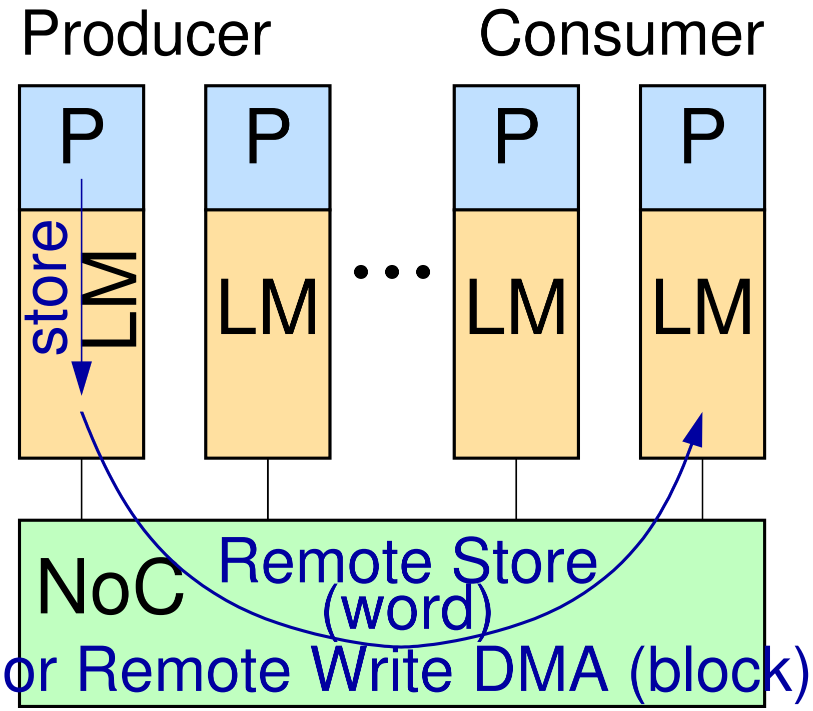


- *Push* style, like *email*
- If you know the consumers
  - else like spam
- If space already exists
- Zero latency, when pre-sent
- One-way latency, when sent "just-in-time"

- *Pull* style, like *web browser*
- When reader decides it needs the data and allocates space
- Two-way latency, starting when the data is needed
  - unless able to prefetch

Remote DMA versus Cache Coherence





Example – push style:
explicitely sending the data:

<u>5x fewer packets</u>

(⇒ less time, less energy)

Can we approach this ideal, in Cluster Communication?

- Cluster Computing / Datacenters / HPC:

  ⇒ many thousands of nodes

  "node" = island of (hardware) cache coherence

- Can we get Cluster Communication to be analogously fast like store & load instructions in a coherence island?

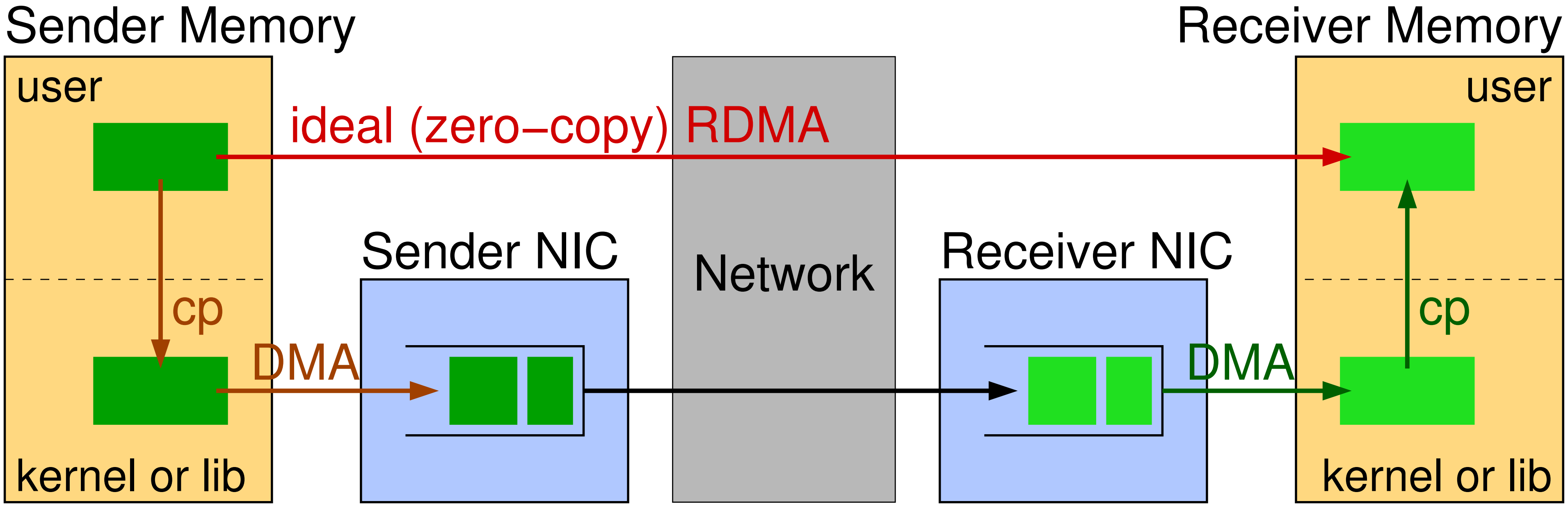
  + small delay due to distance, but not much more

- Clusters originate from mass, commodity market, where customers did not care about "a few extra μs"…

- Work done in Energy-Efficient Datacenter/HPC projects

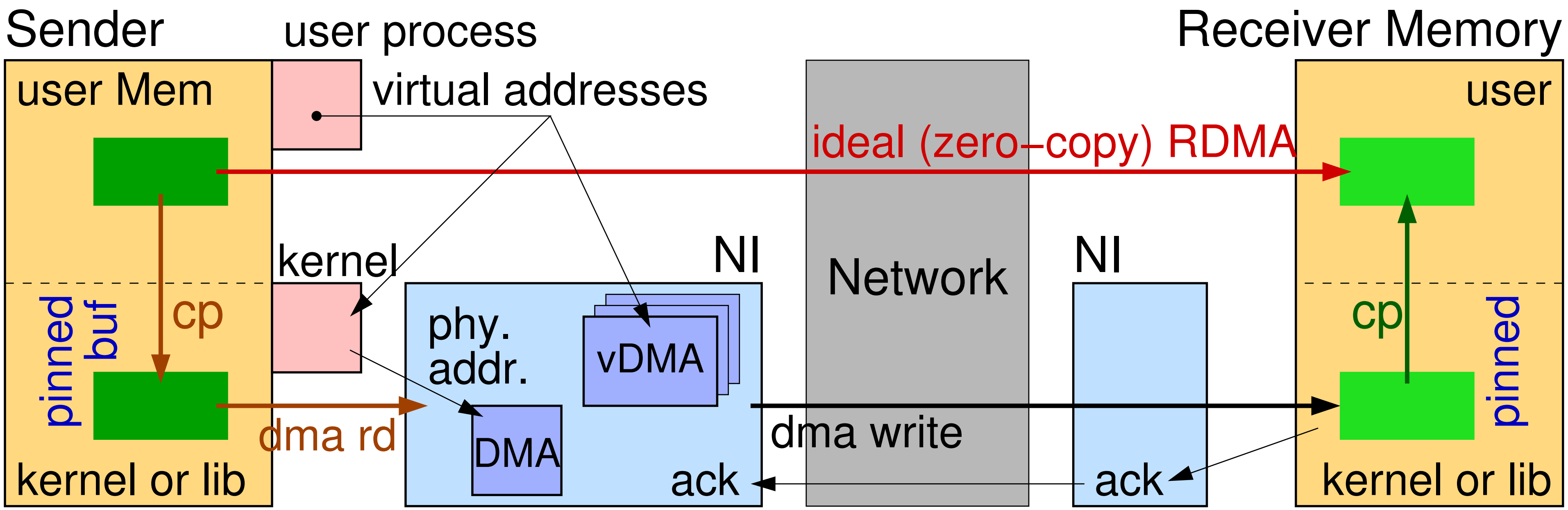
  ⇒ 64-bit ARM processors

Inefficiencies in Cluster Communication, to be overcomed



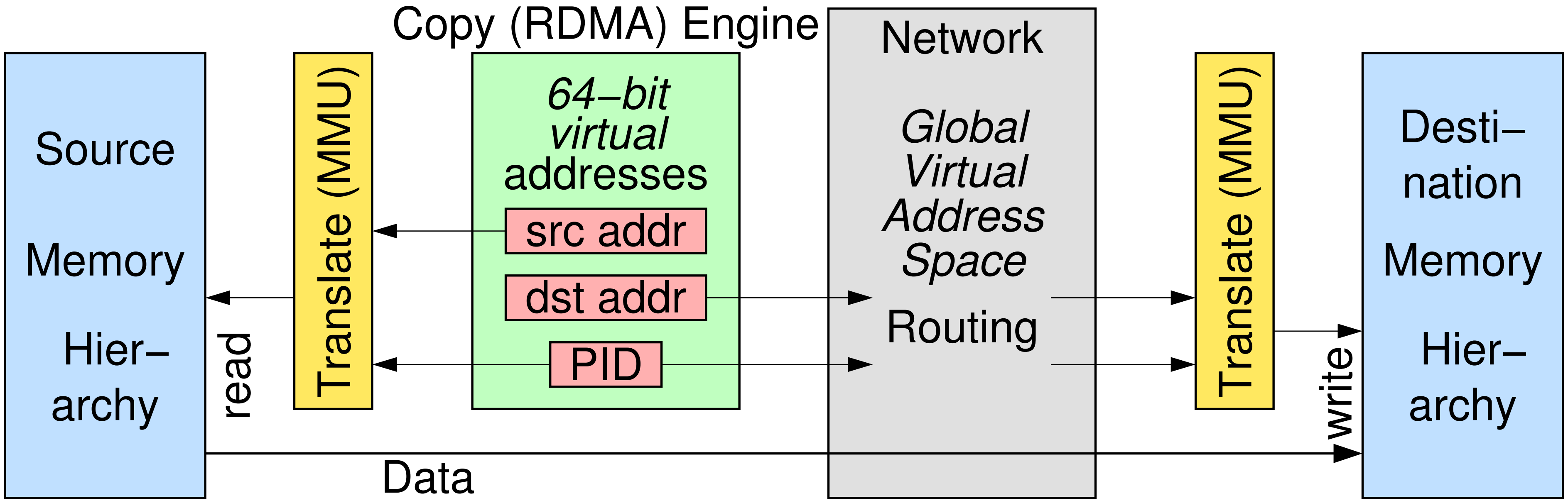
- Five (5) copyings of the data, instead of just one (1) !
  - Data copying consumes time and energy
- Start by eliminating (large) NIC buffers:
  - DMA across the network – RDMA

DMA Initiation: System Call, or directly from User-Level ?



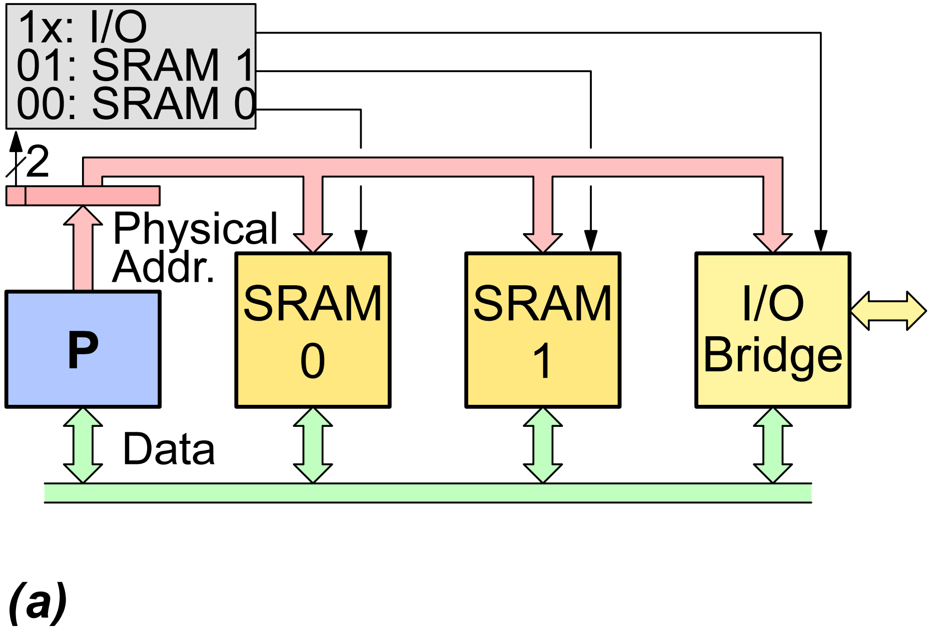
- System Call to initiate a DMA (with physical addr.) ≈ 3 to 4 µs
  - on 64-bit ARM in Xilinx Zynq UltraScale+ under Linux
- For the virtualized DMA engine (8 channels), that accepts virtual address arguments, Initiation (writing 3-4 registers) ≈ <0.5 µs
- Next Questions: Who translates addresses & where?
  Why copy User to/from Kernel/Library?

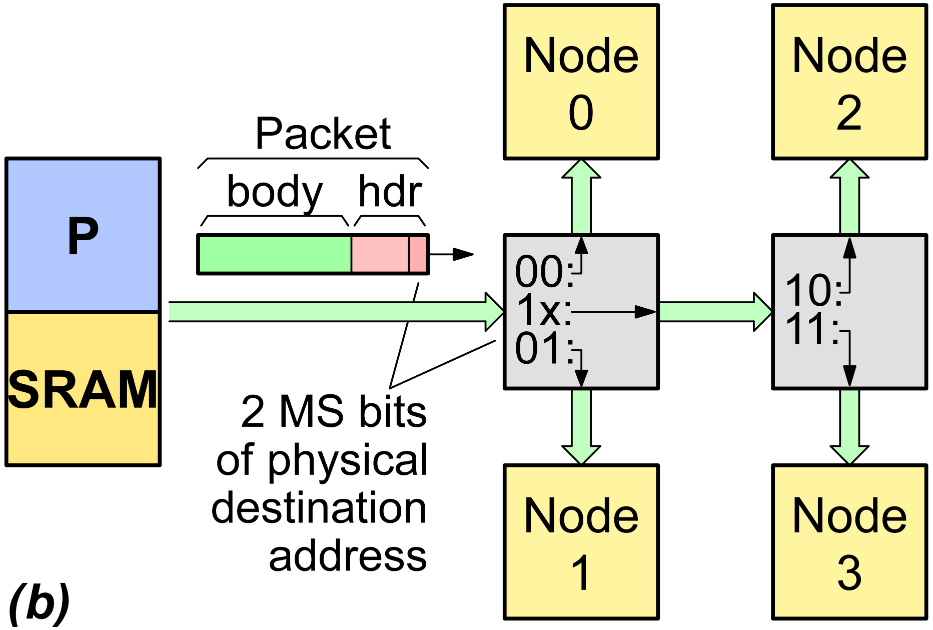
Scalability: Global Virtual Addresses & Progressive Translation



- for Scalability: not all nodes can be required to know all other nodes' translation tables –see "Progressive Address Translation" in the M. Katevenis paper in: *Stamatis Vassiliadis Symposium **2007***
  ⇒ Destination Addresses in network packets must be *Virtual*
- for Scalability: 64-bit GVA + (global) protection domain identifier

Two Slides from my *Stamatis Vassiliadis 2007* Symposium talk:
Network Routing as Generalization of Address Decoding



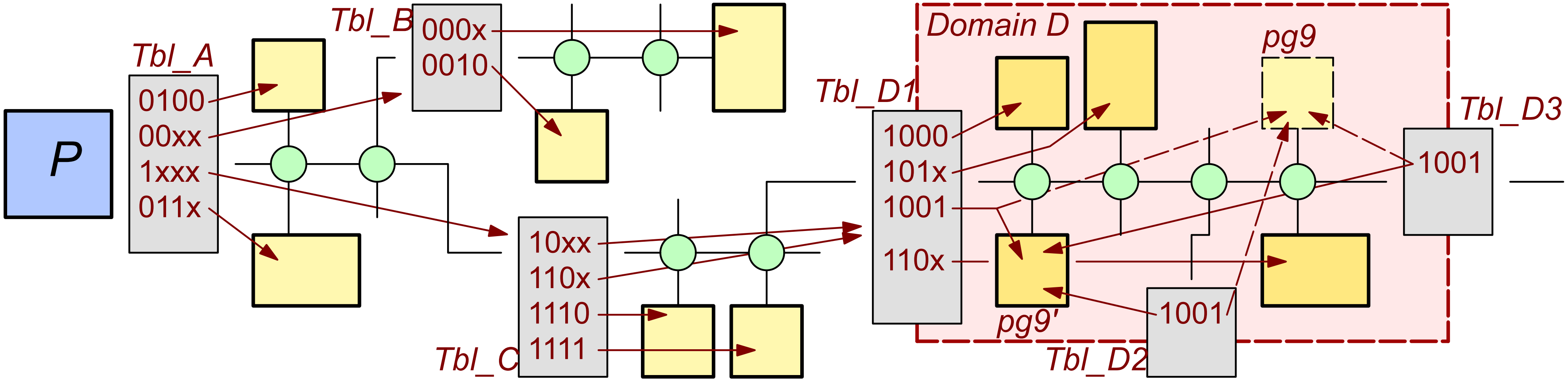


- Physical Address Decoding in a uniprocessor

- Geographical Address Routing in a multiprocessor

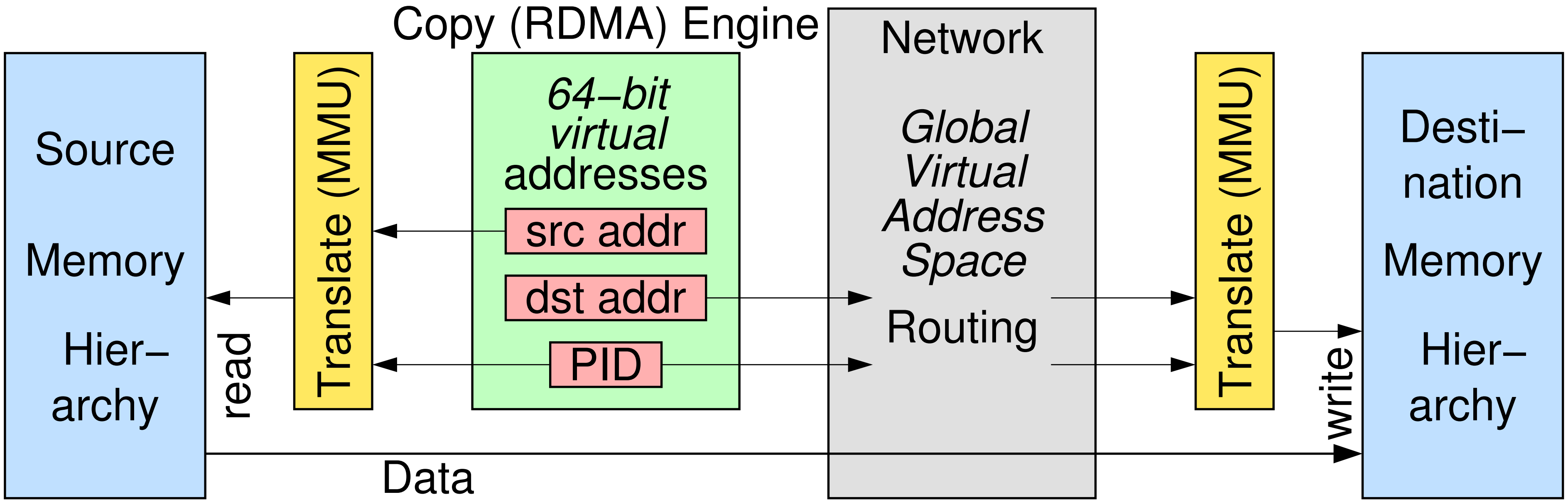
http://www.ics.forth.gr/carv/ipc/ldstgen_katevenis07.pdf

Progressive Address Translation: Localize Migration Updates



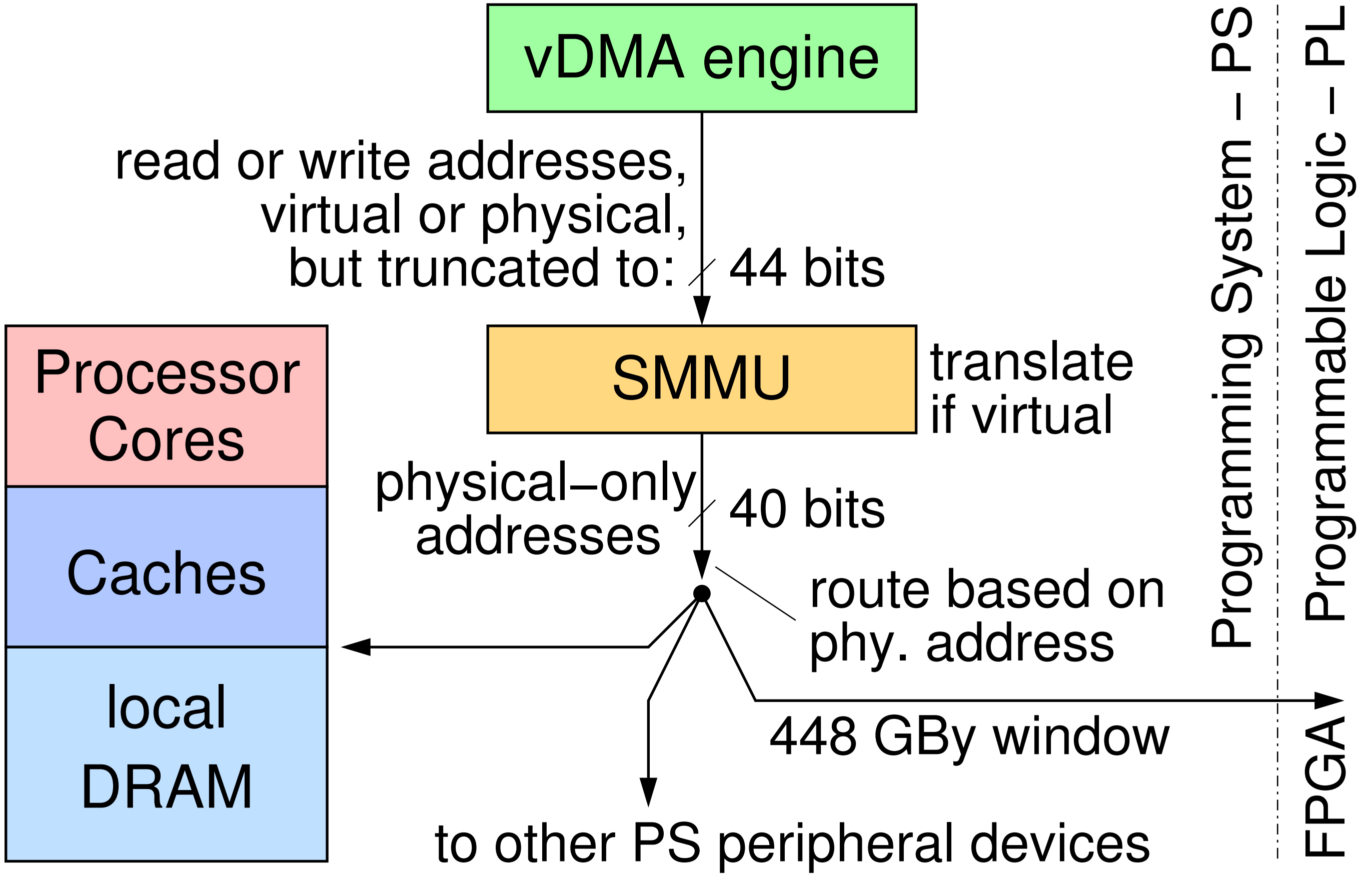
- Packets carry global virtual addresses
- Tables provide physical route (address) for the next few steps
- When page 9 migrates within D, only tables in that domain need updating
- Variable-size-page translation tables look like internet routing tables (longest-prefix matches if we want small-page-within-big-region migration)
- Tables that partition the system, for protection against untrusted operating systems, look like internet firewalls

Scalability: Global Virtual Addresses & Progressive Translation



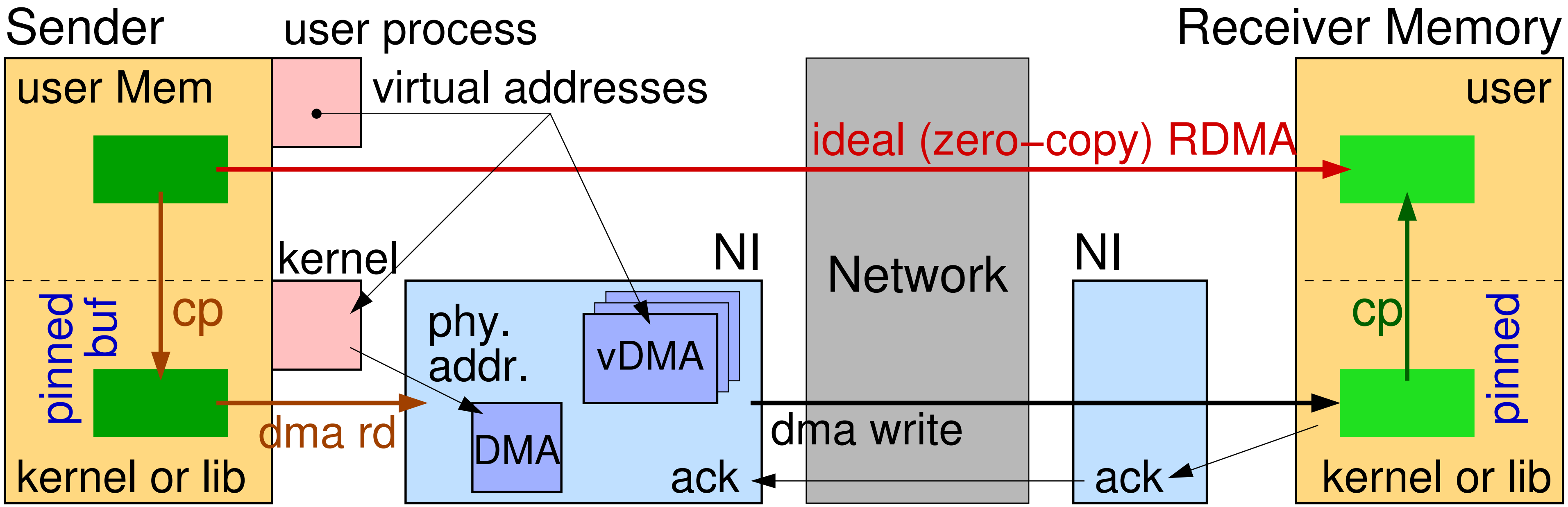
- for Scalability: not all nodes can be required to know all other nodes' translation tables –see "Progressive Address Translation" in the M. Katevenis paper in: *Stamatis Vassiliadis Symposium* ***2007***
  ⇒ Destination Addresses in network packets must be *Virtual*
- for Scalability: 64-bit GVA + (global) protection domain identifier

Current Address Translation Arch. not ready for GVAS



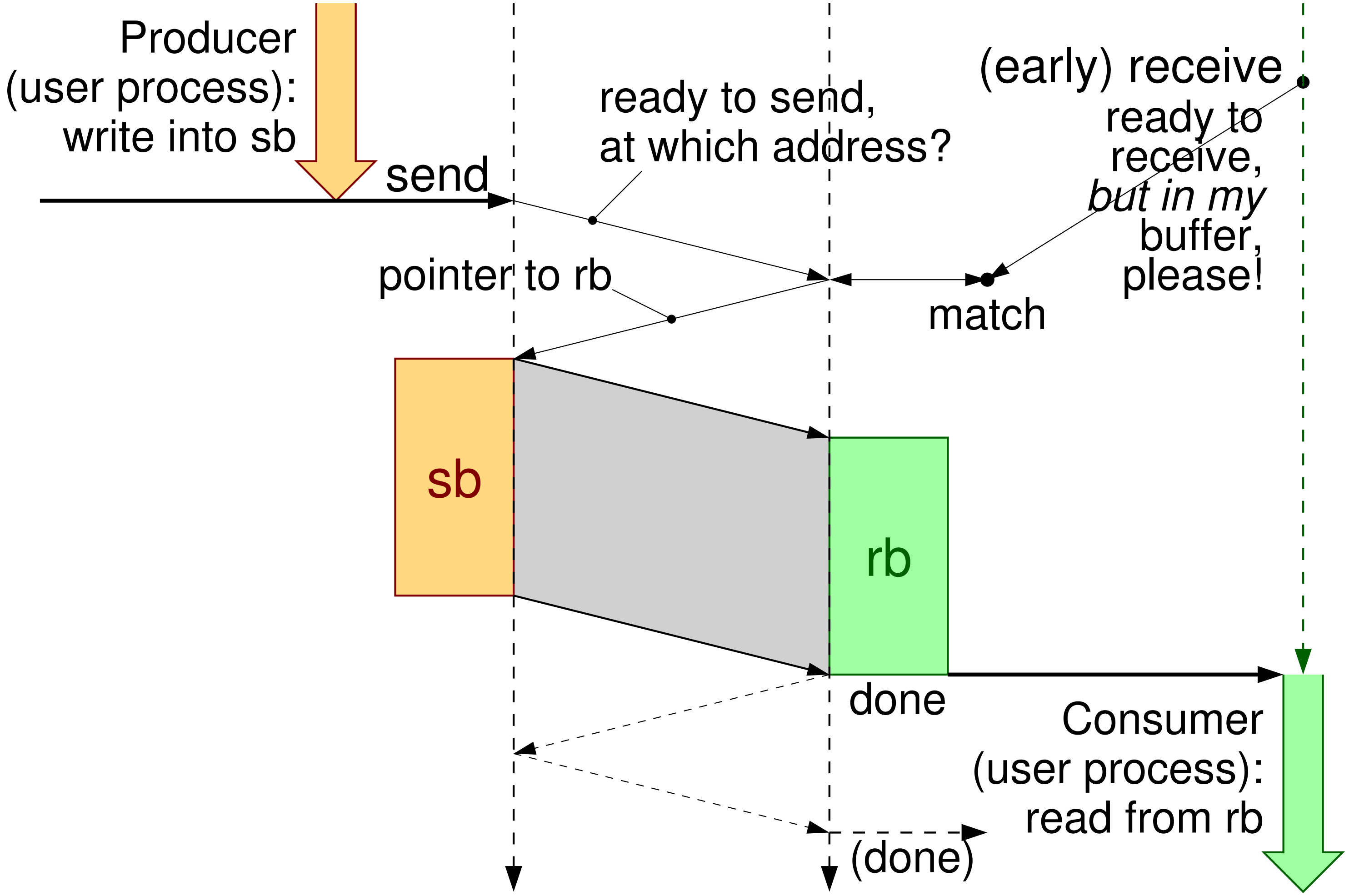
- Address translation architecture of A53 in Xilinx Zynq UltraScale+
- DMA engine is virtualized, but truncates virtual addresses to <64 b
- no mechanism to send the *untranslated* VA to the network
- (remote) load/store instructions suffer compulsory addr. translation

Why copy data between User and Kernel/Library buffers?



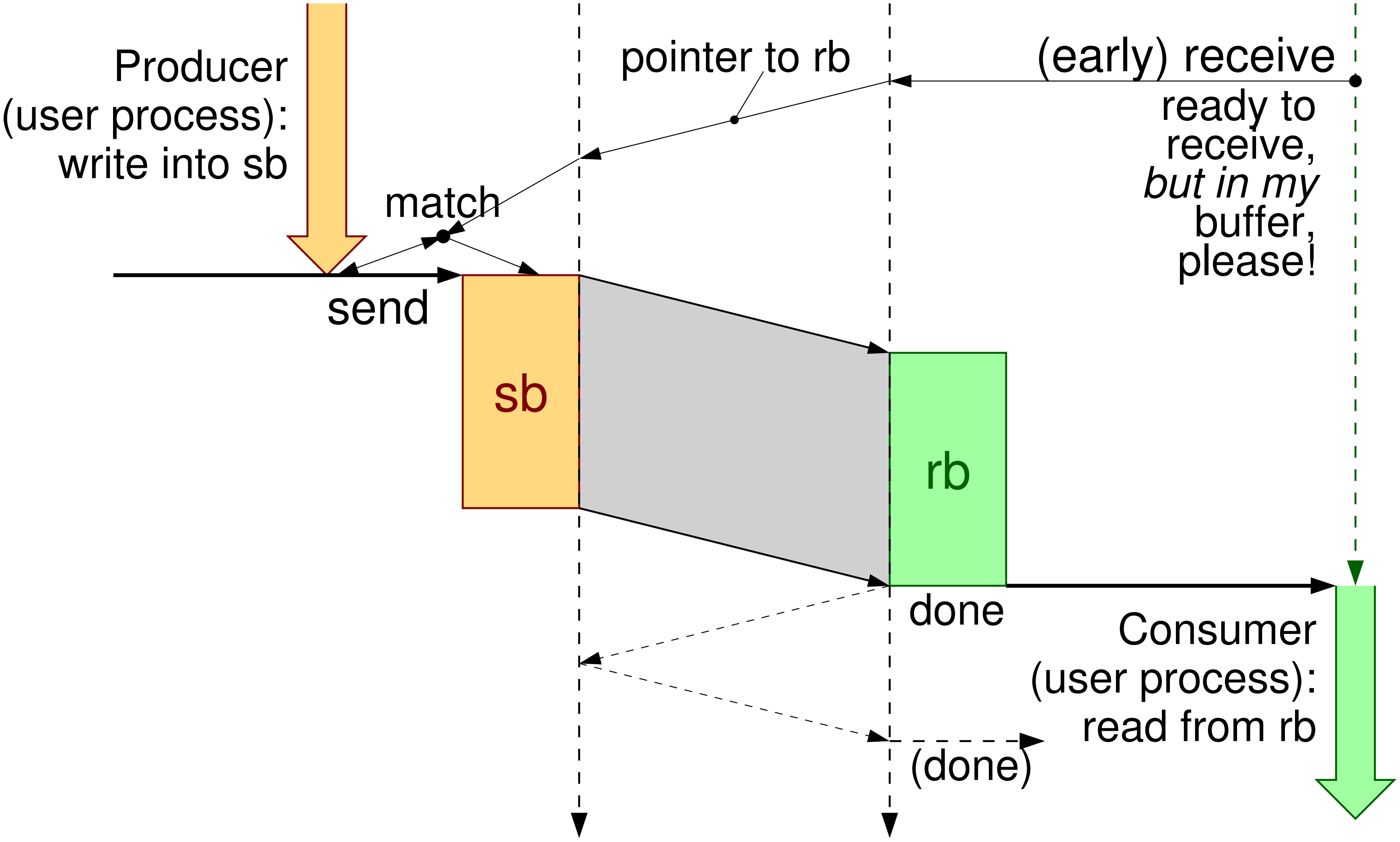
- “Pinned” buffers: traditional DMA does not tolerate page faults
- “Registered” buffer addresses, in the lack of “System” (I/O) MMU
- Non-cacheable buffers, in the lack of cache-coherent DMA
- Send buffer reuse immediately after send initiation
- Transfer occurs before receive-buffer ready, *and* receive Application unwilling to accept reception at library-specified address

MPI: Send-Rcv Match at Receiver, User-Specified Rcv Addr



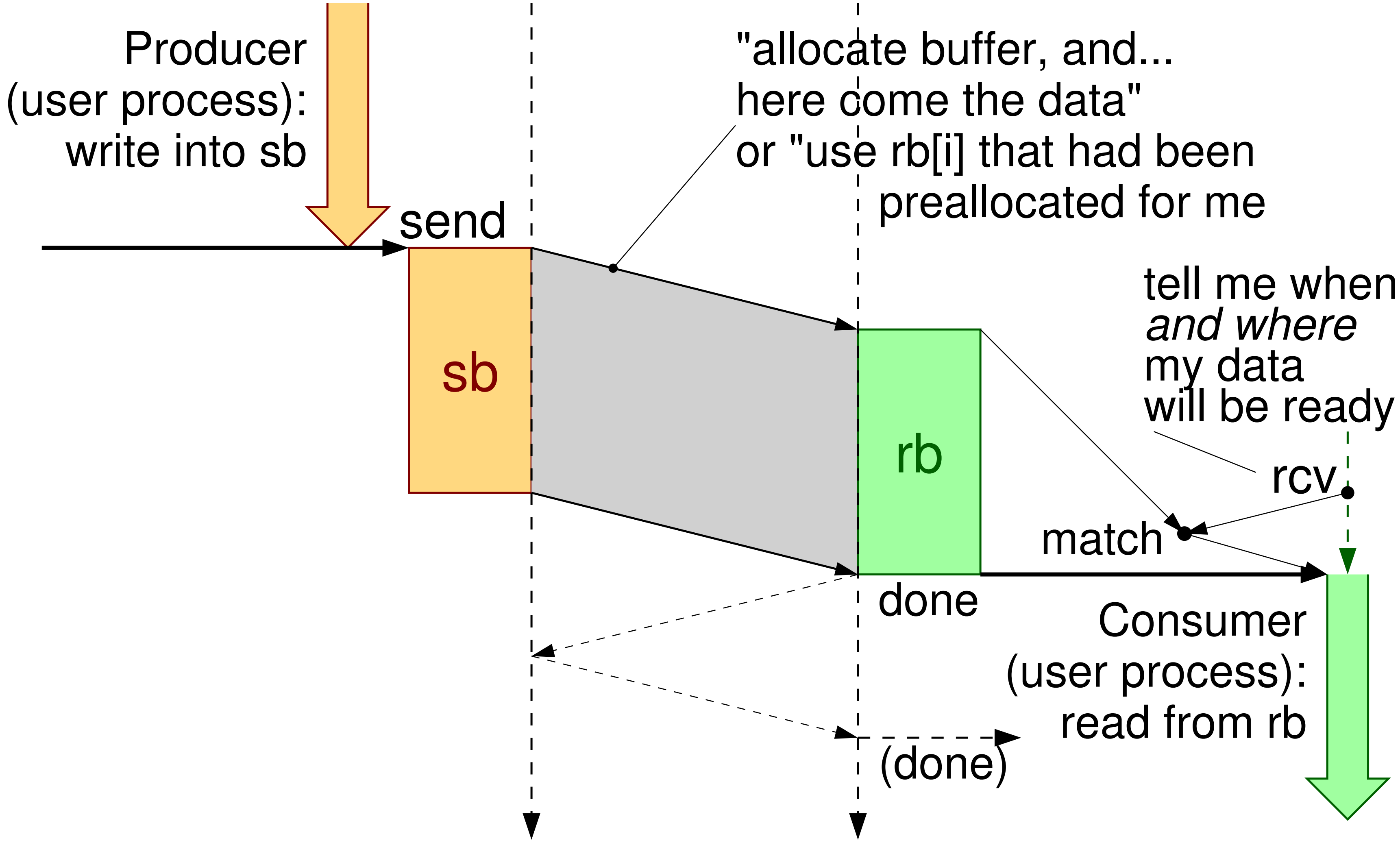
- cost = one extra network round-trip time for sender to learn the address

Snd-Rcv Match at Sender, User-Specified Receive Address



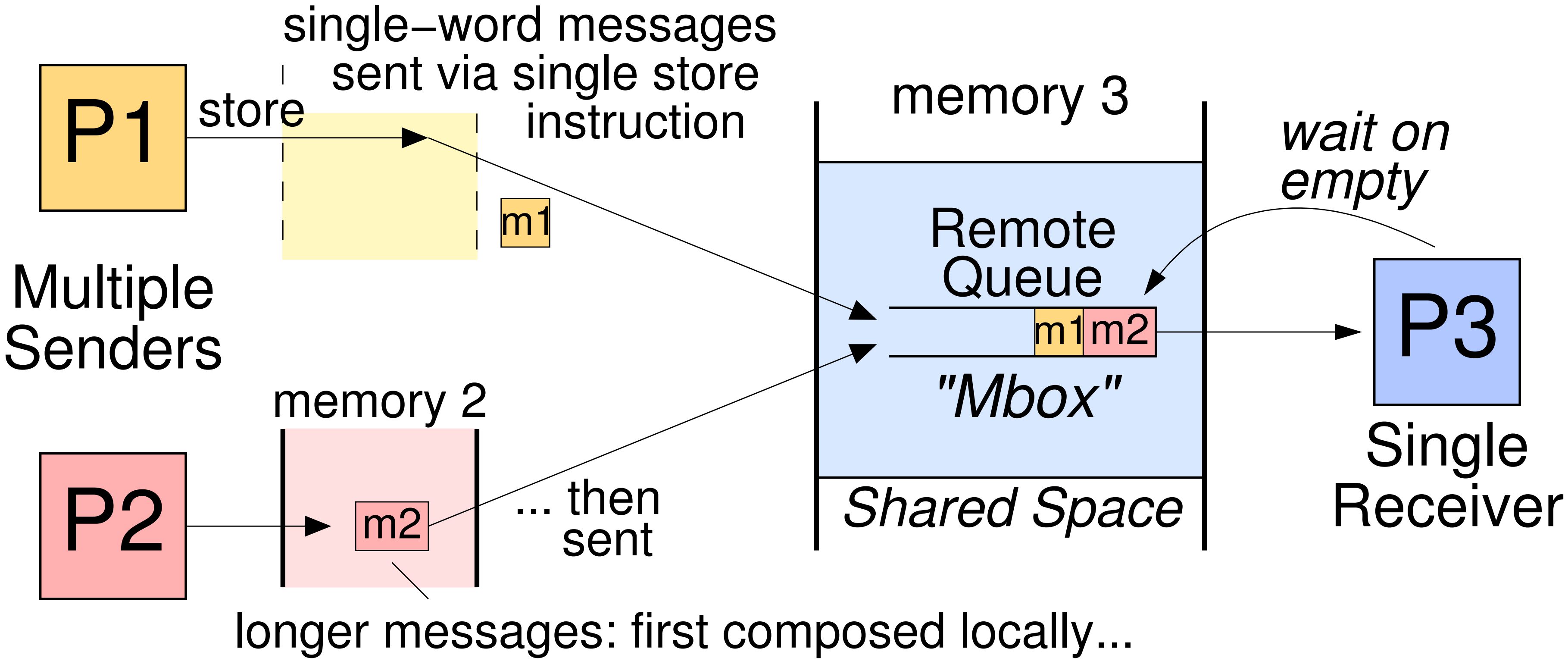
- minimal latency achieved; early receive (rb pre-alloc by user) still needed
- does not work with MPI_ANY_SOURCE

Receiver willing to accept Data at *any* Address (and not copy)



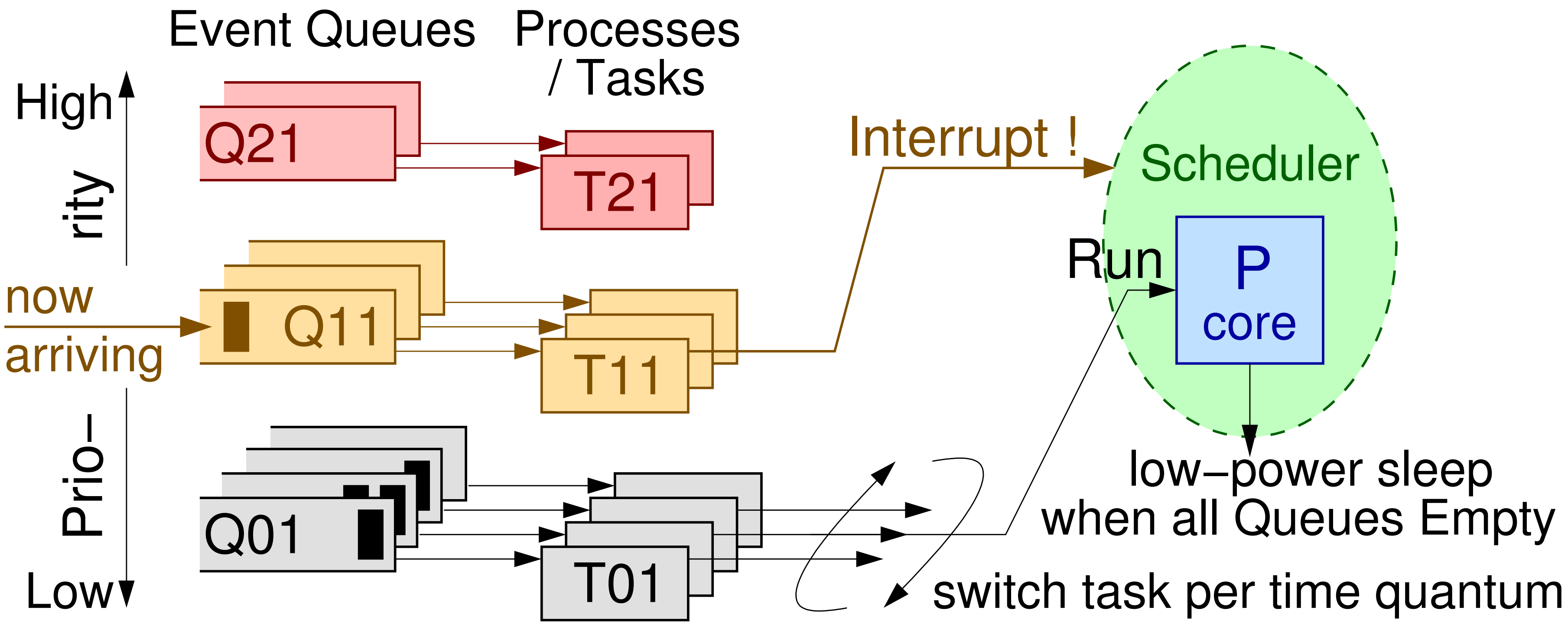
- Eager delivery at preallocated, library-managed buffer, then given to user

Mbox – Remote Enqueue: Asynchronous Event Multiplexing



- Atomic enqueue into shared space, unlike dedicated space with RDMA
- Space reserved only for number of actual senders – not potential senders
- Event Multiplexor – requests, notifications (like *Select* system call)
- RDMA completion should enqueue notification to receiver and sender

Per-Task Mbox Q's & Scheduler as Interrupt Generalization



- Tasks block and wait when synchronously reading from an empty queue
- Scheduler selects a non-empty queue of highest priority and runs its task
- Time quantum switches Q's and Tasks inside top non-empty priority class
- Enqueues into higher priority Q's interrupt lower-priority running tasks

RDMA is the Datapath – Queues are the Control

Conclusions

- Interprocessor Commun. analogous to load/store instr.
- No system calls – virtualized DMA engines
- Need a new Address Translation Architecture

  ⇒*Global Virtual Address Space (GVAS)*
- Notifications, Mboxes, Scheduler: generalized interrupts
- Adapt libraries & API's to user-level, zero-copy commun.

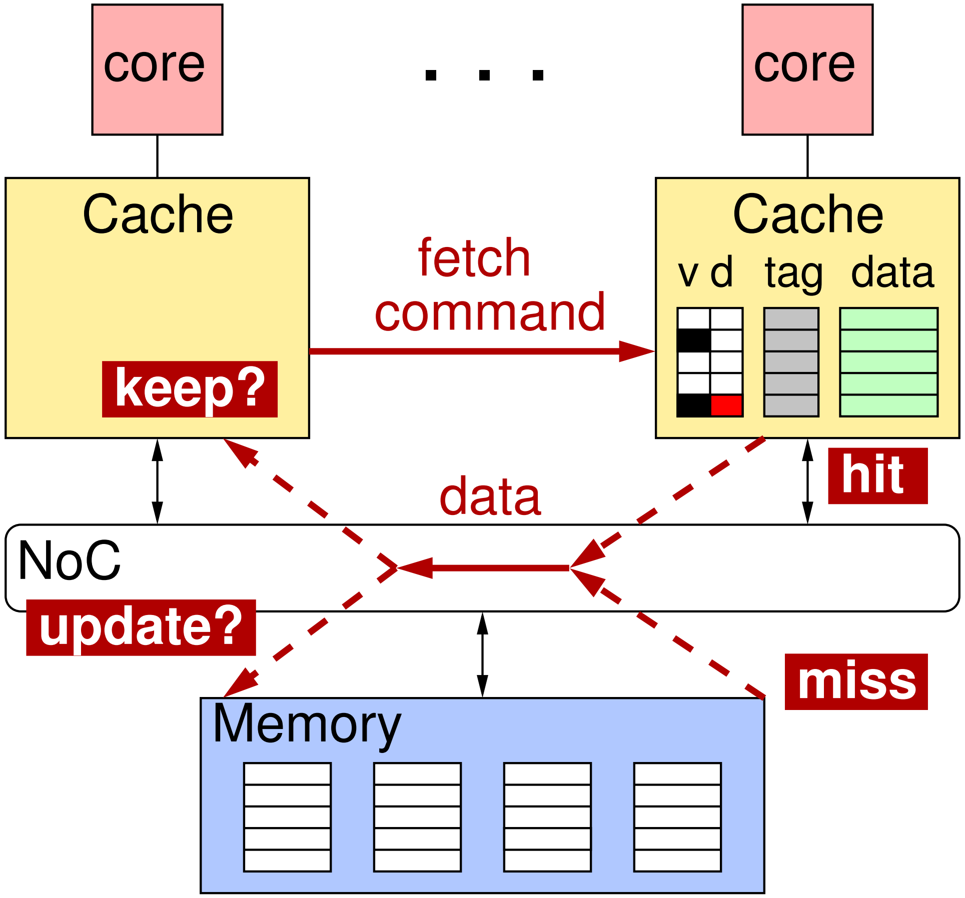
Backup Slides

RDMA/wr-allocate? – Software-Guided Prefetch & Coherence

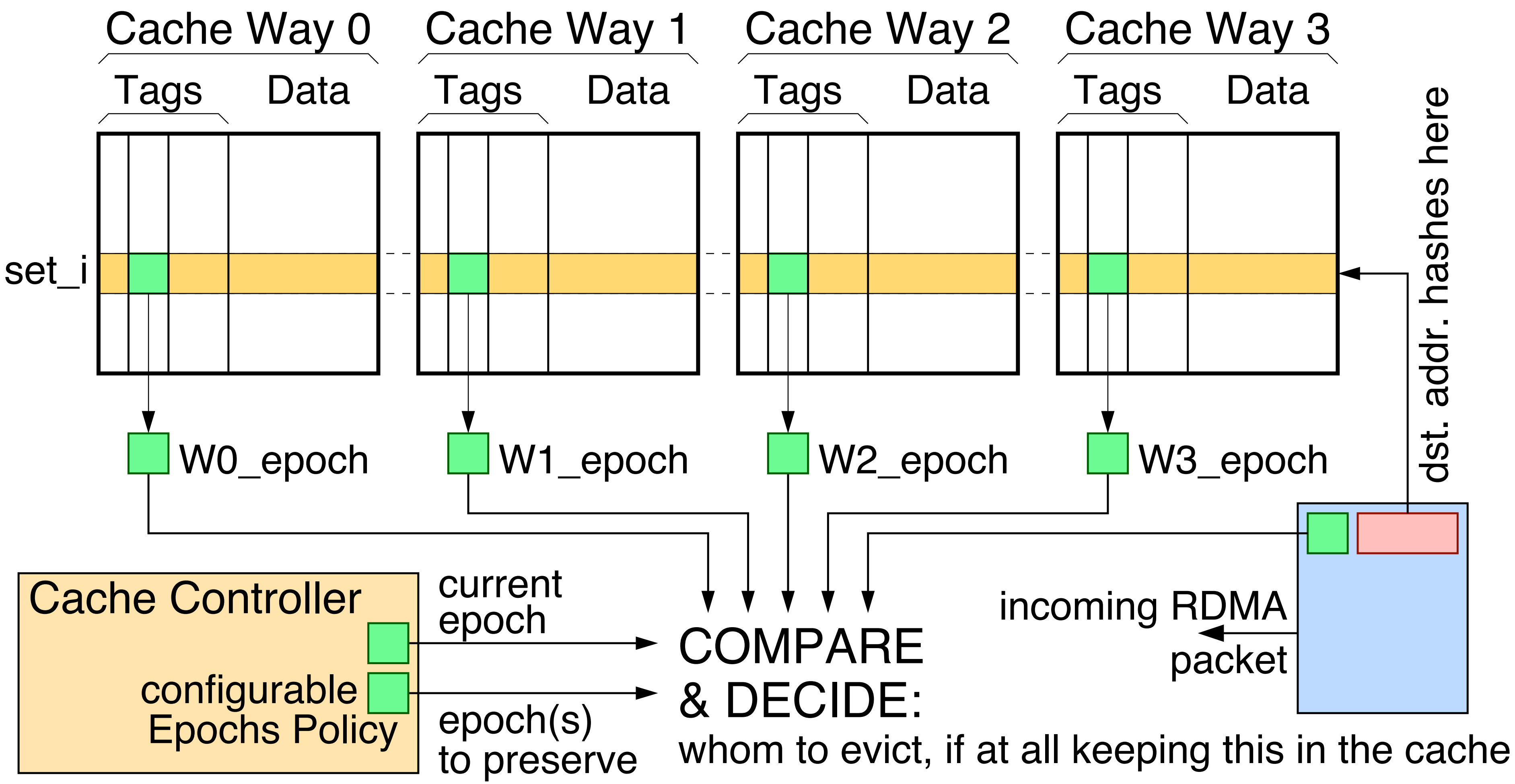
Cache coherence through Software: *simpler HW, energy efficient*

Vassilis Papaefstathiou: ICS'13, and PhD Dissertation

- SW fetches data from the cache of the last producer
  - before task execution
  - prefetch while executing other task
  - wait for completion
- FETCH (read-only)
  - for read-only task arguments
  - override old copies at destination
- FETCH-O (with ownership)
  - for read-write task args
  - migrate cache-line ownership
  - ensure that each cache-line is **dirty in at most one cache** (else, write-back order is undefined)

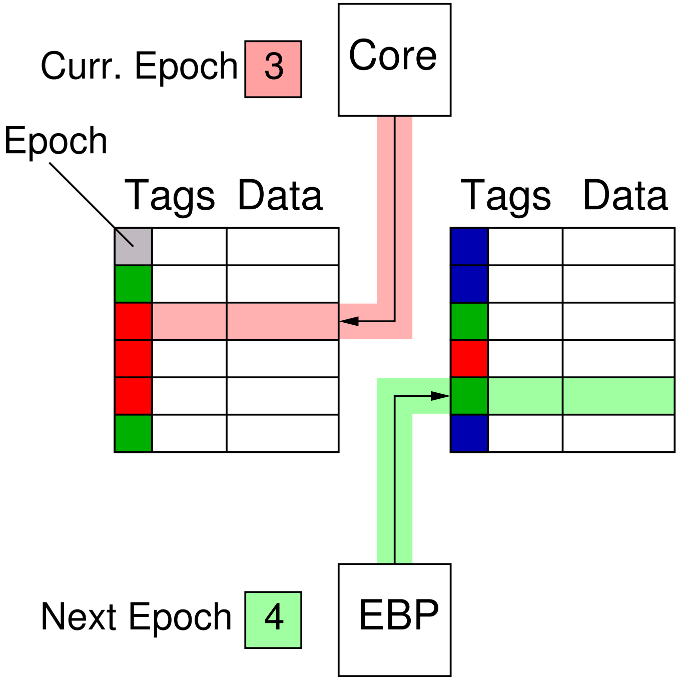


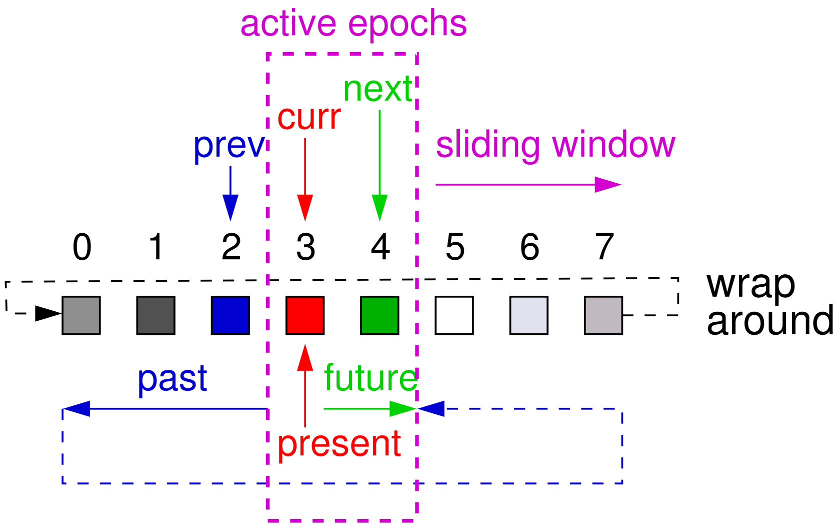
Epoch Numbers: allow SW to control Cache Replacements



- Use a few (e.g. 3) tag bits per cache line to record which “group” of cache lines each belongs to; configure replacement policy by groups

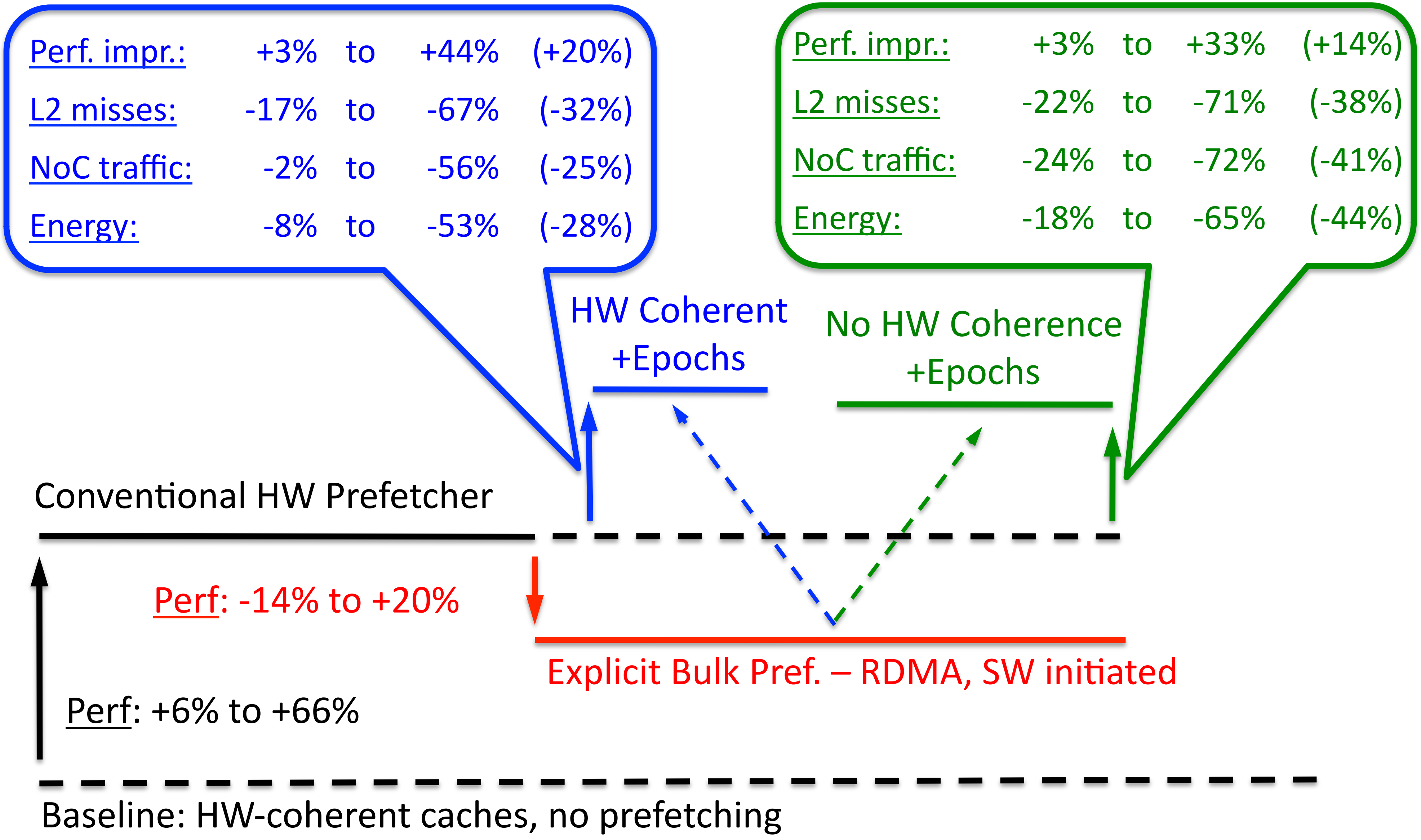
Epoch Numbers Identifying Task Lifetimes





- Scratchpad-like properties with the convenience of caches
- DMA prefetched lines ← "next" epoch
- runtime advances "current" epoch after tasks finish
- evict "past" lines first,…
- … then evict from epoch(s) with # of lines per set > their quota

Evaluation Summary: Coher. (HW/SW), Prefetch (gradual/bulk), Epochs



Perf. impr.:	+3%	to	+44%	(+20%)
L2 misses:	-17%	to	-67%	(-32%)
NoC traffic:	-2%	to	-56%	(-25%)
Energy:	-8%	to	-53%	(-28%)

Perf. impr.:	+3%	to	+33%	(+14%)
L2 misses:	-22%	to	-71%	(-38%)
NoC traffic:	-24%	to	-72%	(-41%)
Energy:	-18%	to	-65%	(-44%)

HW Coherent +Epochs

No HW Coherence +Epochs

Conventional HW Prefetcher

Perf: -14% to +20%

Explicit Bulk Pref. – RDMA, SW initiated

Perf: +6% to +66%

Baseline: HW-coherent caches, no prefetching