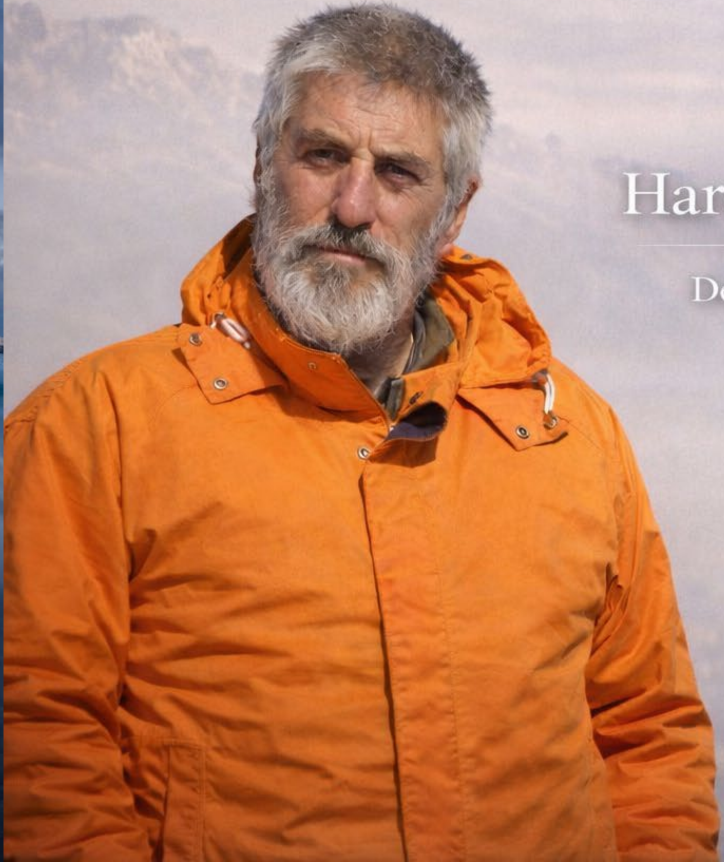
SADRAM and its Implementation

SADRAM

SYMBOLICALLY ADDRESSABLE DRAM

unstructured → SADRAM → structured

Presenting: Jason Trout
Senior Engineer & Director, SADRAM, INC and SADRAM NZ Ltd



Harold Robert George Trout

December 18, 1944 - November 1, 2025

Meet the SADRAM Staff

Jason Trout
Director,
Senior Engineer

Will Kamp
Director,
Senior FPGA/HPC Engineer

Nicolás Erdödy
Director,
CCO

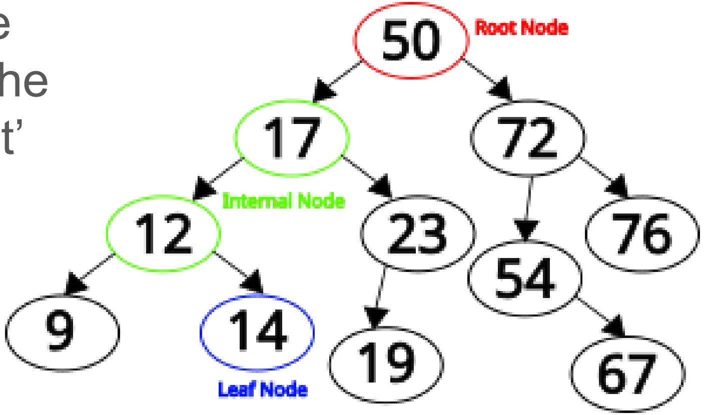
Spencer Burkett
Junior Engineer

What is SADRAM?

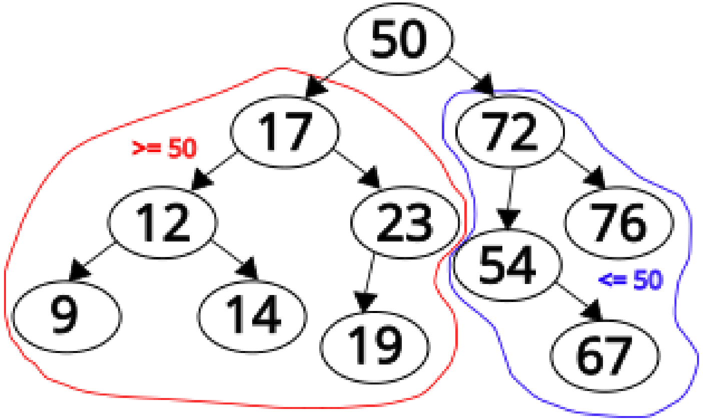
- Processing-near-memory
  - processor on same PCB/package as DDR
  - row buffer augmentations on DDR die
- Implements hardware accelerated database.
  - Insert keyed data
  - Read data associated with key
  - Read data by position in the “sorted order”
- Advantages?
  - Near memory: speed improvements from shorter and fewer I/Os from processor to memory
  - Autonomous: Runs as a co-processor. Data must be provided and harvested, that’s it
  - Databases are ubiquitous: 85% of developers work heavily with *some* manner of database

Binary Search Tree Properties

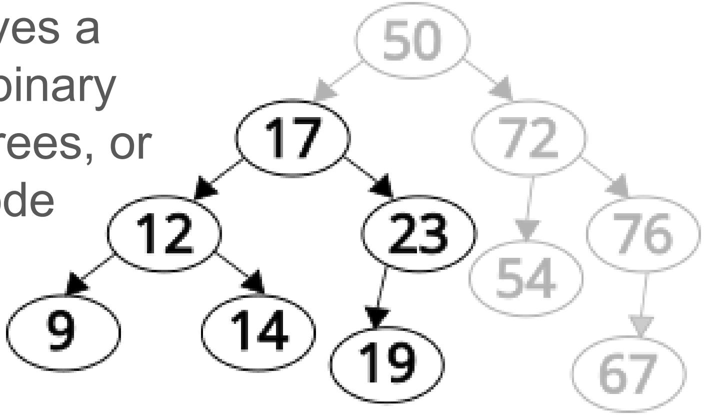
Root node contains the set's 'pivot' point, i.e. the set's median value

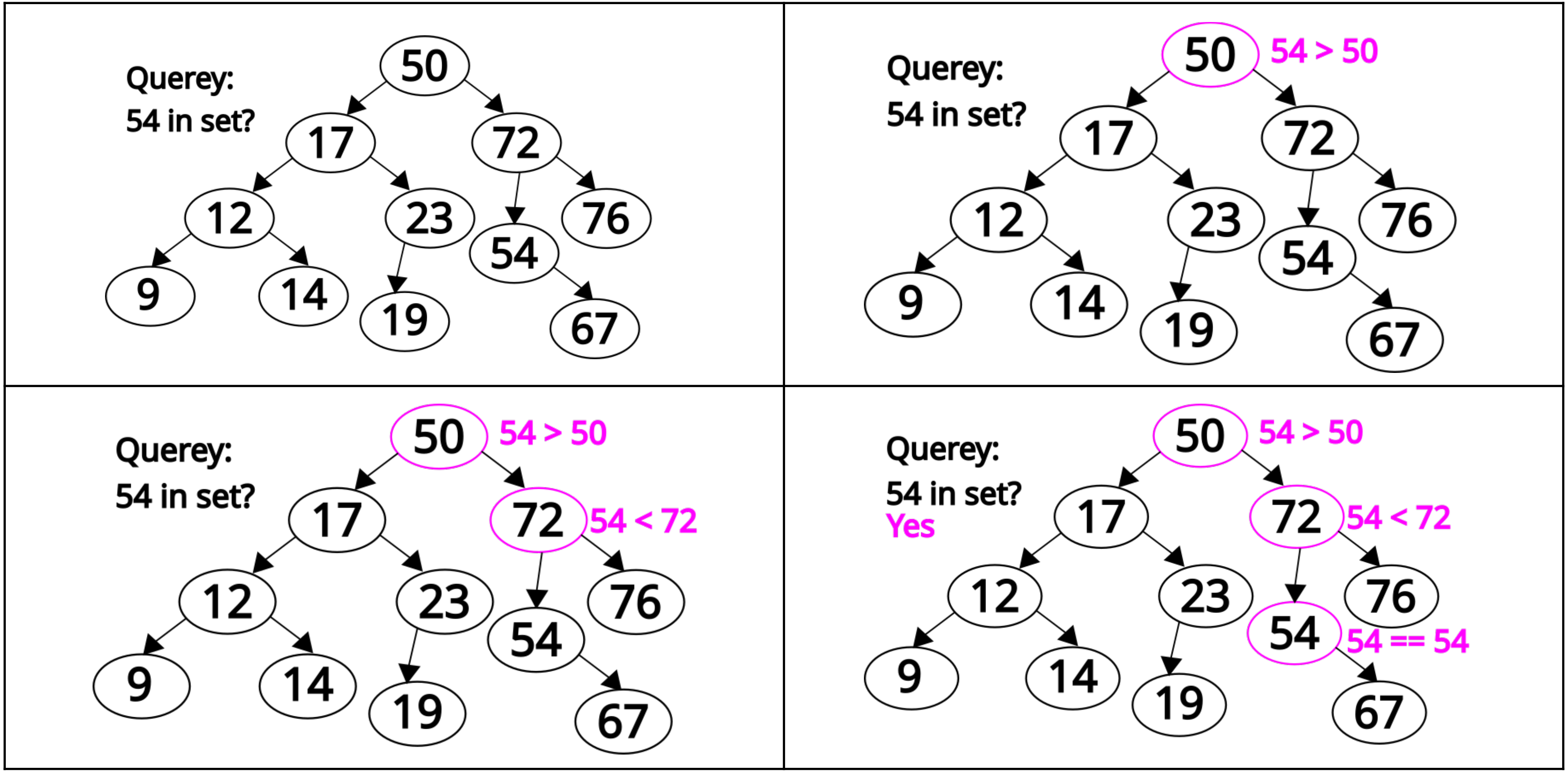


All nodes to the left are <= root, all nodes to the right are >= root



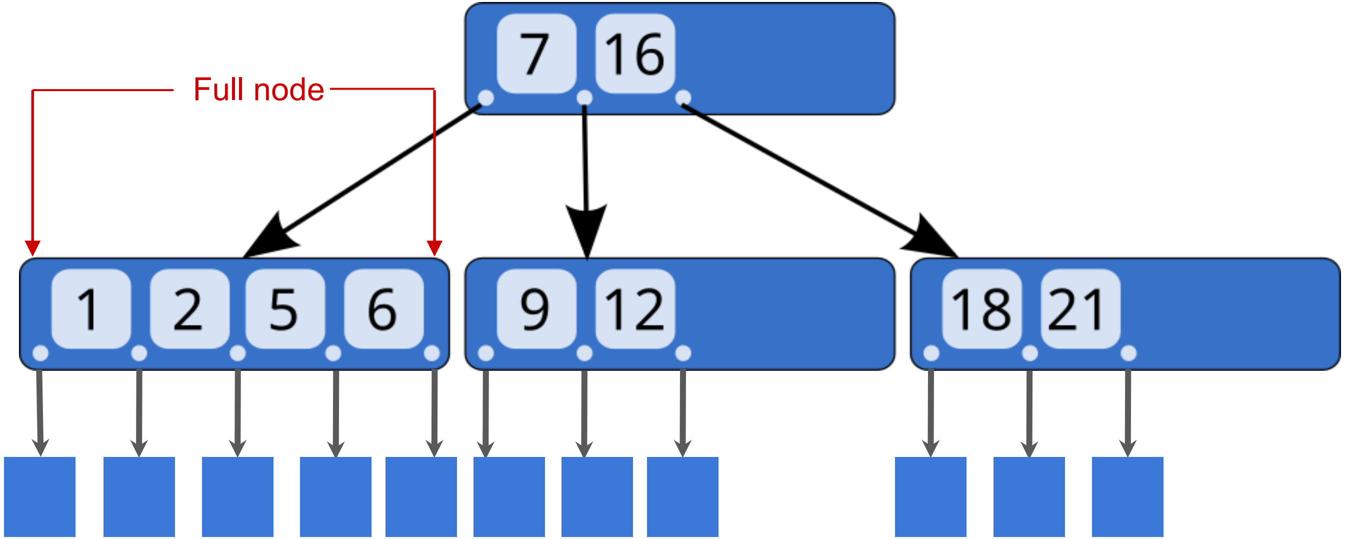
Subnodes are themselves a smaller binary search trees, or a leaf node



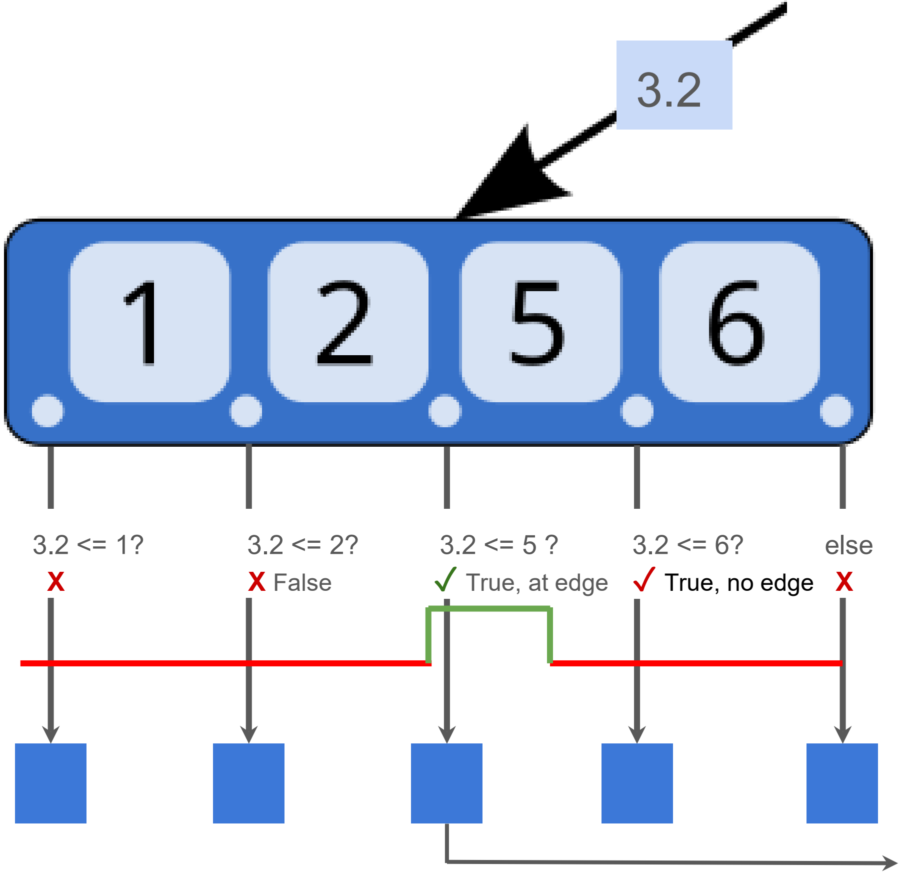


When writing or reading (shown), binary search trees can be traversed in $O(\log_2(N))$ time, instead of $O(N)$ time.

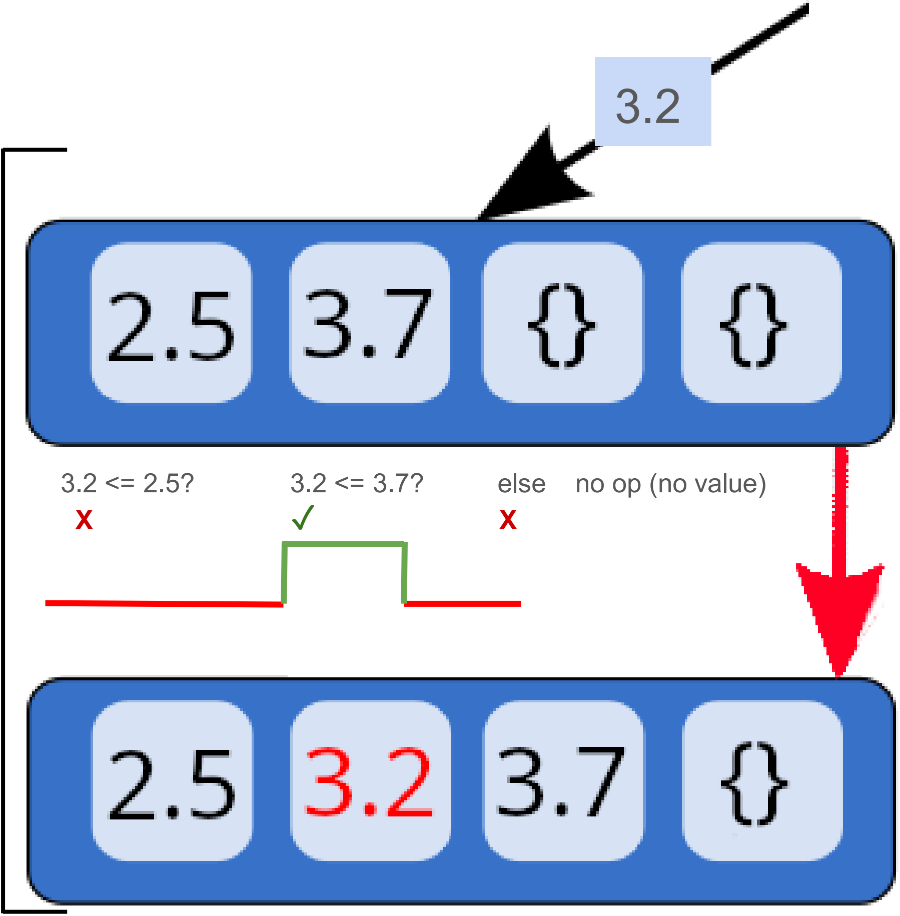
B-Trees



Inserting 3.2
Traversing internal node



Inserting 3.2
Inserting value into leaf node

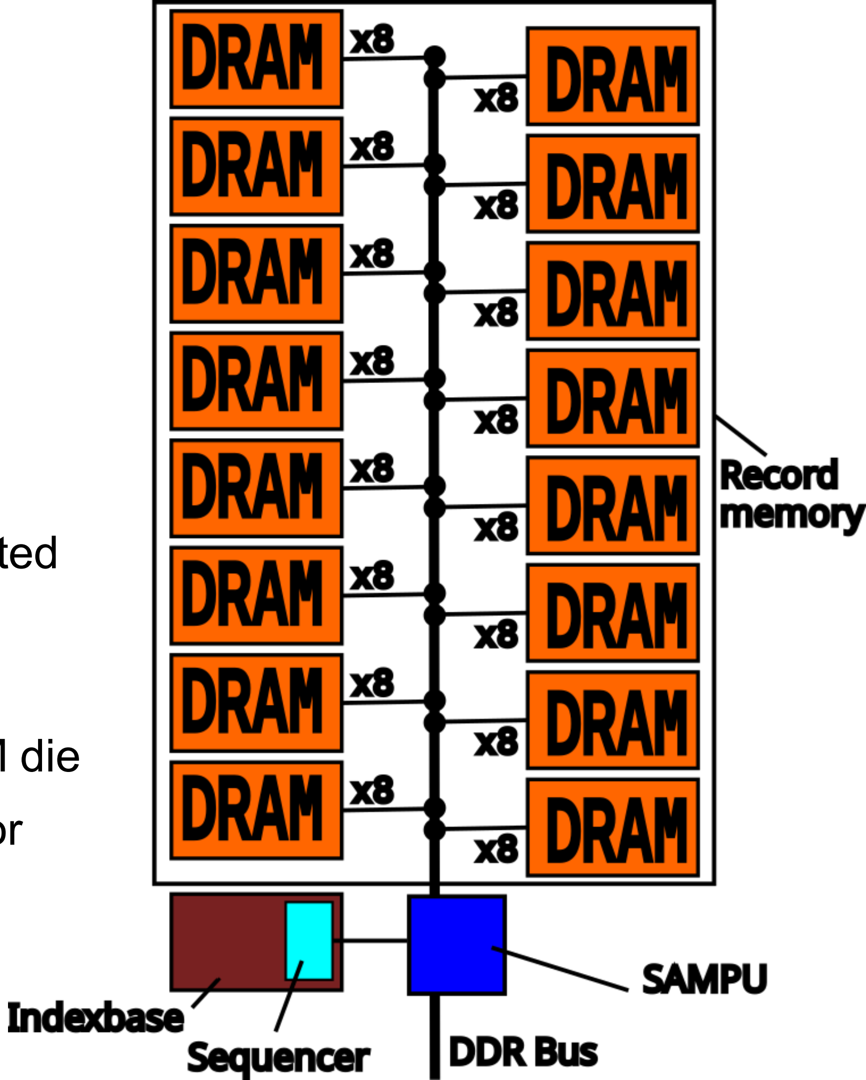


B-Trees limitations

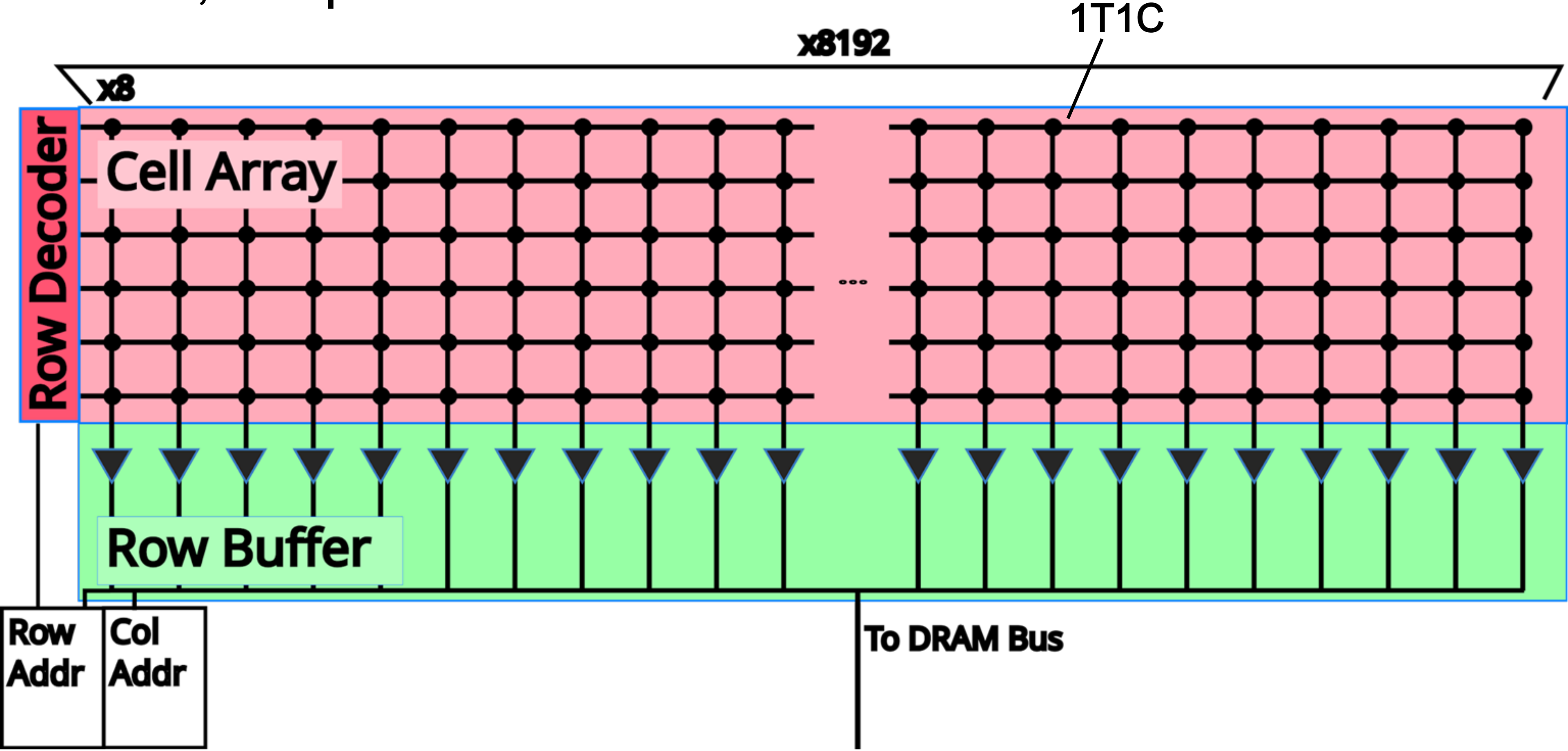
- Memory wall-limited
- Minimizes trips to hardware
  - For disk-backed databases, node size chosen to fill a disk page
  - For DRAM-backed databases, node size chosen to fit in processor cache
- Sorts minimum amounts of data; keys, not records

SADRAM, 1000 foot view

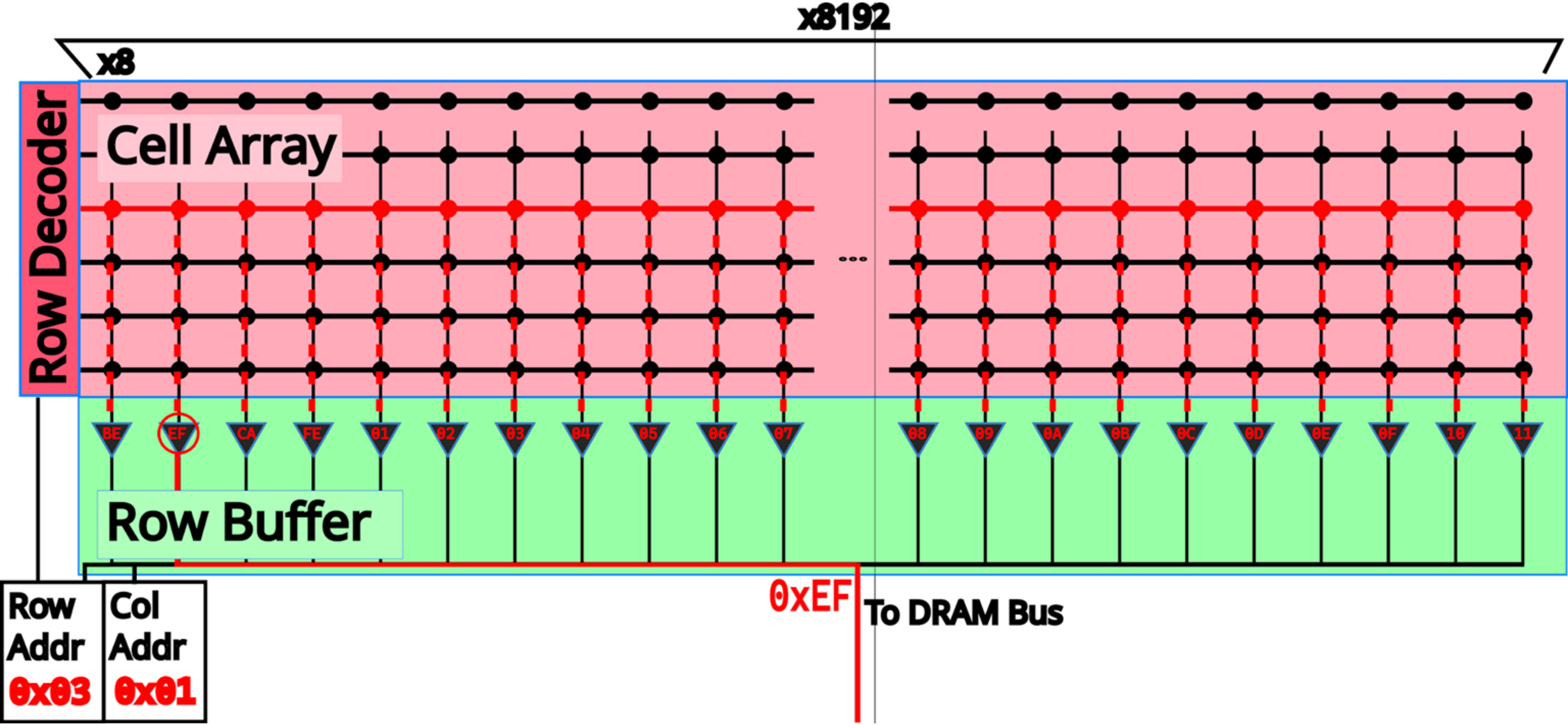
- SADRAM is augmented DRAM
- PCIe/CXL for now, ideally, DIMM slots
- SADRAM triggered by writing to DRAM
  - Key identified within record in-flight
  - Key copied, added to Indexbase in sorted order
- Three major components
  - Sequencer: specialized logic on DRAM die
  - SAMPU: General purpose co-processor
  - Record memory: Standard DRAM



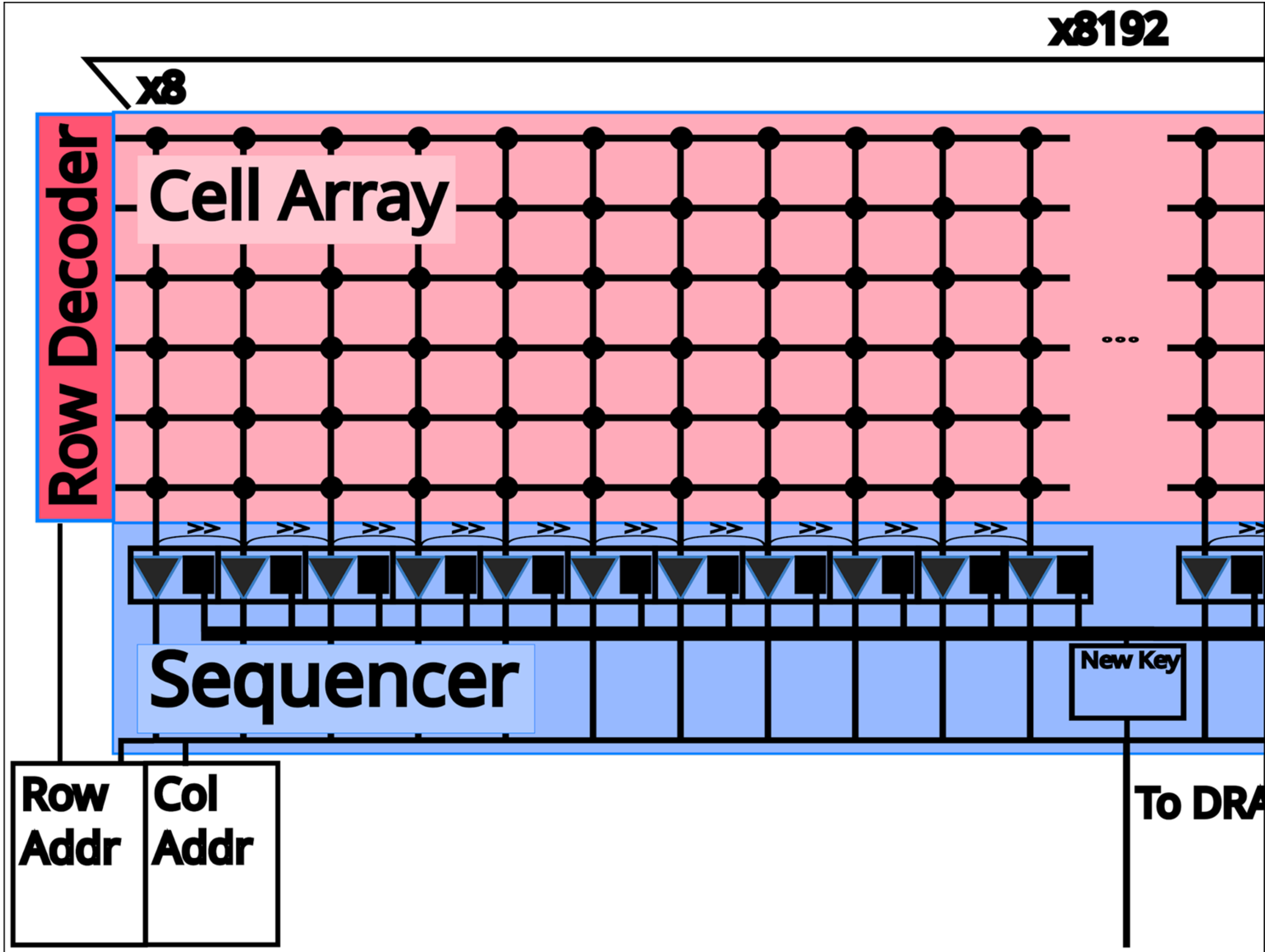
DRAM, simplified



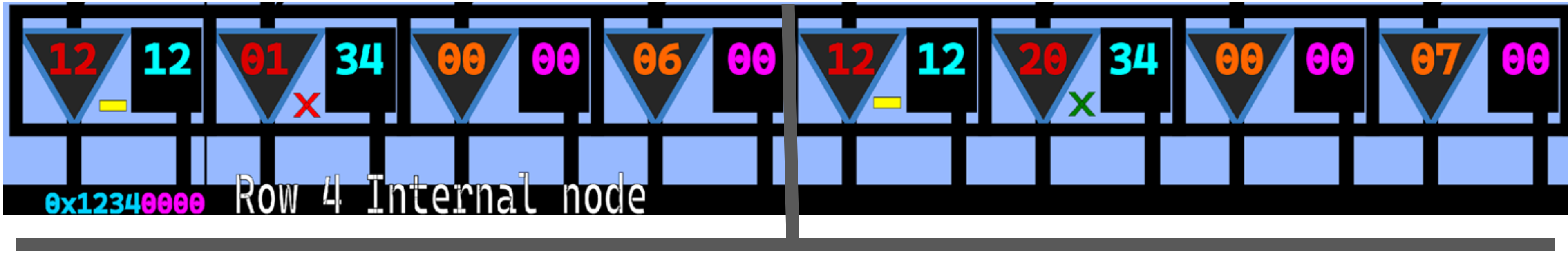
A DRAM read


x8192
x8
Row Decoder
Cell Array
BE
EF
CA
FE
01
02
03
04
05
06
07
08
09
0A
0B
0C
0D
0E
0F
10
11
Row Buffer
Row Addr
0x03
Col Addr
0x01
0xEF
To DRAM Bus

SADRAM Sequencer: row buffer augmentation



SADRAM Key Insertion: compare, internal node



Cell group 1: key 0x1201, pointer 0x0006

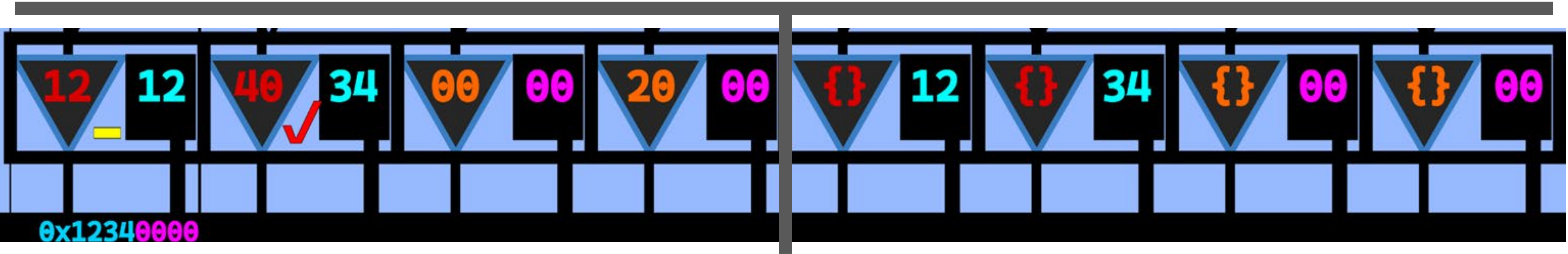
Cell group 2: key 0x1220, pointer 0x0007

```
       Type Legend
Cell group key     : Red
Cell group pointer: Orange
       New key     : Light Blue
       New pointer: Pink
```

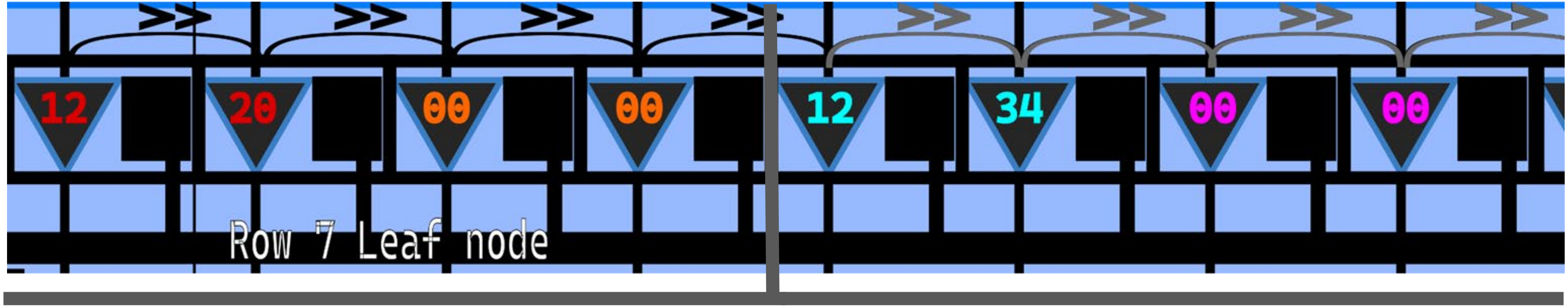
New Key-Pointer: key 0x1234: pointer 0x0000

Cell group 3: key 0x1240, pointer 0x0020

Cell group 4: No entry, data not valid



SADRAM Key Insertion: insert, leaf node



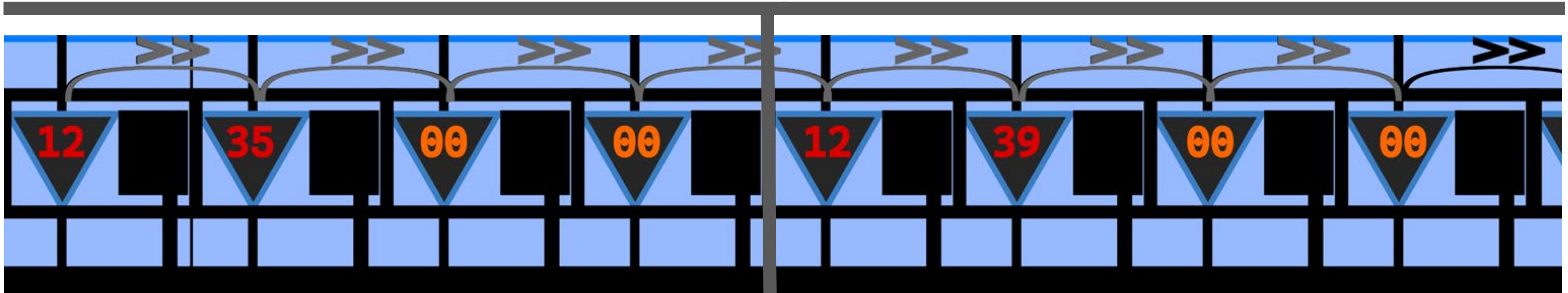
Cell group 1: key 0x1220, pointer 0x0000

Cell group 2: key 0x1234, pointer 0x0000, new value

```
        Type Legend
Cell group key     : Red      New key     : Light Blue
Cell group pointer: Orange   New pointer: Pink
```

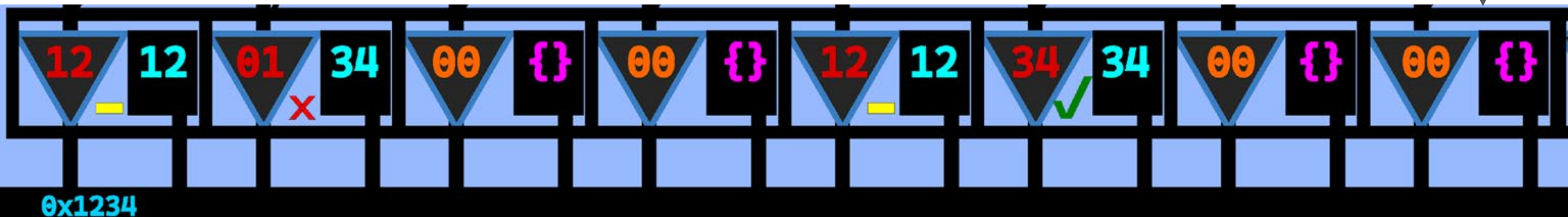
Cell group 3: key 0x1239, pointer 0x0000, from 2

Cell group 4: key 0x1239, pointer 0x0000, from 3

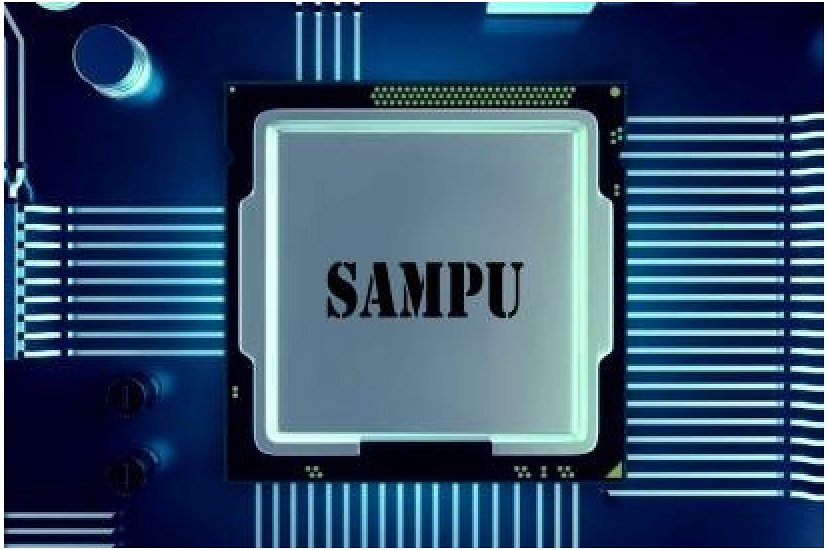


Other Sequencer operations

- Read-by-key: Given a key,
  - if it exists in the Indexbase, return its associated pointer
  - If not, return 0
- Read-by-index: Given an index (a location in the “sorted order”),
  - return key and pointer at the index
  - return 0 if requested index out of range
- Delete
  - Key-pointer pair marked as invalid,
  - no other immediate Indexbase changes
  - Deleted keys removed during idle periods



SADRAM Processor: SAMPU



- Patented lightweight processor
- Specs
  - Single core, 100 MHz clock, 1.25 MB program memory, 4Kb stack, debug controls
  - 16-bit instruction set, supporting basic math, functions, loops, conditionals, sequencer interface
- Supporting software
  - GUI debugger
  - Compiler, SAMCODE to SAMPU assembler instructions

SADRAM Applications

- Local Database
  - Read-by-key and Key insertion implement a locally available database.
  - Ubiquity of database applications means big multiplier on SADRAM impact.

- Sorting
  - Pointer-key pairs are sorted within Indexbase. Read-by-index recovers data in the sorted order.
  - Sorts as it stores

- Increase effective memory density
  - Read-by-key means hash tables are perfectly efficient.
  - Sparse array/matrix storage; don’t store zeros, pack non zeros tightly

- Speed and power improvements.
  - Adoption of processor caches in the 80s led to dramatic performance improvements.
  - SADRAM similarly positioned.

Current activities

- Porting design to Alveo
  - Emulate 8192/16384-bit wide memory buses
  - Bring over the SAMPU, adapt for wider buses

Longer term plans

- SQLite
  - modified to work with SADRAM,
  - generate performance statistics against existing solutions
- Architectural improvements
  - SAMPU row cache
  - Sequencer multi-bank access
- Investigate RISC-V based SAMPU variant

MCW Collaboration

- How much logic can we put in the Sequencer?
  - Simultaneous compare vs binary search, similar situation with shift logic. **What can we get away with?**
  - Can we have right shift **and** left shift? Circular row buffer?
  - eDRAM: logic-fab-created DRAM?
- Where is the SAMPU relative to the Sequencer?
  - Same die? Same package, different die? Common PCB? **What packaging makes sense?**
  - Sequencer *must* be on die with DRAM cells.
  - SAMPU would benefit from a 8192/16384 interface to Sequencer, but not required.
- How much of the design can be reuse?
  - Use existing DRAM chiplet as base for Sequencer?
  - Leverage existing RISC-V chiplet as base for SAMPU?
  - **When do we need to start thinking about where & how to get chiplet(s)?**

Closing Remarks

- SADRAM is augmented DRAM; database access in local hardware.
- Databases are common; dedicated hardware like SADRAM could make a big impact.
- SADRAM needs guidance and industry partners to be realized.

Thank you, MCW 2026!

Row Splitting

Triggers when the last subgroup in a node fills.

Moves 50% of data of full row into newly allocated row, adds entry to parent node.

Can trigger recursively; if a path through the tree contains a series of “about to fill” nodes ending in a leaf node, one addition to that leaf node would result in a cascade of splits. Rare, but time consuming.

Initiated and performed by the SAMPU when it detects a full node.

Cannot be performed if no SADRAM rows allocatable (B-Tree style tree balancing can be performed instead).

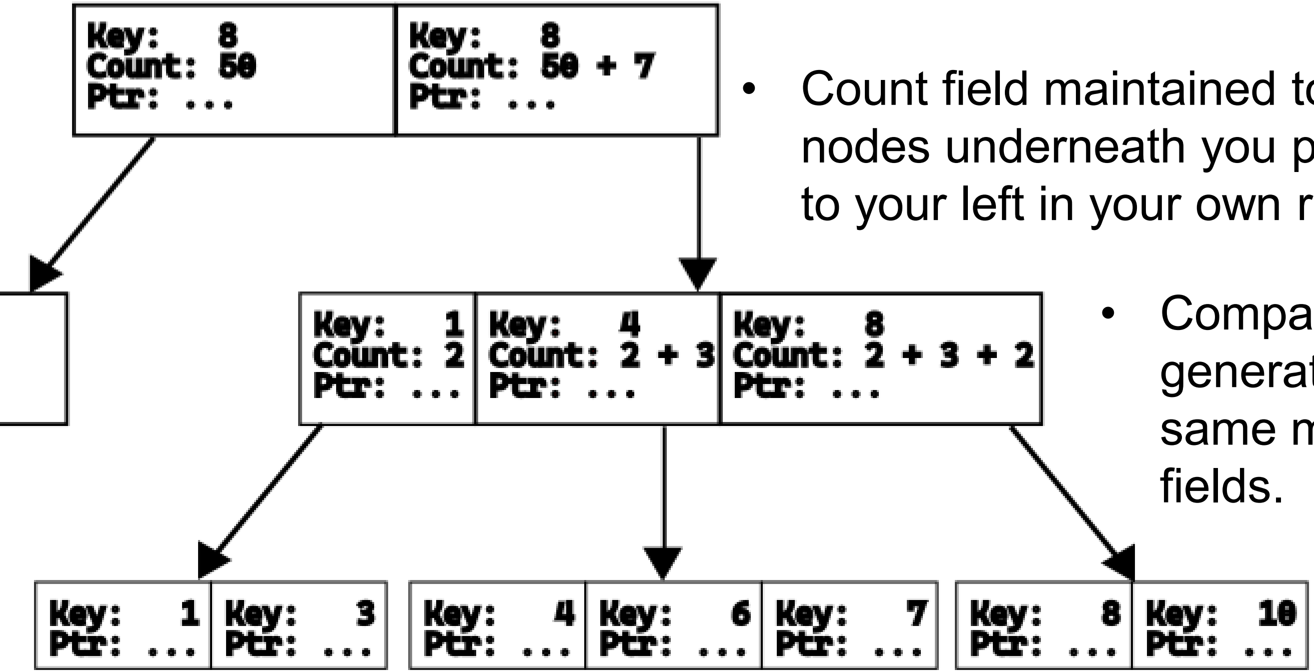
Read by Index

- Neither the Indexbase nor record memory is sorted if read like a standard array.

- Internal nodes (not leaf nodes) have a “count” field. They store key-pointer-count triads.

- Count field maintained to be the number of leaf nodes underneath you plus count fields from triads to your left in your own row.

- Comparison result “edges” can be generated with count fields in the same manner they are with key fields.



B-Trees limitations

- Memory wall-limited
- Minimizes trips to hardware
    - For disk-backed databases, node size chosen to fill a disk page.
    - For DRAM-backed databases, node size chosen to fit in processor cache.
- Sorts minimum amounts of data; keys, not records.

SADRAM: Symbolically Addressed DRAM

- Write records to SADRAM.
  - Triggers key recognition and extraction: **key-insertion**.
  - Key copies sorted and stored in **Indexbase**.
- Read from SADRAM as you would a database
  - Present key to SADRAM: **read-by-key.**
  - Sadram provides pointer to record
- Key Insertion + read-by-key = Symbolically addressable
  - Symbolically addressable DRAM.
  - S. A. DRAM.
  - SADRAM