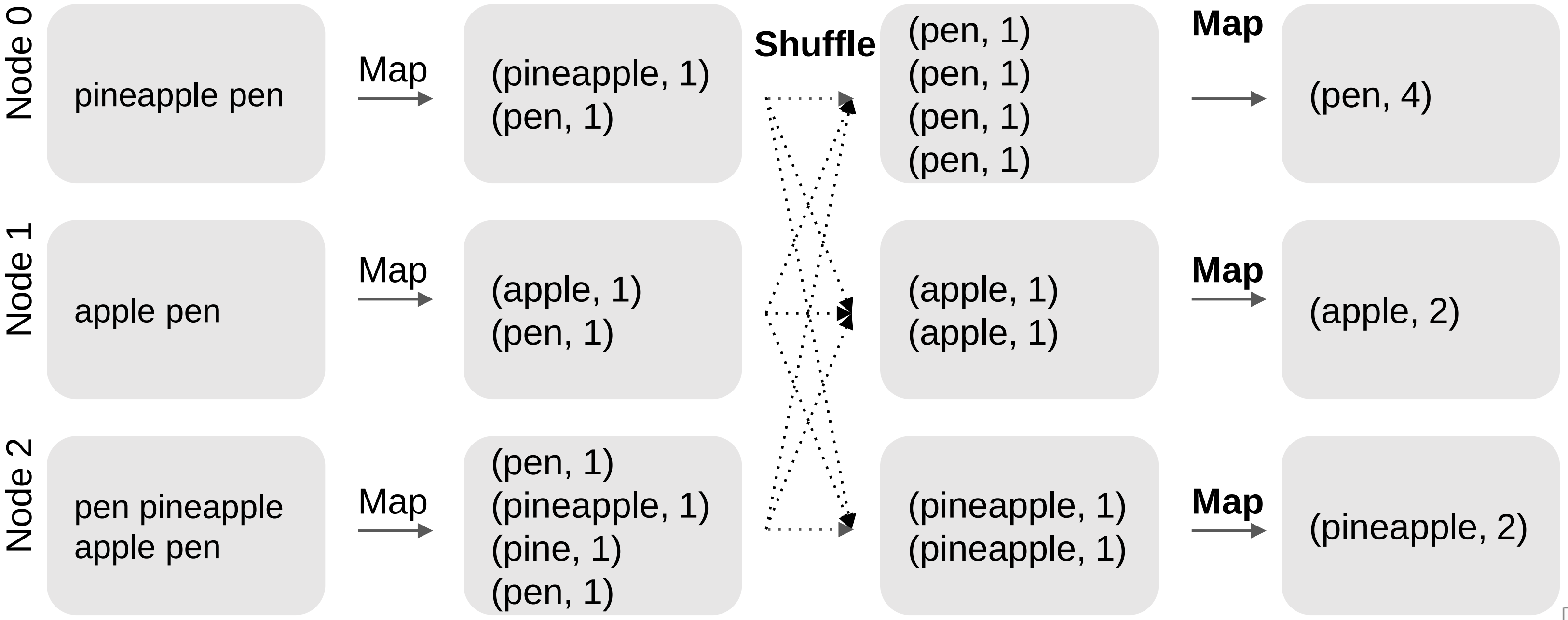
ICCD 2021 Paper #4

# PSACS: Highly-Parallel Shuffle Accelerator on Computational Storage

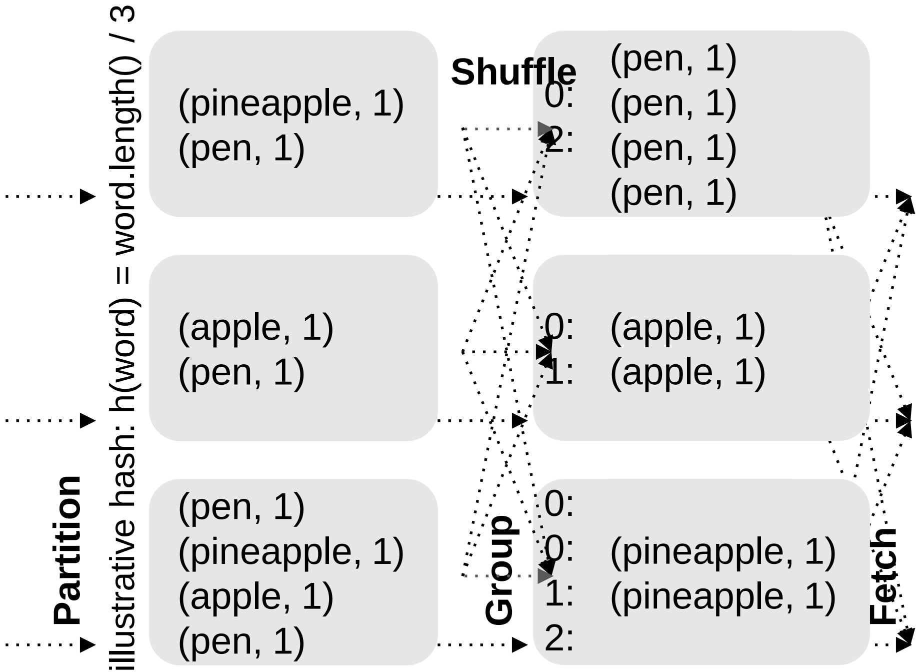
Chen Zou[1], Hui Zhang[2], Yang Seok Ki[2], Andrew A. Chien[1,3]
{chenzou@, achien@cs.}uchicago.edu, {w.hzhang86, yangseok.ki}@samsung.com

[1]Department of Computer Science, University of Chicago
[2] Memory Solution Lab, Samsung Semiconductor Inc.
[3] Mathematics and Computer Science, Argonne National Laboratory

# Shuffle enables parallelism exploitation for OLAP



# However, Shuffle may be a bottleneck



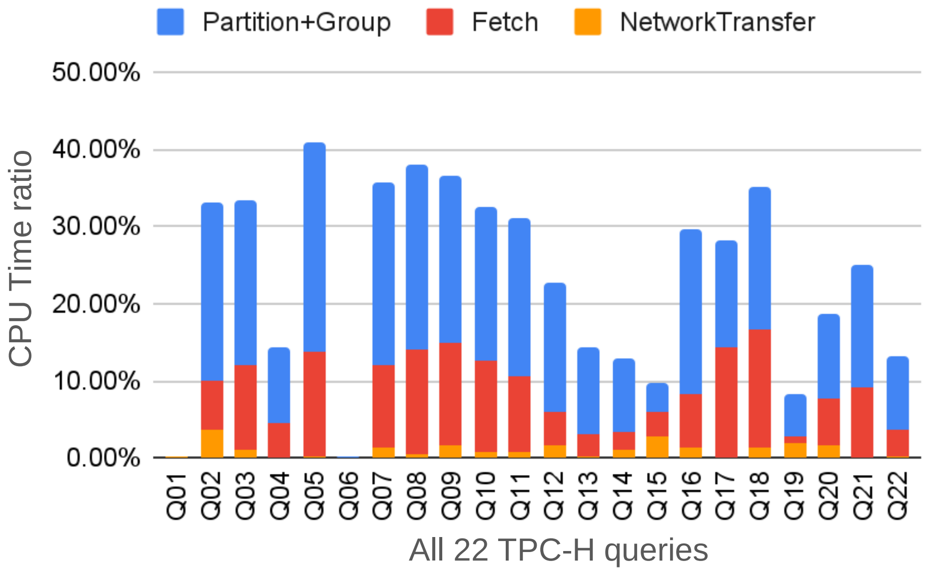

Most challenging!
Cache thrashing. Spills

# Shuffle may take ⅓ latency of an OLAP query

TPC-H with scaling factor 1000.

4 nodes running Spark 3.0.1 connected with 1Gbps ethernet.

CPU time collected with JVM-profiler[6].



# PSACS: Highly-Parallel Shuffle Accelerator on Computational Storage

First shuffle acc on computational storage.

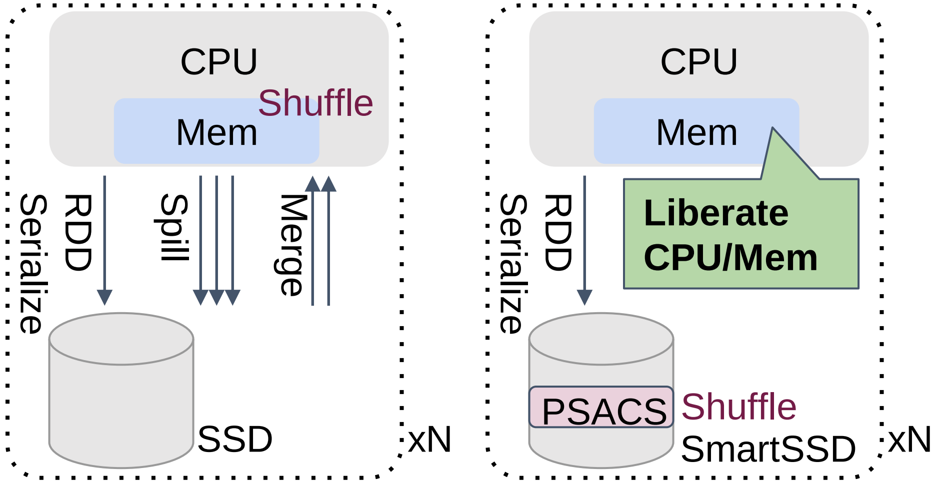
- Confines shuffle traffics in storage
- Liberates CPU and memory

PSACS microarchitecture exploits:

- Task, subtask and data-level parallelism
- Custom scratchpad for efficient gathering

PSACS achieves acceleration benefits:

- 5x kernel-level shuffle throughput
- 23% OLAP query speedup on average



# PSACS partition approach: Hash

We opt for hash-based partition for its generality.

‘evenness’ rather than ‘collision resistance’ is the key here.

We apply a variant of fold hash that additionally zig-zag the input.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $b_{00}$ | $b_{01}$ | $b_{02}$ | $b_{03}$ | $b_{04}$ | $b_{05}$ | $b_{06}$ | $b_{07}$ | $b_{08}$ | $b_{09}$ |
| $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ |
| $b_{19}$ | $b_{18}$ | $b_{17}$ | $b_{16}$ | $b_{15}$ | $b_{14}$ | $b_{13}$ | $b_{12}$ | $b_{11}$ | $b_{10}$ |
| $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ | $\oplus$ |
| $b_{20}$ | $b_{21}$ | $b_{22}$ | $b_{23}$ | $b_{24}$ | $b_{25}$ | $b_{26}$ | $b_{27}$ | $b_{28}$ | $b_{29}$ |
| | | | | ⋮ | | | | | |
| $\parallel$ | $\parallel$ | $\parallel$ | $\parallel$ | $\parallel$ | $\parallel$ | $\parallel$ | $\parallel$ | $\parallel$ | $\parallel$ |
| $h_{00}$ | $h_{01}$ | $h_{02}$ | $h_{03}$ | $h_{04}$ | $h_{05}$ | $h_{06}$ | $h_{07}$ | $h_{08}$ | $h_{09}$ |

# PSACS group approach

Bucketing (BypassMergeSortShuffleWriter, FPGAPart[12], Vitis[13]):

- Maintain a bucket for each destination partition to hold records going there.
- Hard to implement growable buckets without malloc-like dynamic memory allocation in hardware accelerators.
- Static bucket allocation presents linear scaling between capacity and #partitions.
- Parallelism exploitation leads to another shuffle problem.

# PSACS group approach

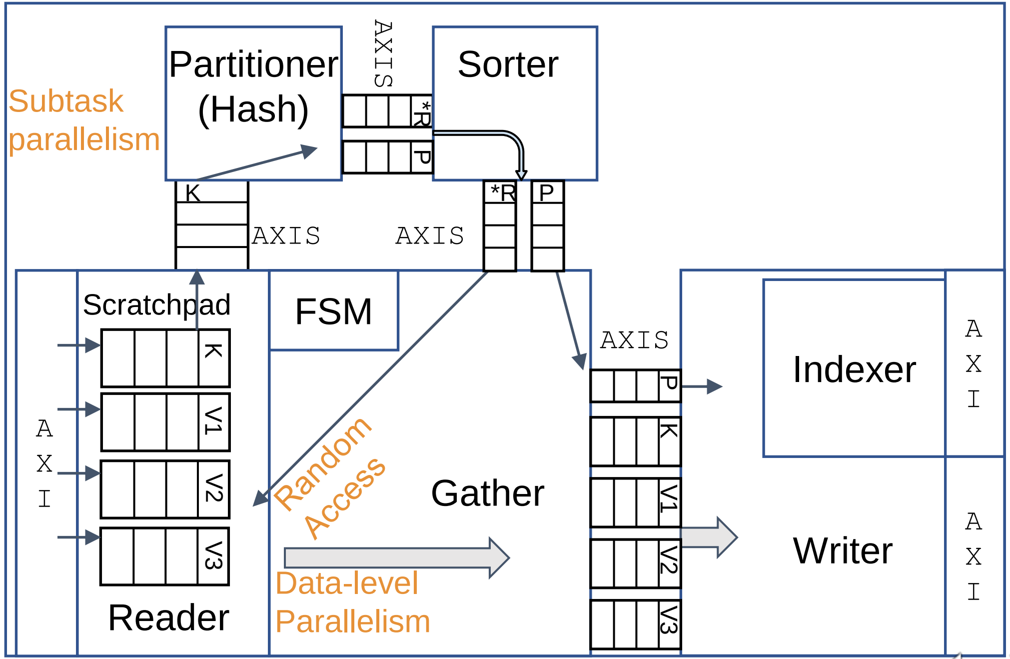
Bucketing (BypassMergeSortShuffleWriter, FPGAPart[12], Vitis[13]):

- Maintain a bucket for each destination partition to hold records going there.
- Hard to implement growable buckets without malloc-like dynamic memory allocation in hardware accelerators.
- Static bucket allocation presents linear scaling between capacity and #partitions.
- Parallelism exploitation leads to another shuffle problem.

Sort (SortShuffleWriter):

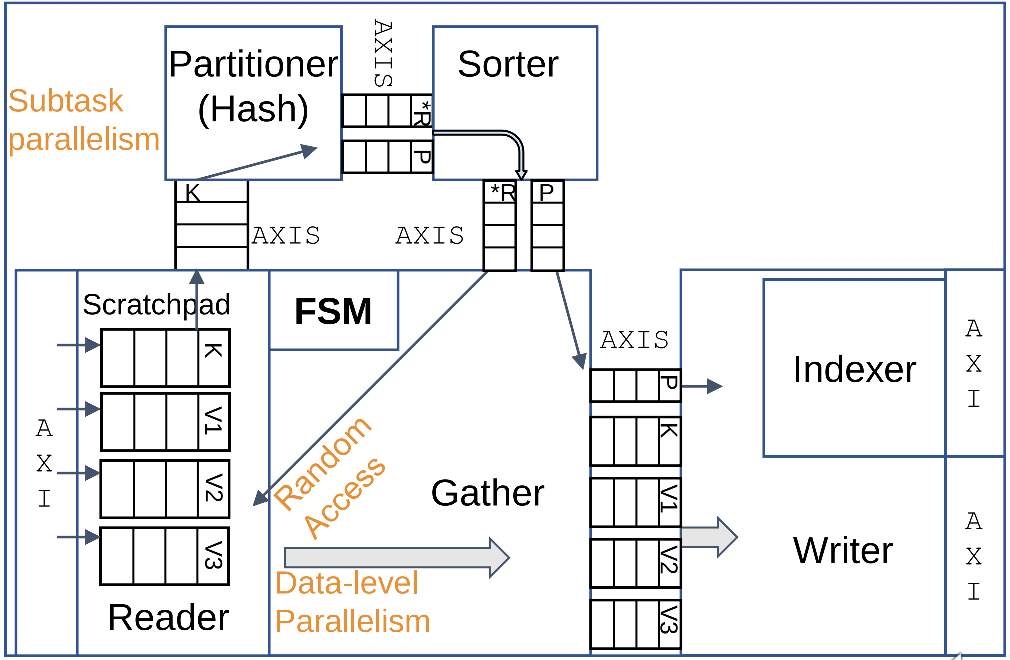
- Sort the records by their assigned destination partition ids.
- Fixed resource requirement for different #partitions.
- Further optimization of only sorting record pointers rather than full records. Gather the records with sorted record pointers later using customized scratchpads.

# Putting it together: PSACS uArch



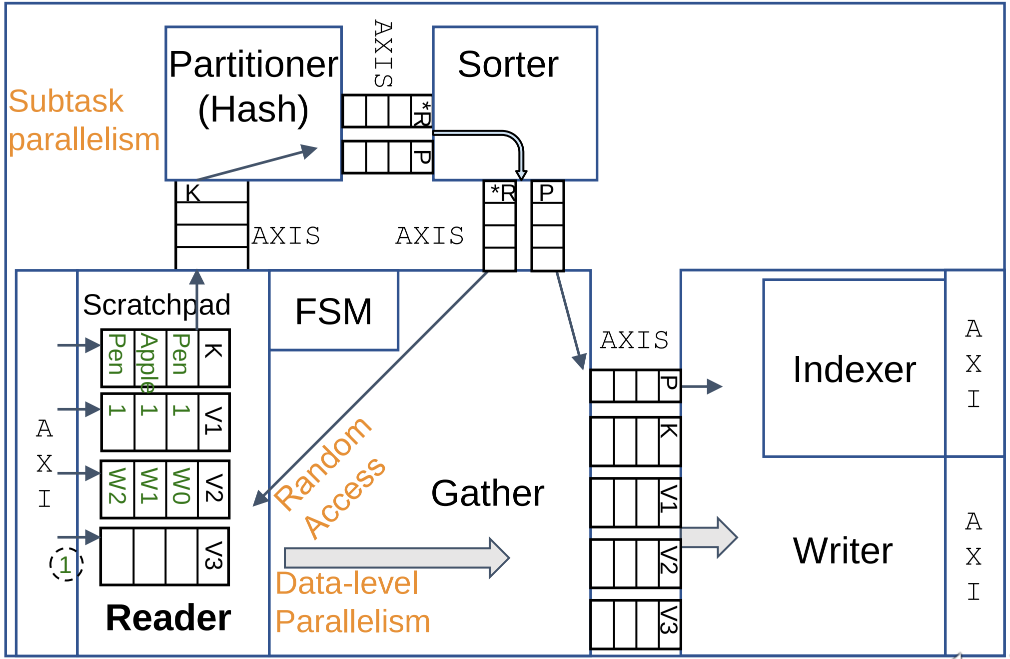
# Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.



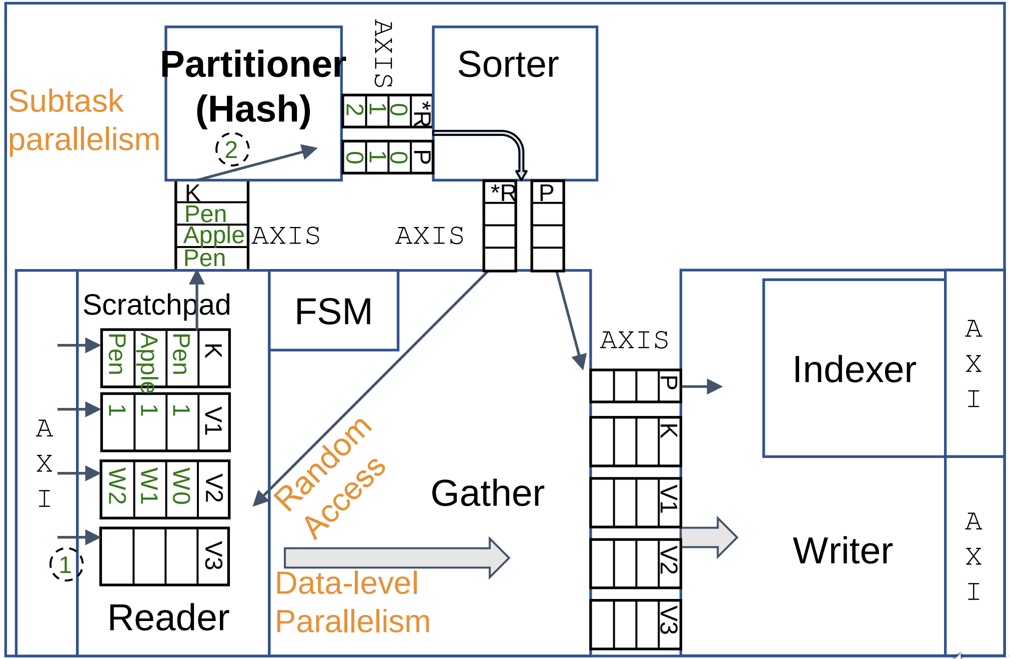
# Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.
**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.



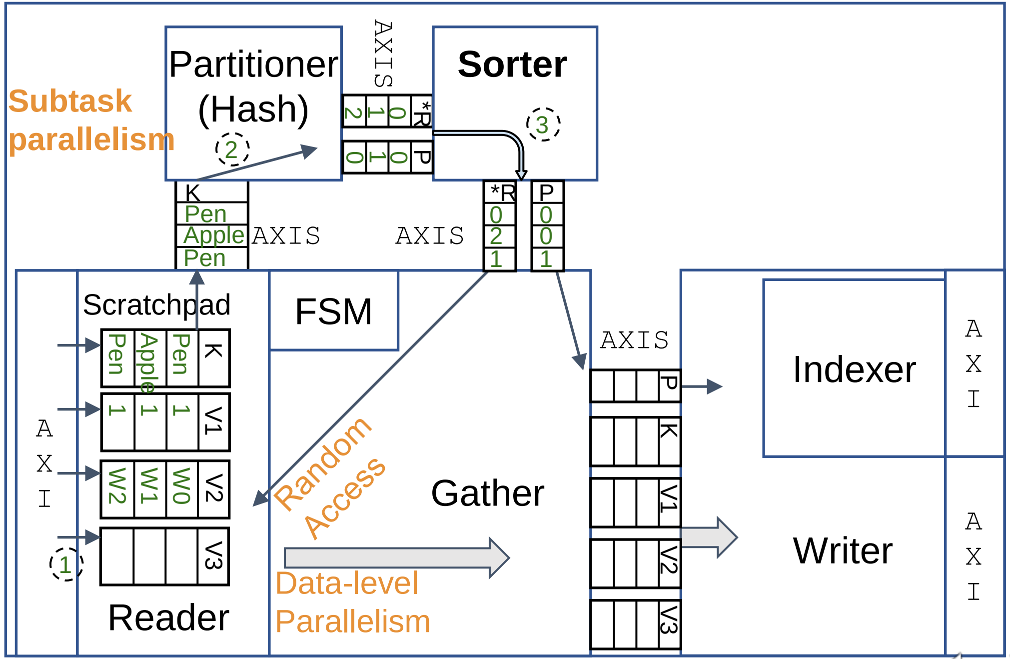
# Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.
**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.
**Partitioner**: Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).



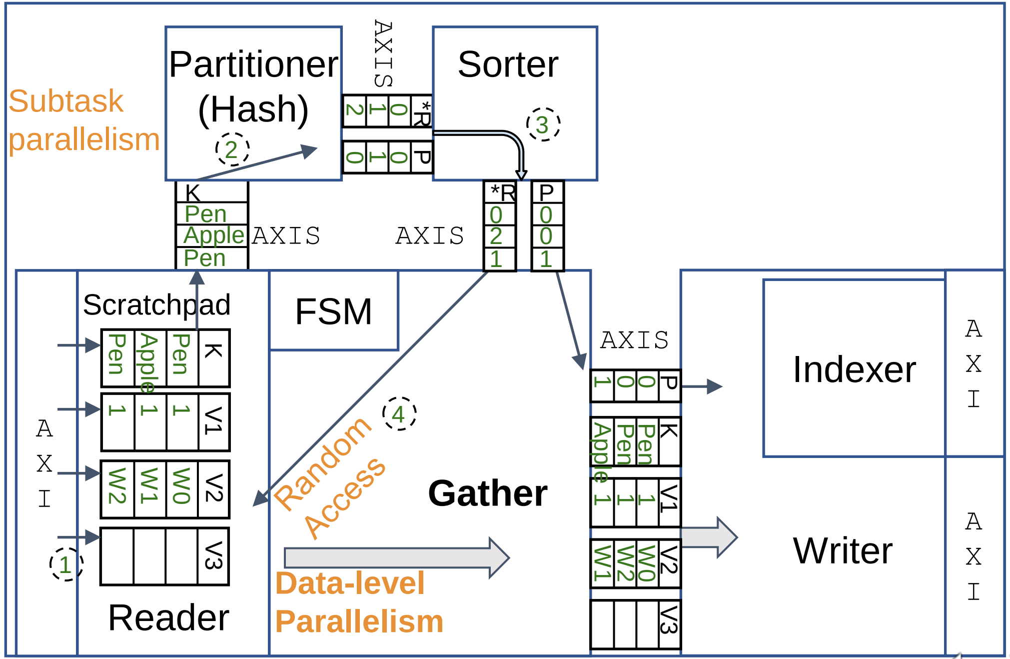
# Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.
**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.
**Partitioner**: Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).
**Sorter**: Sorts tuple (PID, RecPtr) from Partitioner by PID and stream the sorted tuples into Gather.



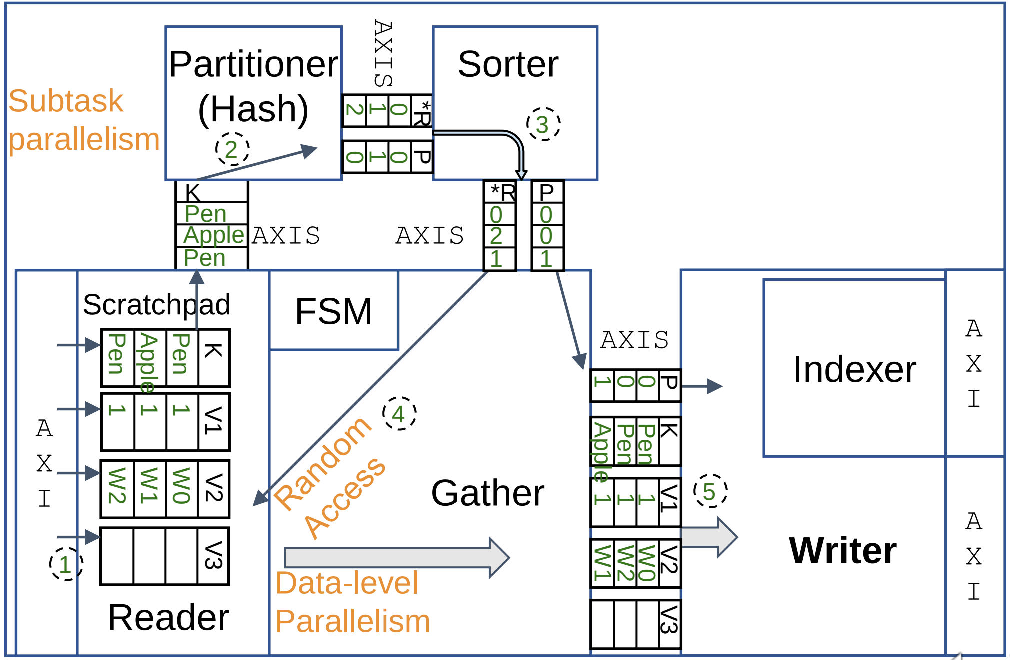
# Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.
**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.
**Partitioner**: Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).
**Sorter**: Sorts tuple (PID, RecPtr) from Partitioner by PID and stream the sorted tuples into Gather.
**Gather**: Gathers records from Reader by RecPtrs in sorted tuple streams through random accessing.

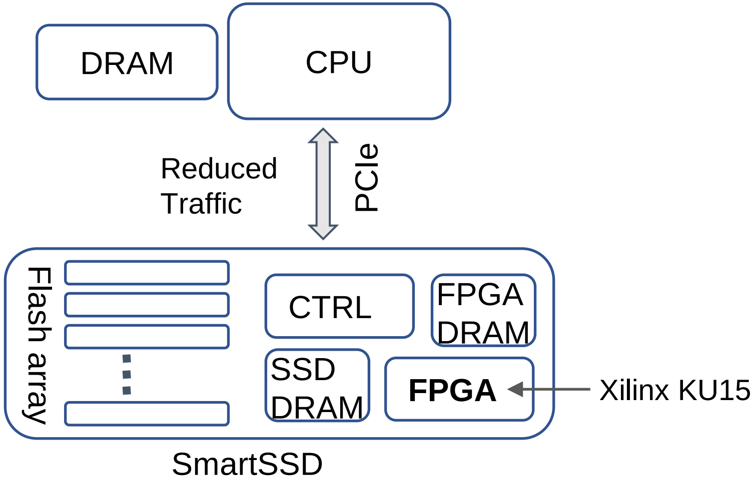


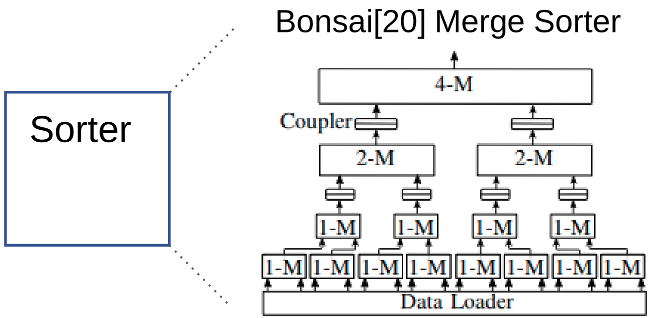
# Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.
**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.
**Partitioner**: Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).
**Sorter**: Sorts tuple (PID, RecPtr) from Partitioner by PID and stream the sorted tuples into Gather.
**Gather**: Gathers records from Reader by RecPtrs in sorted tuple streams through random accessing.
**Writer**: Stream records from Gather to DRAM, handling DRAM protocol



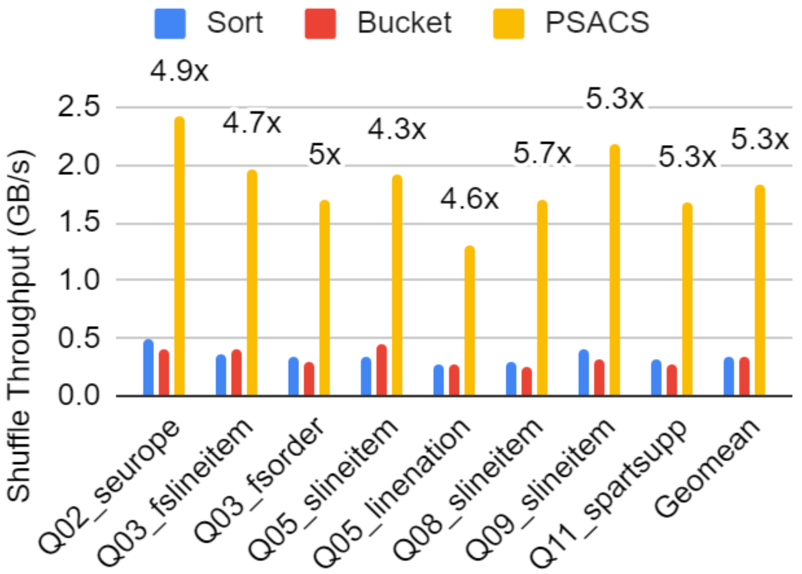
# PSACS implementation




| | LUT | FF | BRAM | URAM | DSP |
|---|---|---|---|---|---|
| Avail | 522720 | 1045440 | 984 | 128 | 1968 |
| Used | 61917 | 80885 | 433 | 64 | 0 |
| Util | 11.85% | 7.74% | 44.00% | 50.00% | 0.00% |

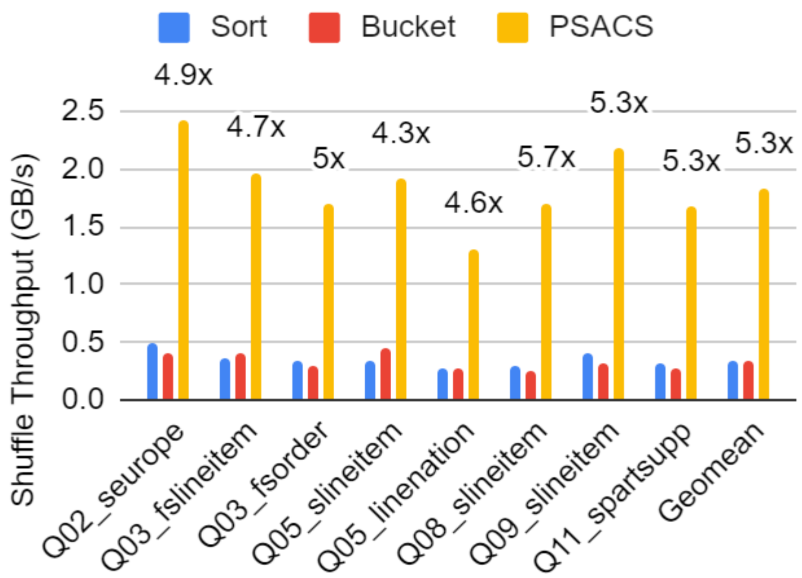
# PSACS VS hand-optimized software



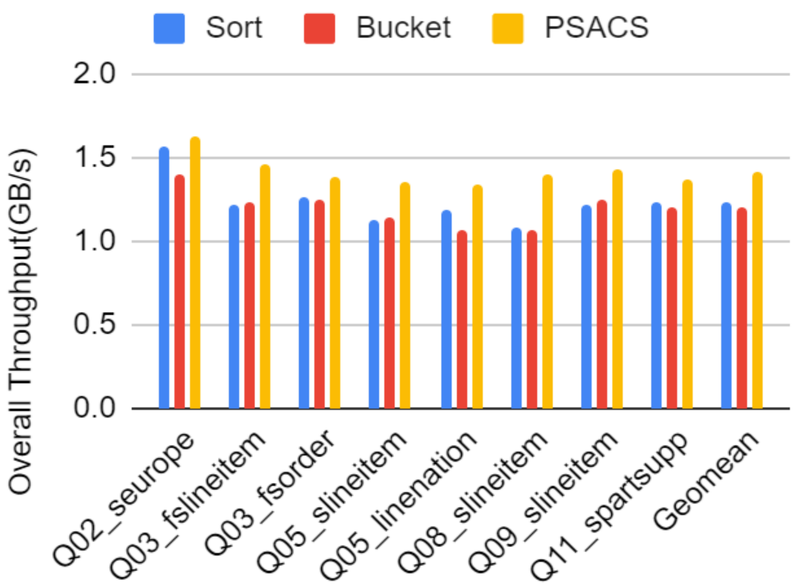

Single PSACS kernel VS single-thread software

**5x kernel-level throughput**

# PSACS VS hand-optimized software

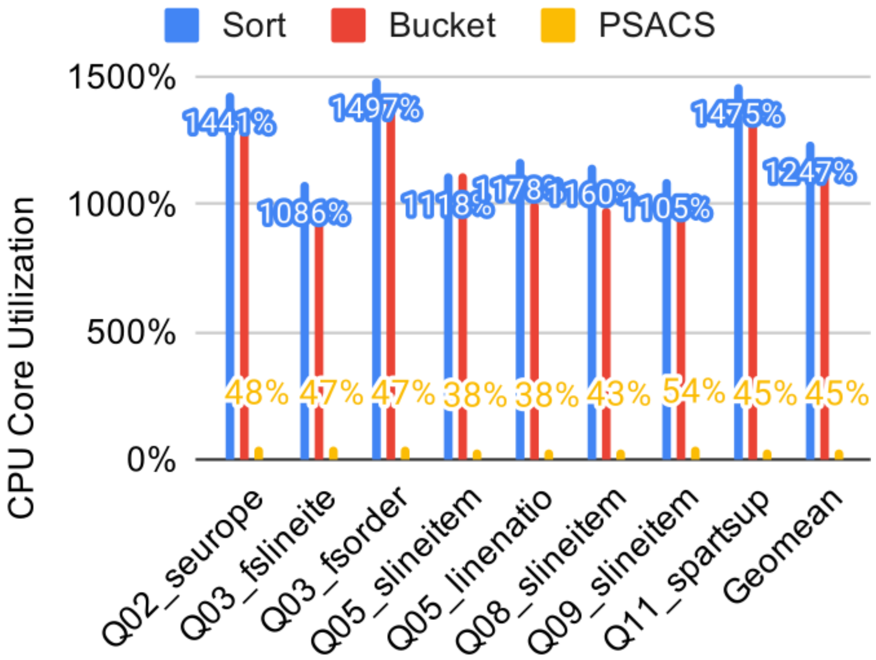



Single PSACS kernel VS single-thread software
**5x kernel-level throughput**

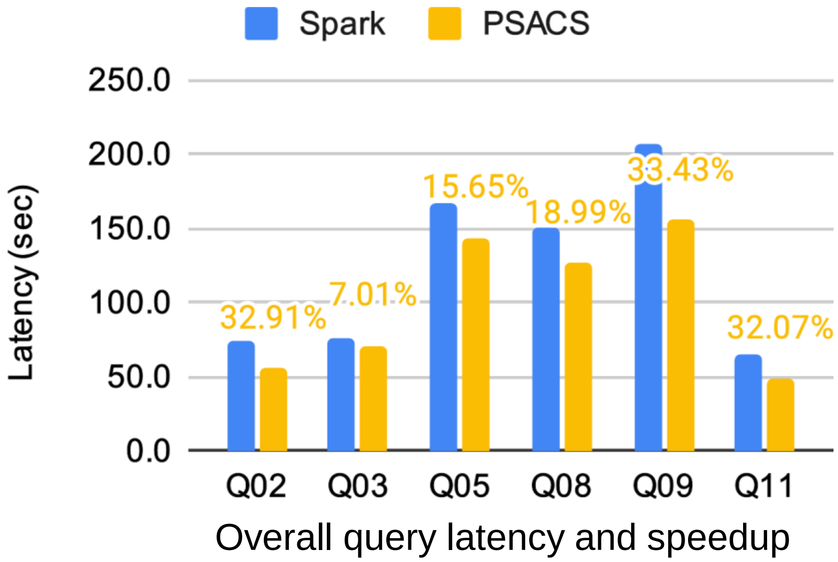



4-way-system pipeline with single PSACS VS 32-thread software (Data preparation and results writing back to storage included).
**~20% higher system-level performance**

# CPU utilization reduction and overall query speedup




**20x CPU utilization reduction**




**23% end-to-end query speedup on average**

# Related work

**Single-node partition acceleration:** Both academia[12] and industry[13]. Bucket-based grouping for single-node is used.

**Specific function acceleration in OLAP**: Cereal[23] accelerates serialization, SortAcc[14] accelerates sorting, CASM[15] accelerates local aggregation.

**Network fabric improvement**: SparkRDMA[10], SparkPMoF[11].

**Software optimization for shuffle**: Riffe[8] optimizes storage IO and network transfer with special merge policy. Intel proposes in-memory shuffle[24] in a disaggregated optane pool.

# Summary

We proposed PSACS, the first shuffle accelerator addressing the shuffle bottlenecks of the OLAP systems.

- Employs rising computational storage paradigm
- Hardware acceleration through exploitation of multiple levels of parallelism
- Utilizes custom scratchpad for high-speed gathering

PSACS delivers 5x throughput at the kernel level and on average 23% end-to-end OLAP query speedup.

See our paper for more PSACS features

- Tiled shuffling tailoring for different levels inside the memarch
- Column-major output for higher compression ratio during fetch