NOCS 2020

PART: Pinning Avoidance in RDMA Technologies

Antonis Psistakis *, Nikolaos Chrysos, Fabien Chaix, Marios Asiminakis, Michalis Gianioudis, Pantelis Xirouchakis, Vassilis Papaefstathiou, Manolis GH Katevenis

Institute of Computer Science (ICS), Foundation for Research and Technology-Hellas (FORTH)
Heraklion, Crete, Greece

24-25 September 2020

FORTH
INSTITUTE OF COMPUTER SCIENCE

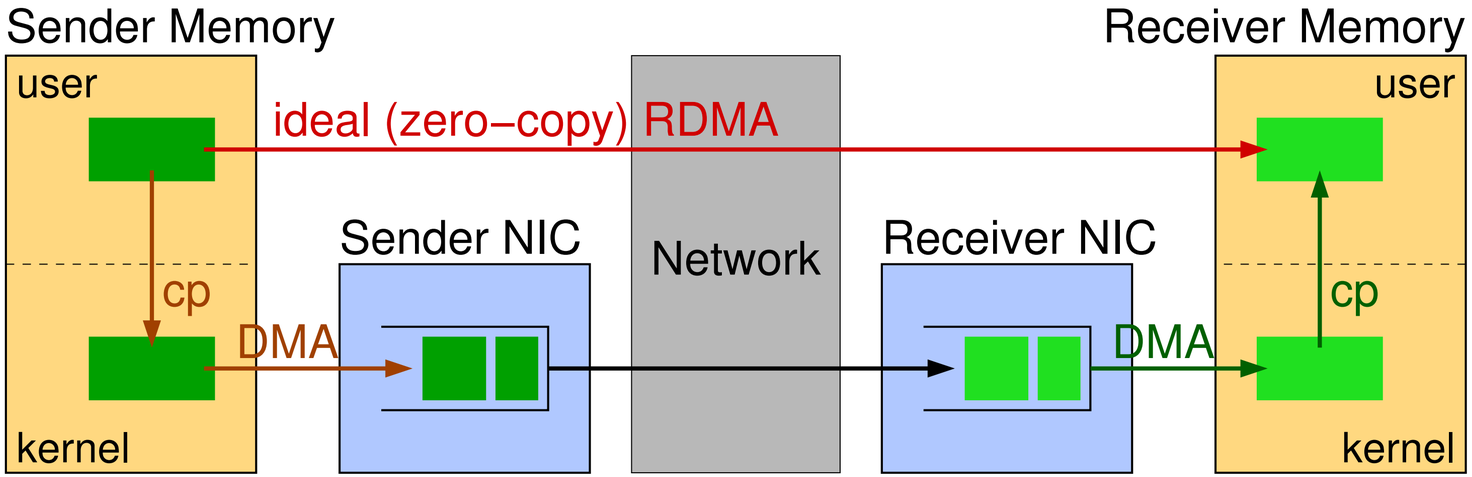
* currently: PhD student at UIUC (iacoma group)

Outline

- Background
- Contributions
- Environment
- PART Mechanism
- Evaluation
- Conclusion

Background

- Modern computing systems (e.g. datacenters, supercomputers) strive to **eliminate use of kernel path** in communication
  - systems calls, undesirable memory copies during transfers (**performance**)
- Alternative: User-level initiated **RDMA** allows users to **bypass OS**, thus avoids overheads
  - but, mandates use **virtual addresses** for transfers…

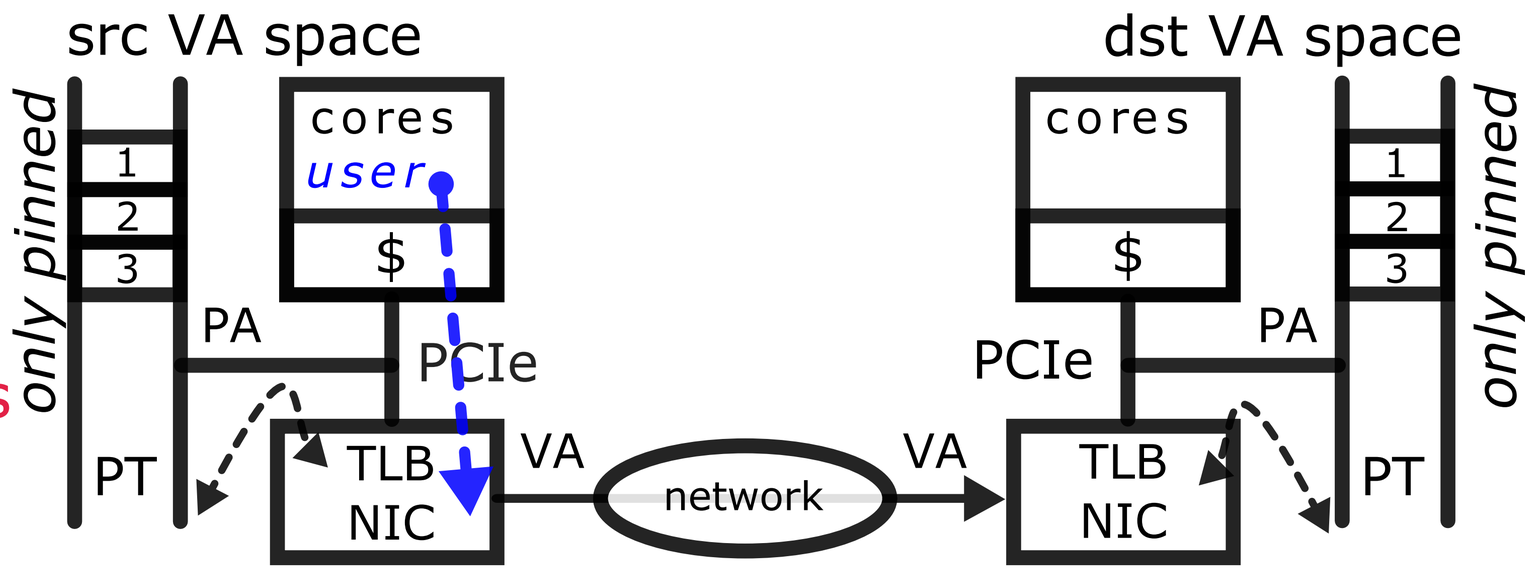


User-level initiated RDMA vs TCP-like communication

Image source: M. Katevenis "I/O is no longer slow" PER-18 Workshop, 2018

Background: State-of-the-art

- Common RDMA technologies (RoCE, IB) **copy** page mappings into NICs, which become responsible to handle the address translations
  - planted mappings on TLB-like tables per core (**expensive**)
  - need to avoid page faults on network path → **pinning** communication buffers
- But, **pinning** is bad
  - Hinders **memory utilization**
  - Incompatible with various **OS optimizations** (e.g. THP)
  - Pin & unpin ⇨ extra system calls
  - Increases program. complexity



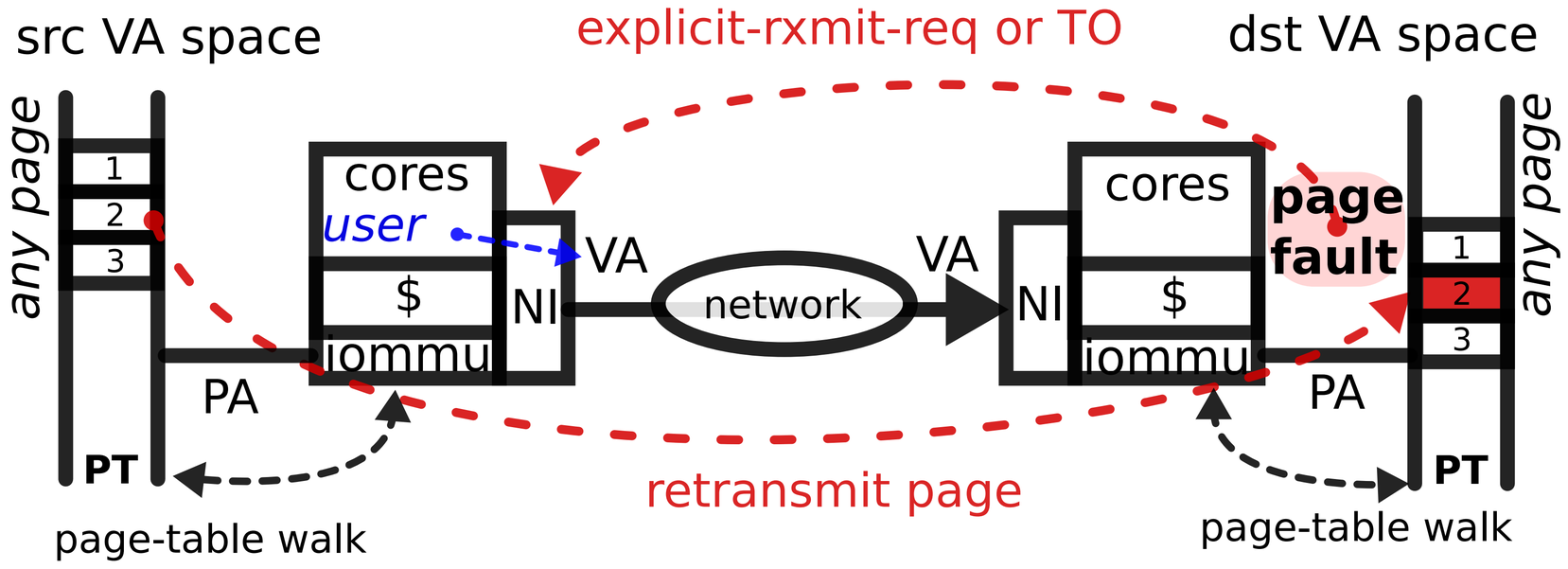
State-of-the-art: Pin pages, plant mappings to NIC

Outline

- Background
- Contributions
- Environment
- PART Mechanism
- Evaluation
- Conclusion

Towards the next generation RDMA

- In modern well-designed systems, **page faults** are expected to be **rare**
  - **Re-usability** of buffers
- The **working-sets** of **HPC applications** are dimensioned to **fit in memory**



Next-generation: Dynamic paging, handle page faults on the fly

This paper: Handles page faults similarly to other transmission errors

Contributions

- PART handles **page-faults** during RDMA by retransmitting failed pages (**no pinning**)
  - includes system software and hardware components
- We implement PART on a **NI, closely coupled** with **the processor**, and we **re-use the IOMMU** for address translation instead of relying on specialized NICs
- We evaluate PART in a **cluster of interconnected ARM processors**
- We present results from microbenchmarks and a **real-HPC application** (LAMMPS) using 16 nodes and 64 cores

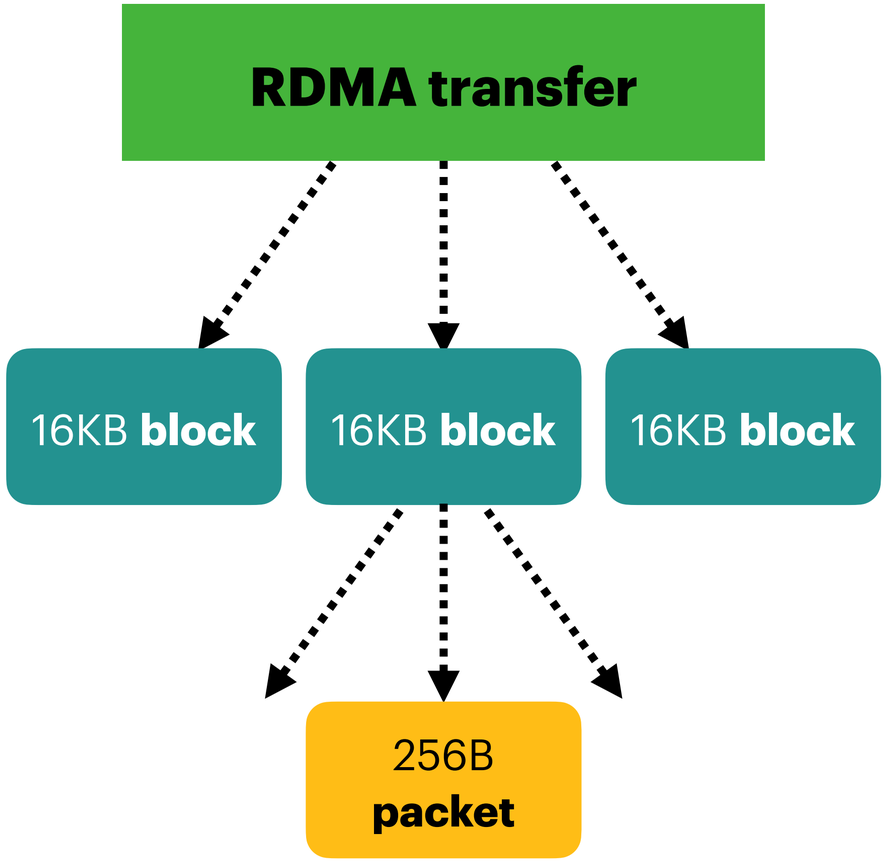
Outline

- Background
- Contributions
- Environment
- PART Mechanism
- Evaluation
- Conclusion

Environment

Our RDMA Engine

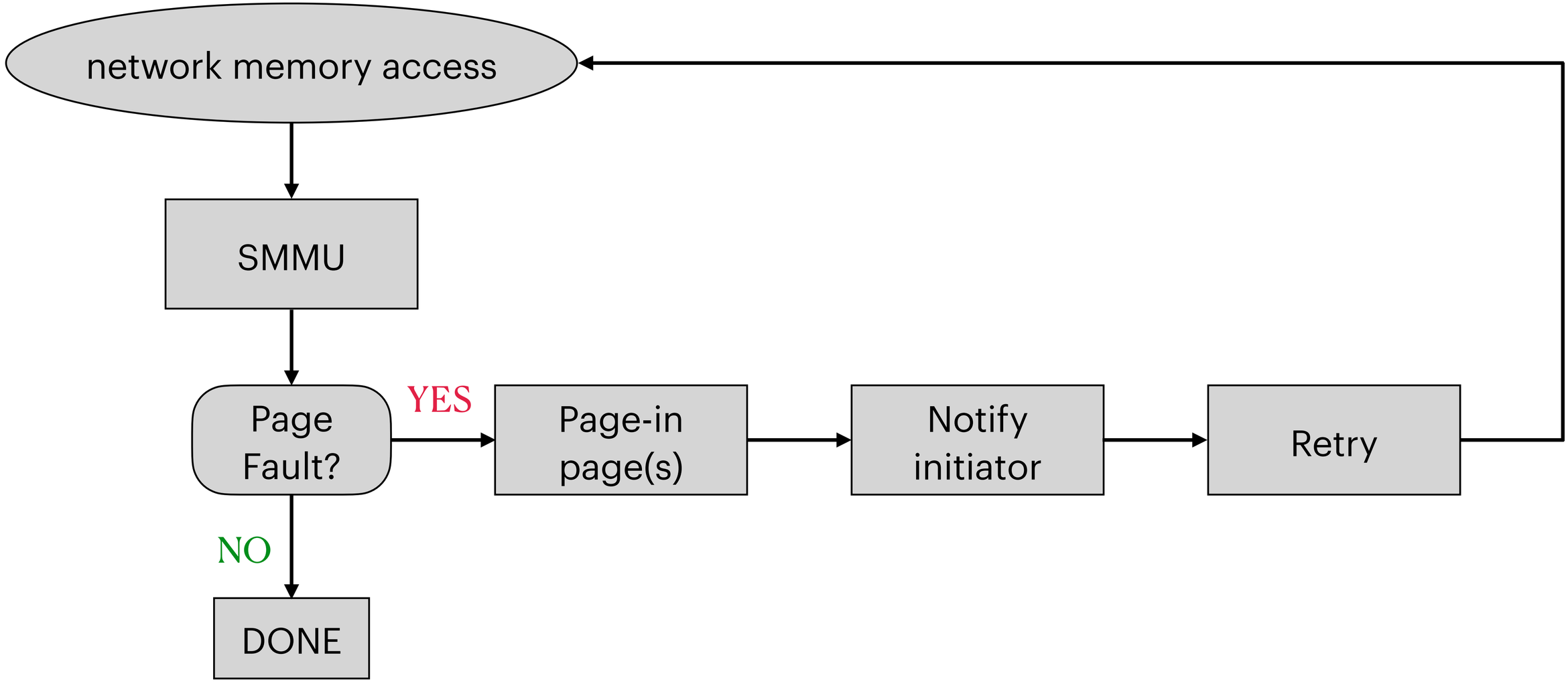
- Segments RDMA transfers into blocks/transactions
  - 1 block = 16 KB = 4x4KB pages
    - e.g. 64 KB transfer ⇨ four (4x16KB) blocks
- Hardware segments blocks further into packets of 256 Bytes
- Selectively retransmits failed blocks (on NACK or TimeOut)



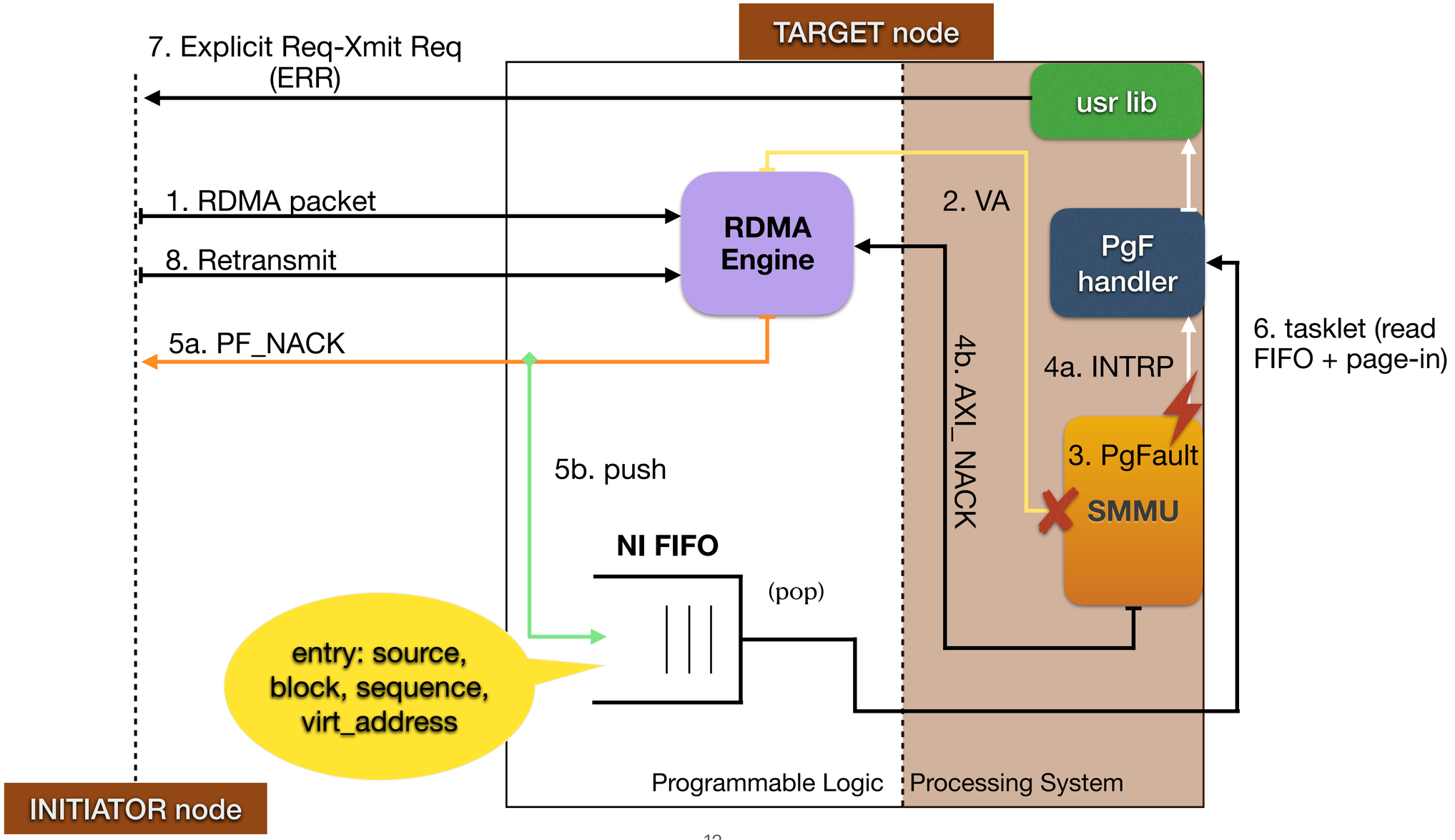
Outline

- Background
- Contributions
- Environment
- PART Mechanism
- Evaluation
- Conclusion

PART Mechanism: The general flow



PART Mechanism: Handling RDMA PgFaults at dest.



Outline

- Background
- Contributions
- Environment
- PART Mechanism
- Evaluation
- Conclusion

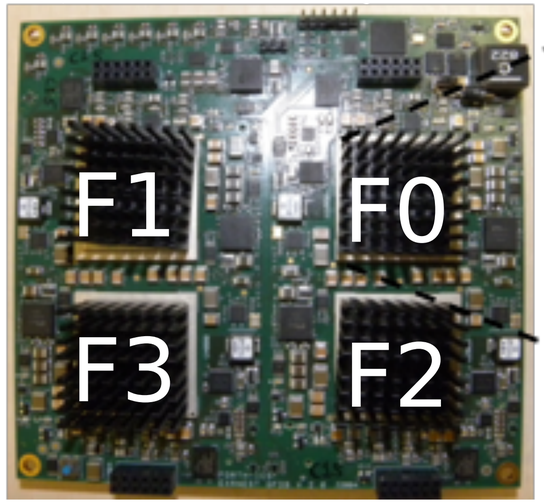
Evaluation: Testbed

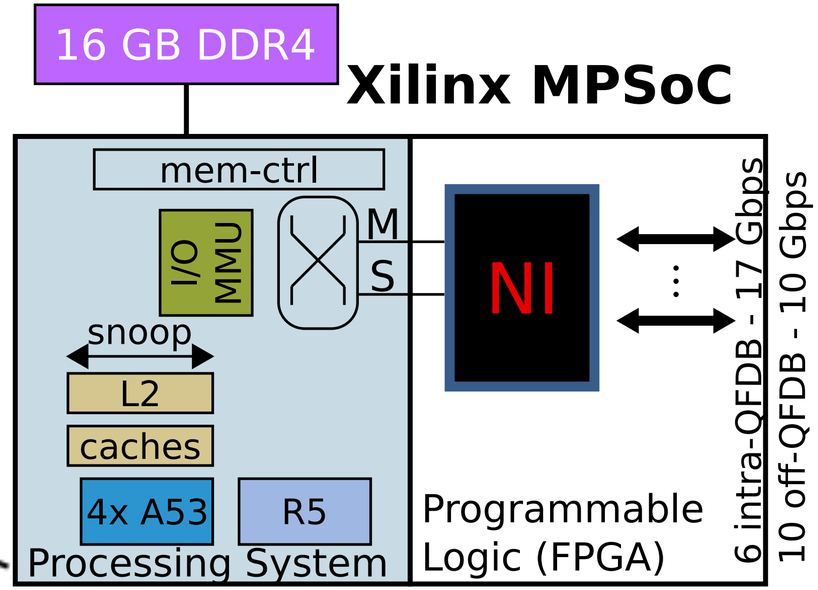
One Blade (Mezzanine) consists of 4 Quad-FPGA Daughter Boards (QFDBs)

- Each QFDB:
  - 4 Xilinx Ultrascale+ MPSoCs
    - 4xA53 (1.1 GHz) on each FPGA
  - 64 GB of DDR4 SDRAM (16 GB per MPSoC)
  - 10-17 Gbps High Speed Serial Links

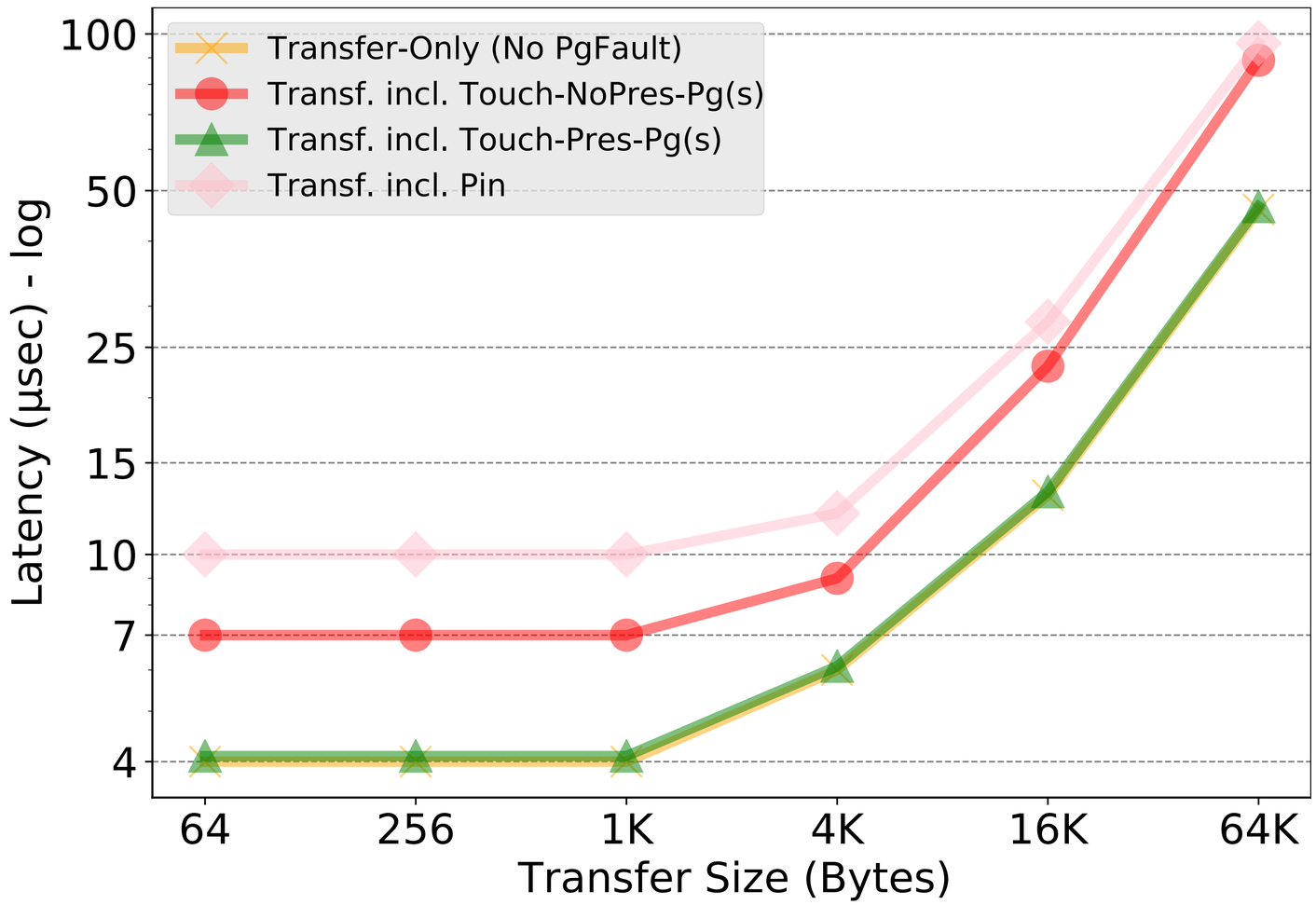
In our evaluation we present only **minor page faults**

QFDB



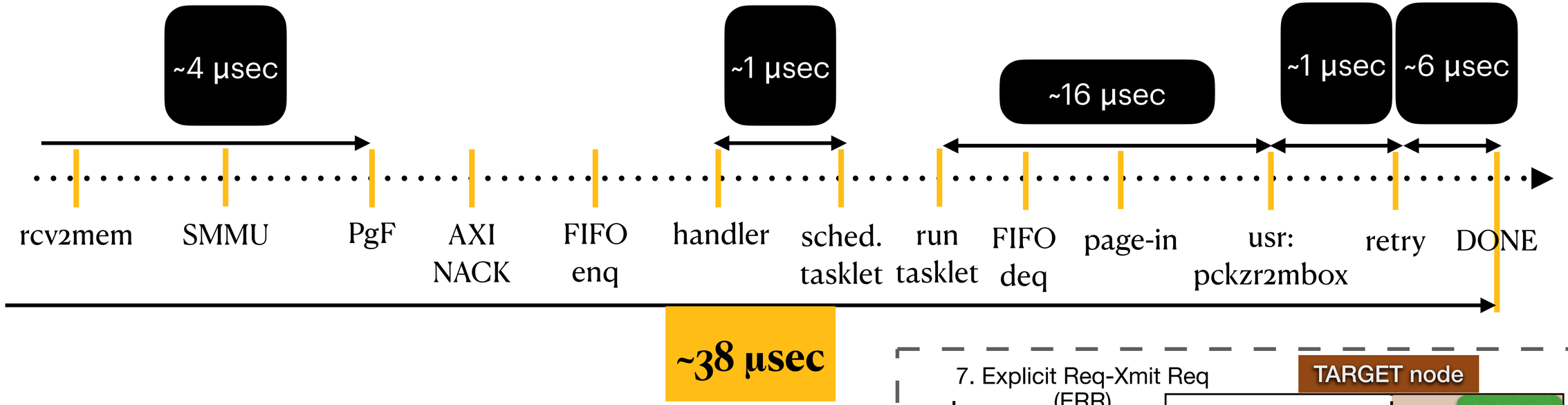


Evaluation: RDMA Latency (no page faults)



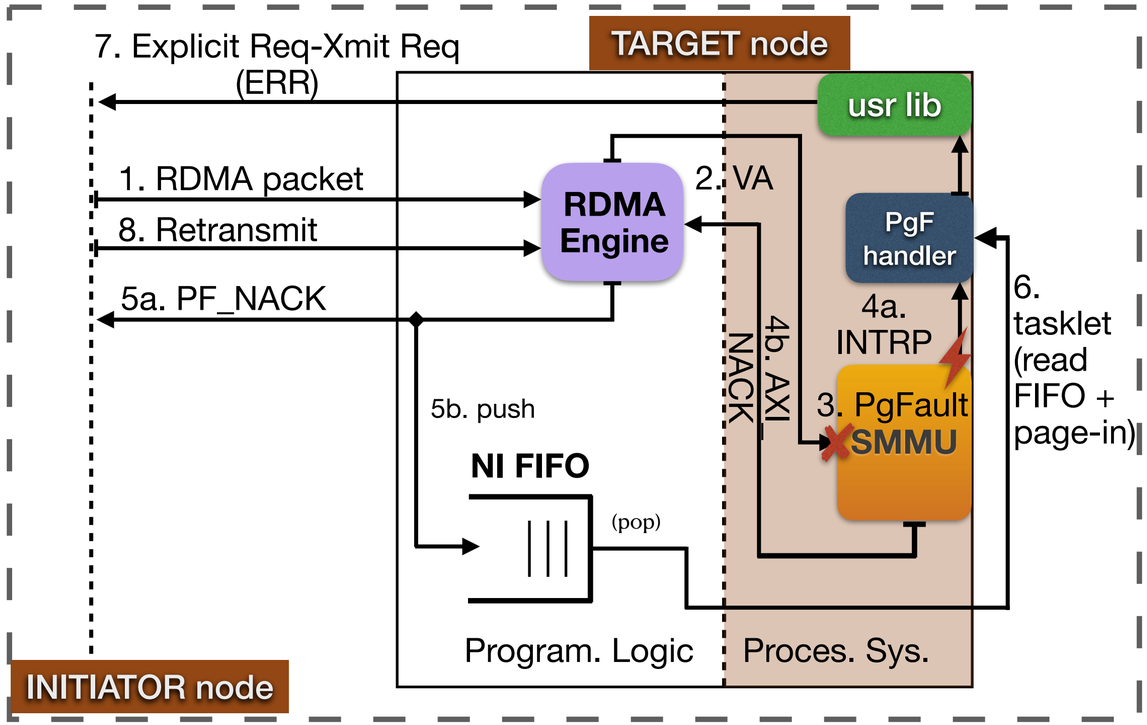
- "Transfer-Only" and "Transf. incl. Touch-Pres-Pg(s)" are identical ⇨ Touching a present page takes approx. 200 nsec.
- "Pin" and "Touch-NoPres-Pg(s)" converge when the transfer size increases ⇨ Both exhibit MMU page fault(s)

Evaluation: Latency Breakdown (4KB)

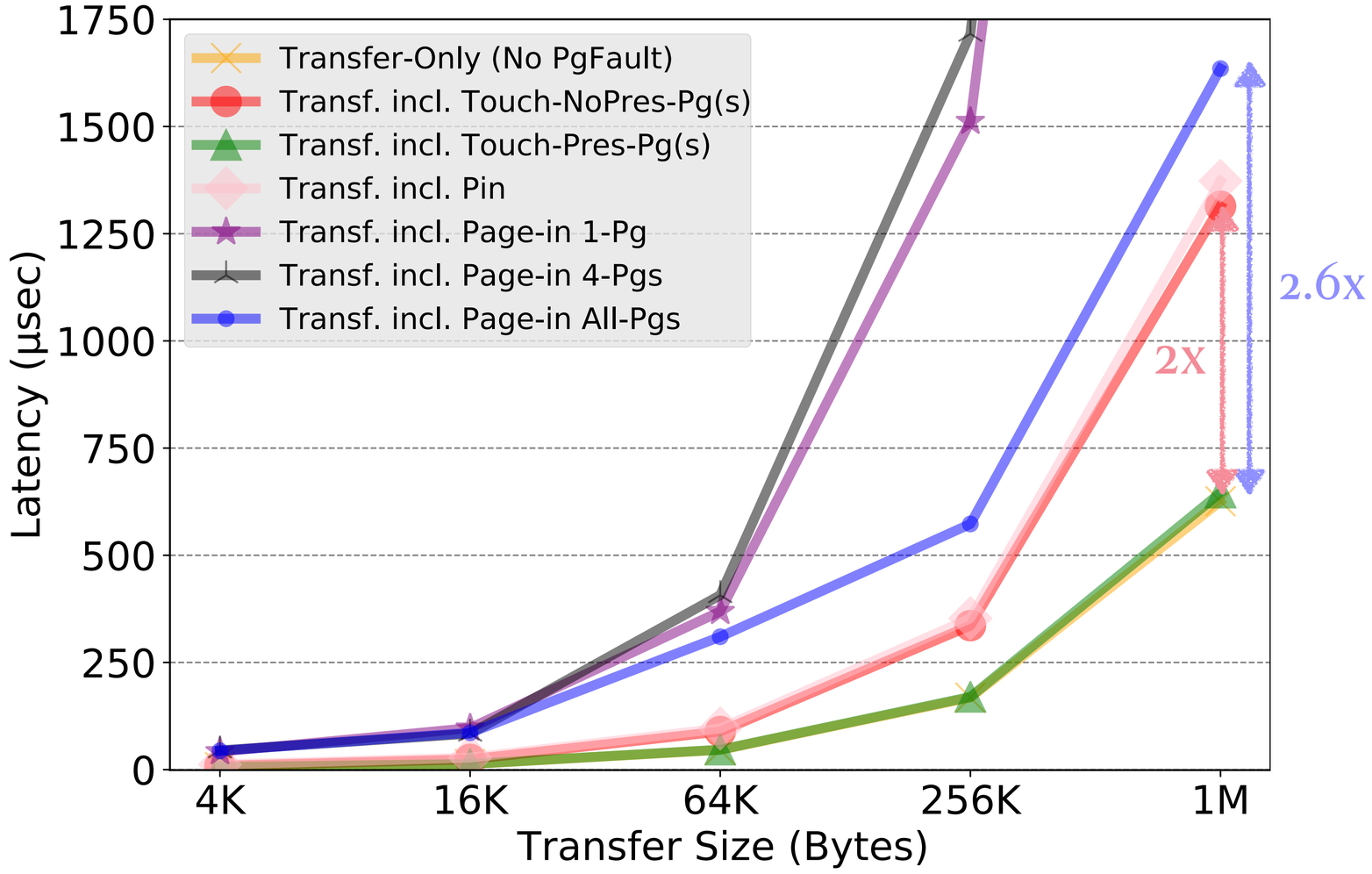


Tasklet involves:

1. Reading from FIFO
2. Sending information to userspace through Netlink sockets (~10 µsec)

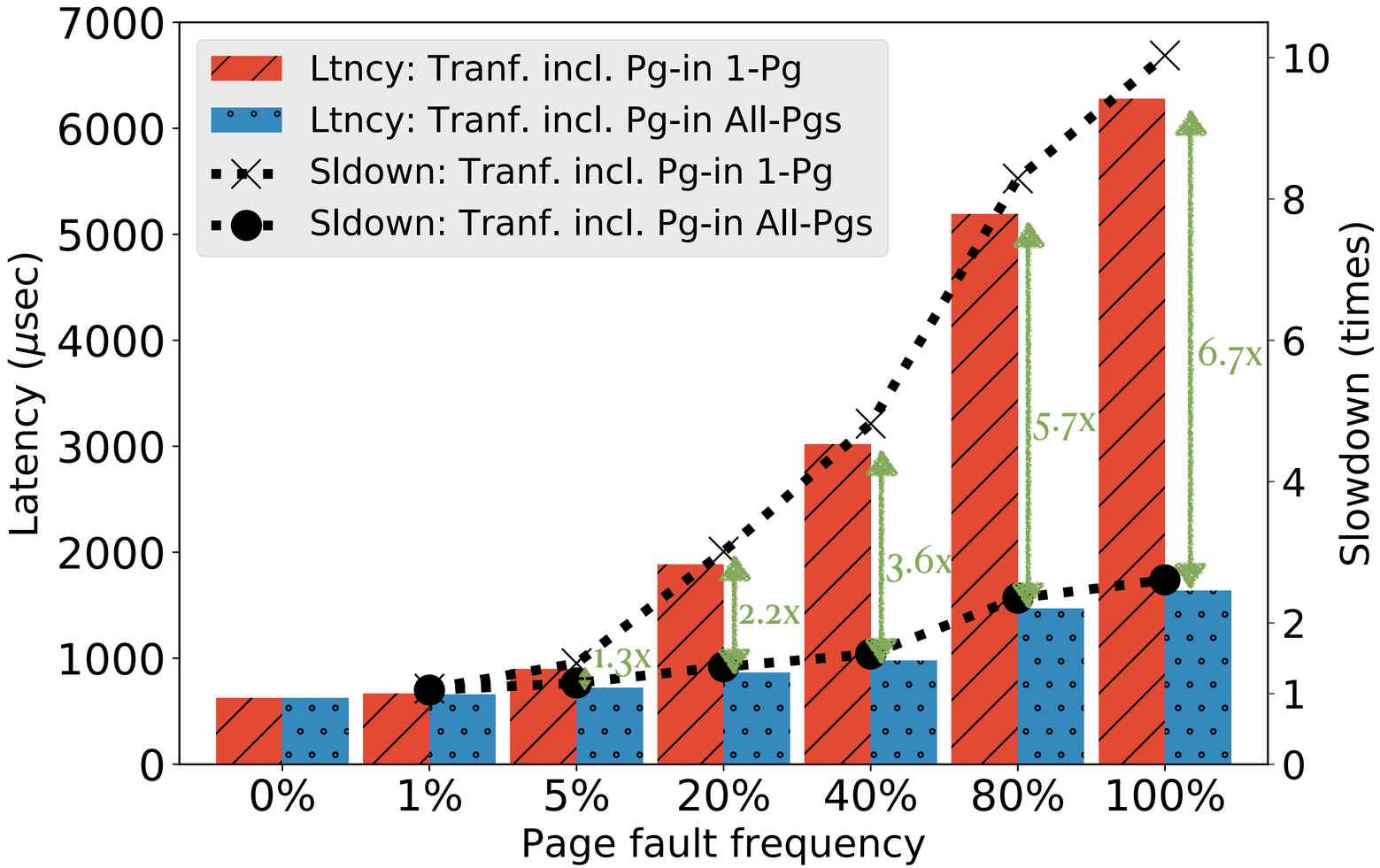


Evaluation: Page fault at dest. on all pages



- Paging-in all pages of the transfer upon the first page fault (Page-in All-Pgs) provides the best performance compared to alternatives
- For 1MB, compared to "Transfer-Only":
  - Page-in All-Pgs (PIA): 2.6x worse
  - Touch-NoPres-Pg(s): 2x worse

Evaluation: Page fault at dest. on a percentage of pages



- Slowdown comparing "Page-in All-Pgs" and "Transfer-Only" (0%):
  - 1.15x for up to 5%, 1.56x for 40%, and 2.6x for 100%

Evaluation: Real HPC application (LAMMPS)

- Comparing baseline (no page-fault) to PART: **no performance degradation**

	Loop time (sec)		Timesteps/s	
Processes	**Baseline (no pg-fault)**	**PART**	**Baseline (no pg-fault)**	**PART**
1	76.2046	76.1135	1.312	1.314
2	79.1183	79.0789	2.528	2.529
4	84.2753	84.1988	4.746	4.751
16	97.1399	96.7787	16.471	16.533

Outline

- Background
- Contributions
- Environment
- PART Mechanism
- Evaluation
- Conclusion

Conclusion

PART mechanism:

- tolerates (handles) occasional page faults, leveraging re-xmit capabilities of NIC
- re-uses IOMMU & process page table: no need for separate mem. management
- avoids pinning (simplifies programming model, enjoys OS advantages)

Performance:

- Overheads of handling page faults during RDMA
  - up to 1.1x for small fraction of pages (up to 5%)
  - up to 2.6x for all pages
  - no overhead for a real-HPC app (LAMMPS)

Any questions?

Thank you!