Affinity Alloc: Taming ~~Not-So~~ Near-Data Computing

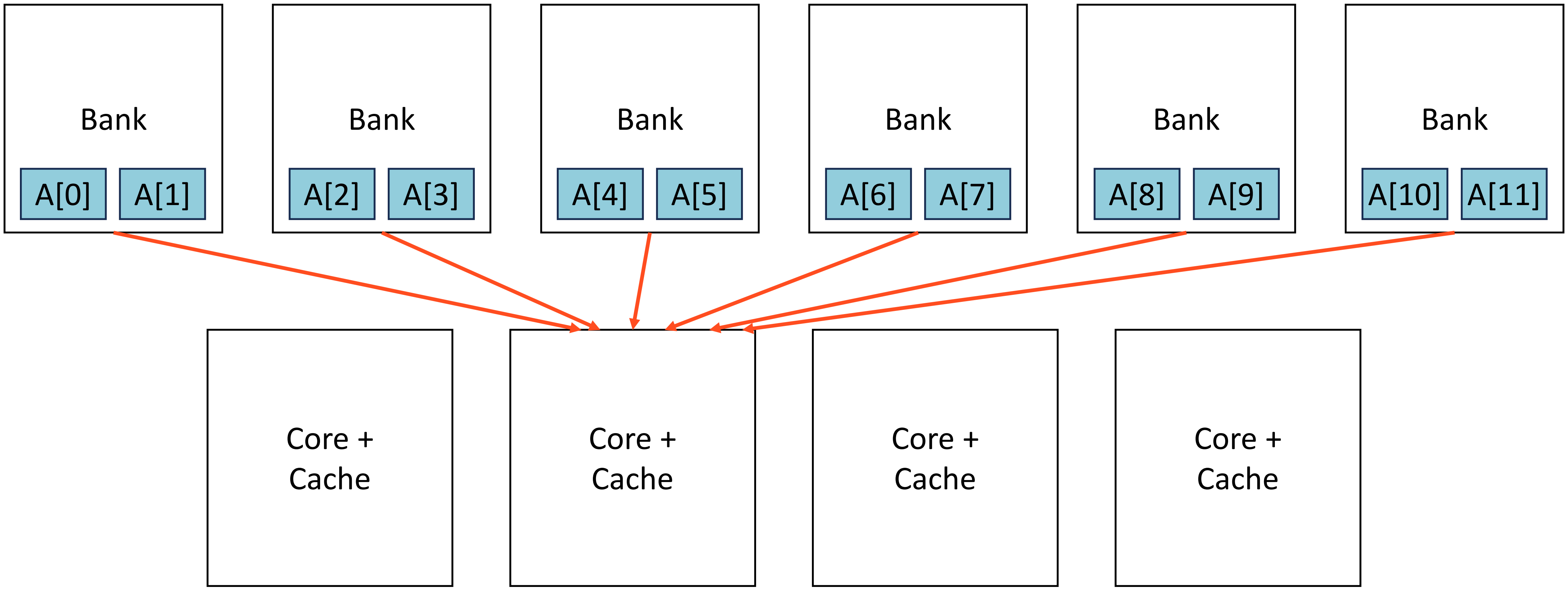
Zhengrong Wang[1], Christopher Liu[1], Nathan Beckmann[2], Tony Nowatzki[1]

[1]UCLA, [2]CMU

October. 2023

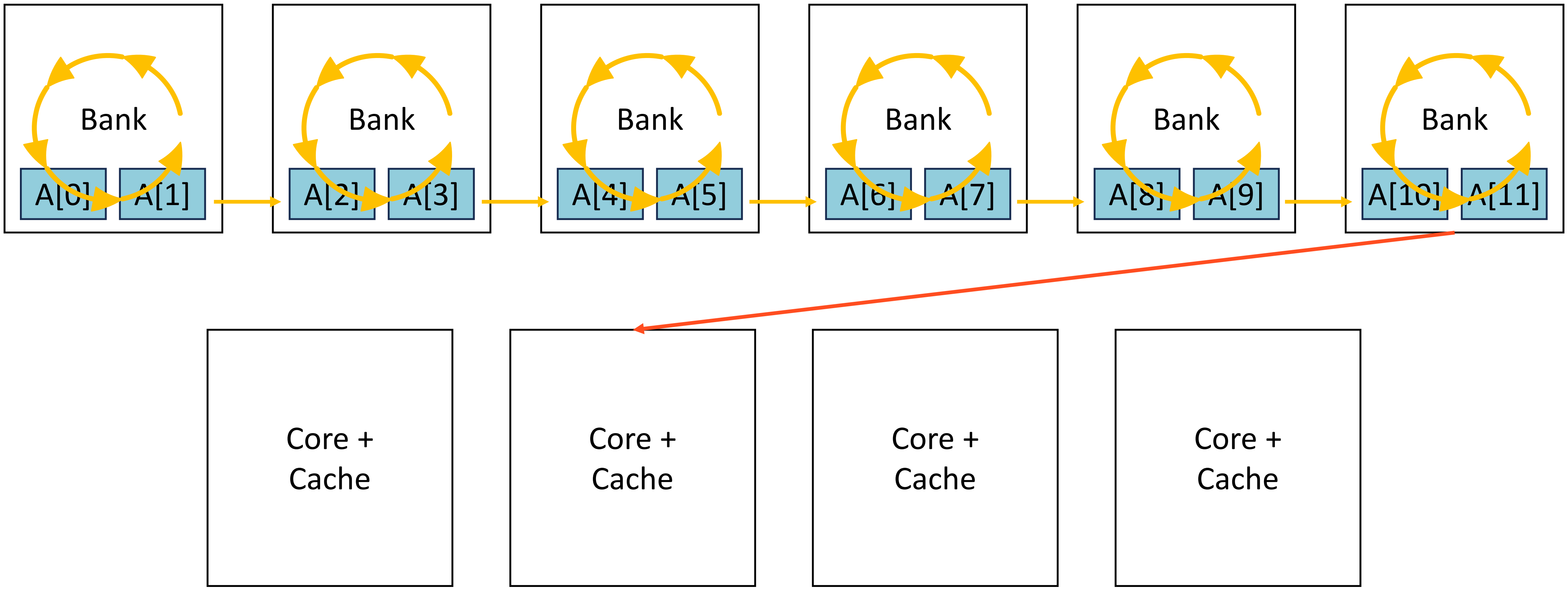
Conventional Computing

sum(A[i])



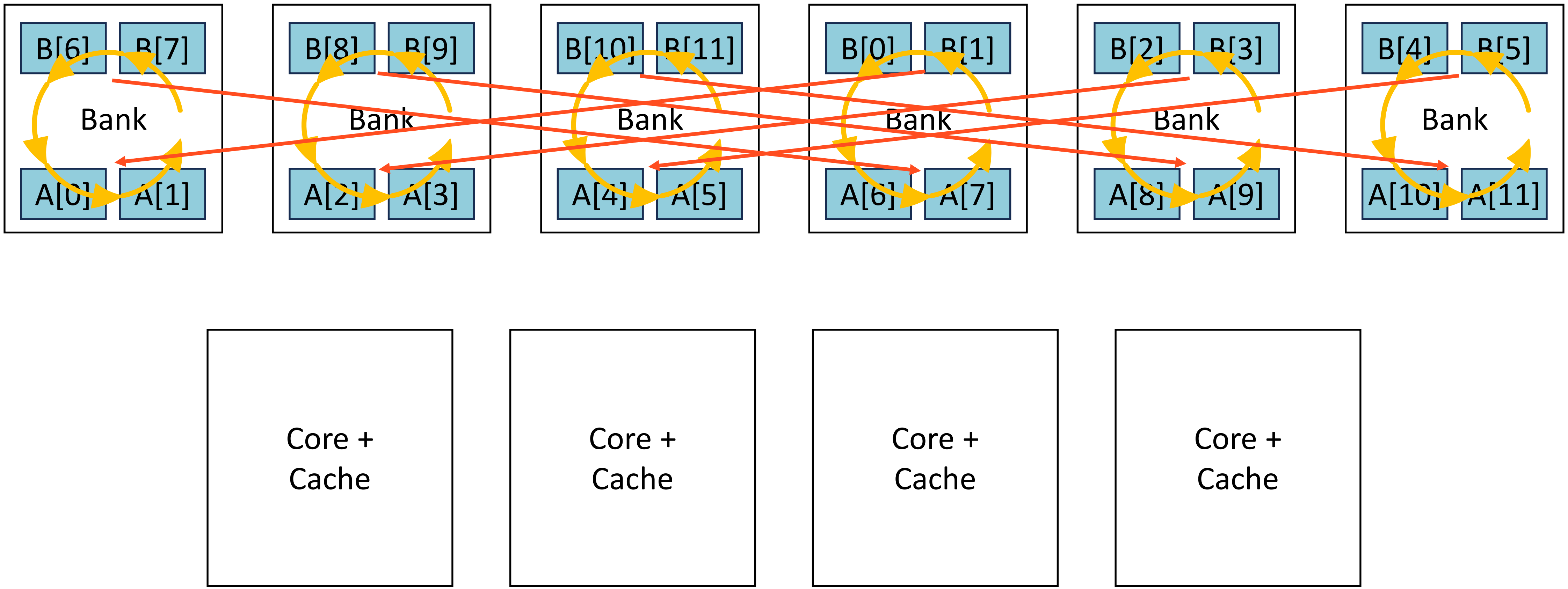
Near-Data Computing

sum(A[i])

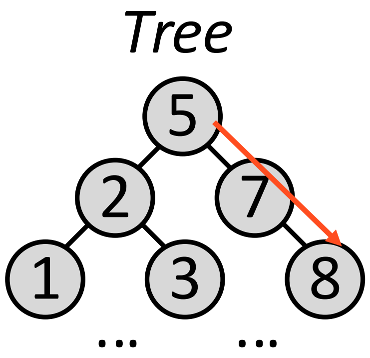


Not-so Near-Data Computing

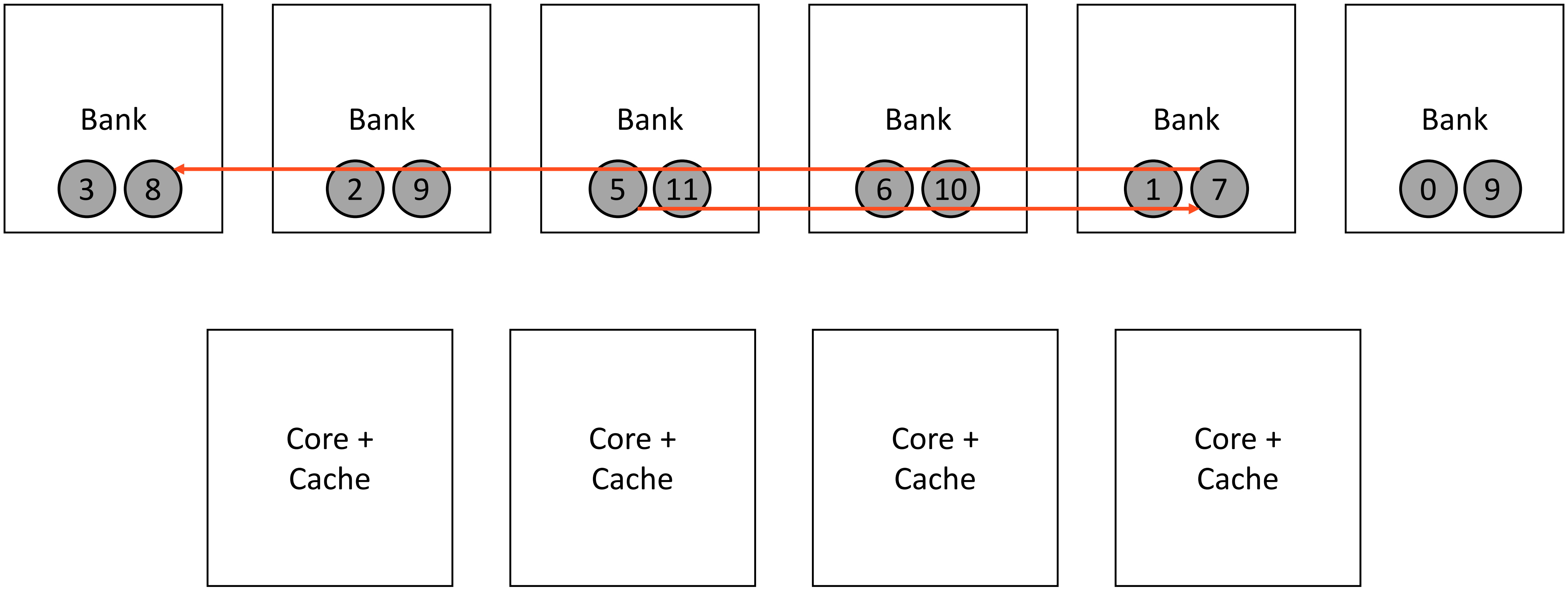
sum(A[i]*B[i])



Not-so Near-Data Computing

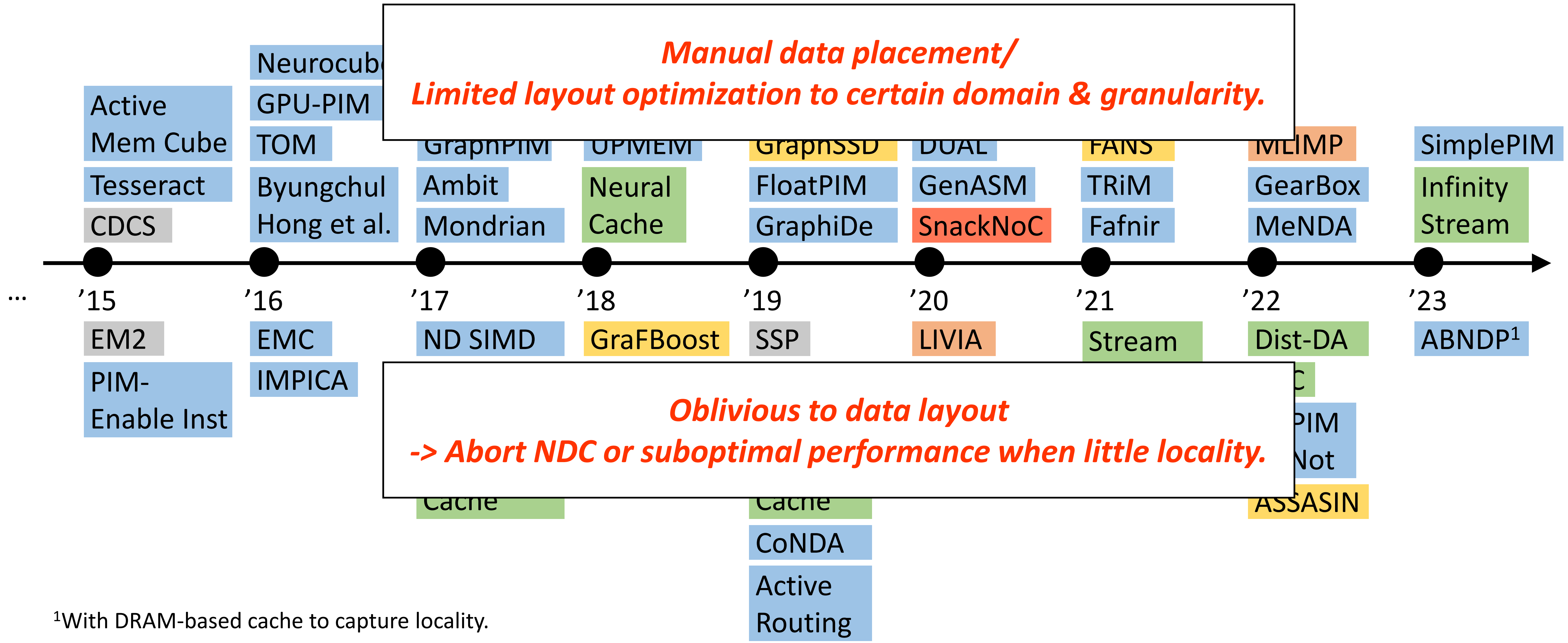


Tree Traversal



Prior Works

Core SRAM NoC DRAM/HMC SSD Multi-Level



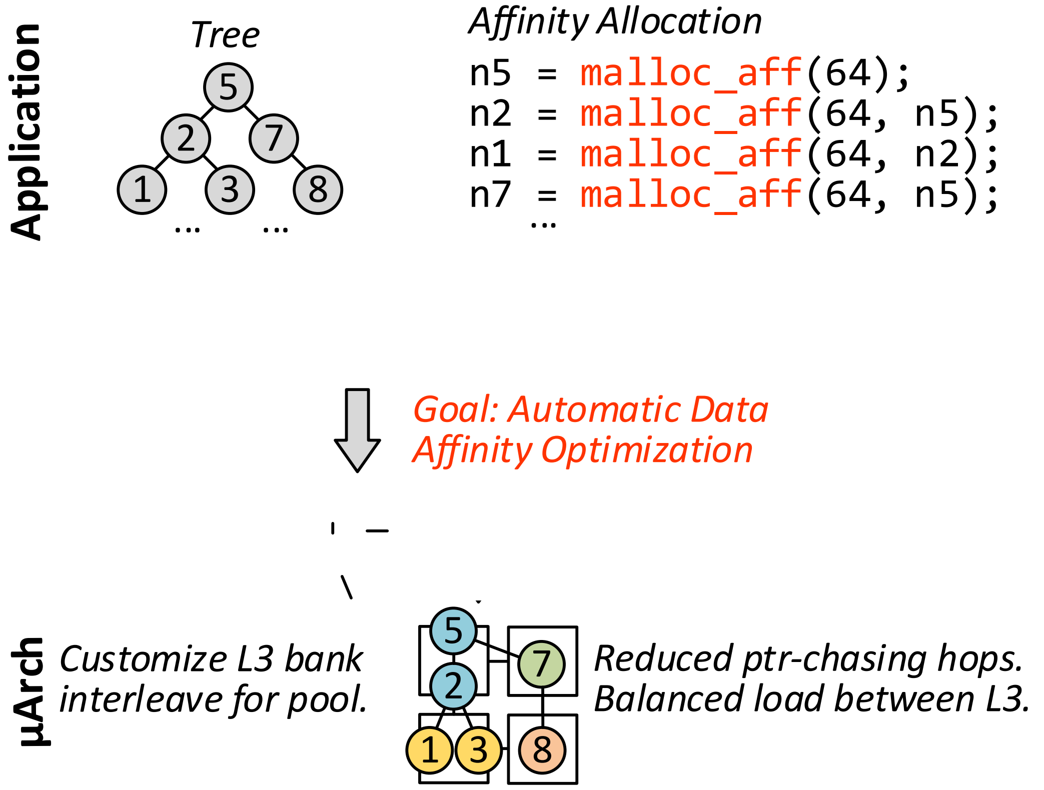
[1]With DRAM-based cache to capture locality.

Goal: Automatic Data Affinity Optimization

- From various data structures…
- To automatic optimized layout in μArch

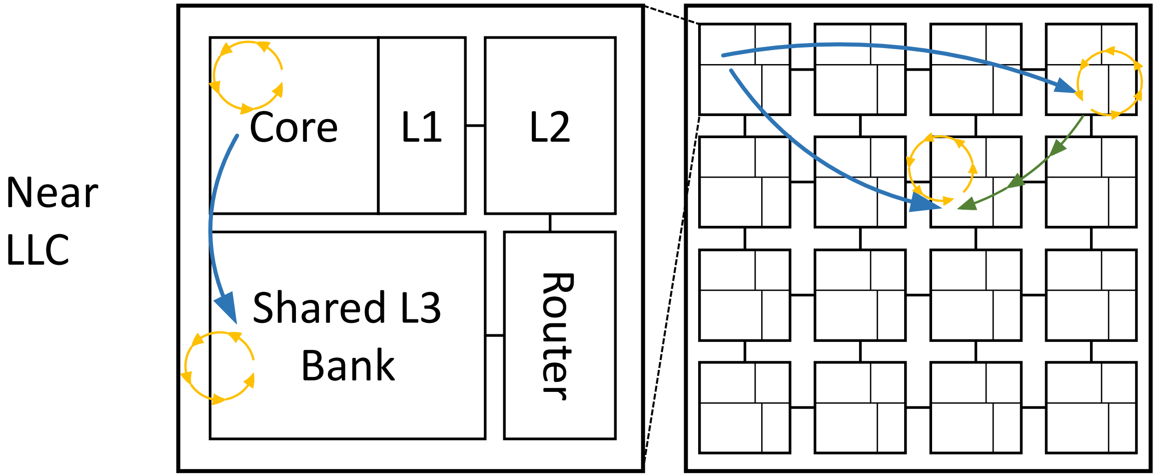
- Key Insight:
  - All data structures have affinity relationships
  - This information is independent of hardware
  - Relationships are available at allocation time

- Approach: Expose affinity info to allocator.

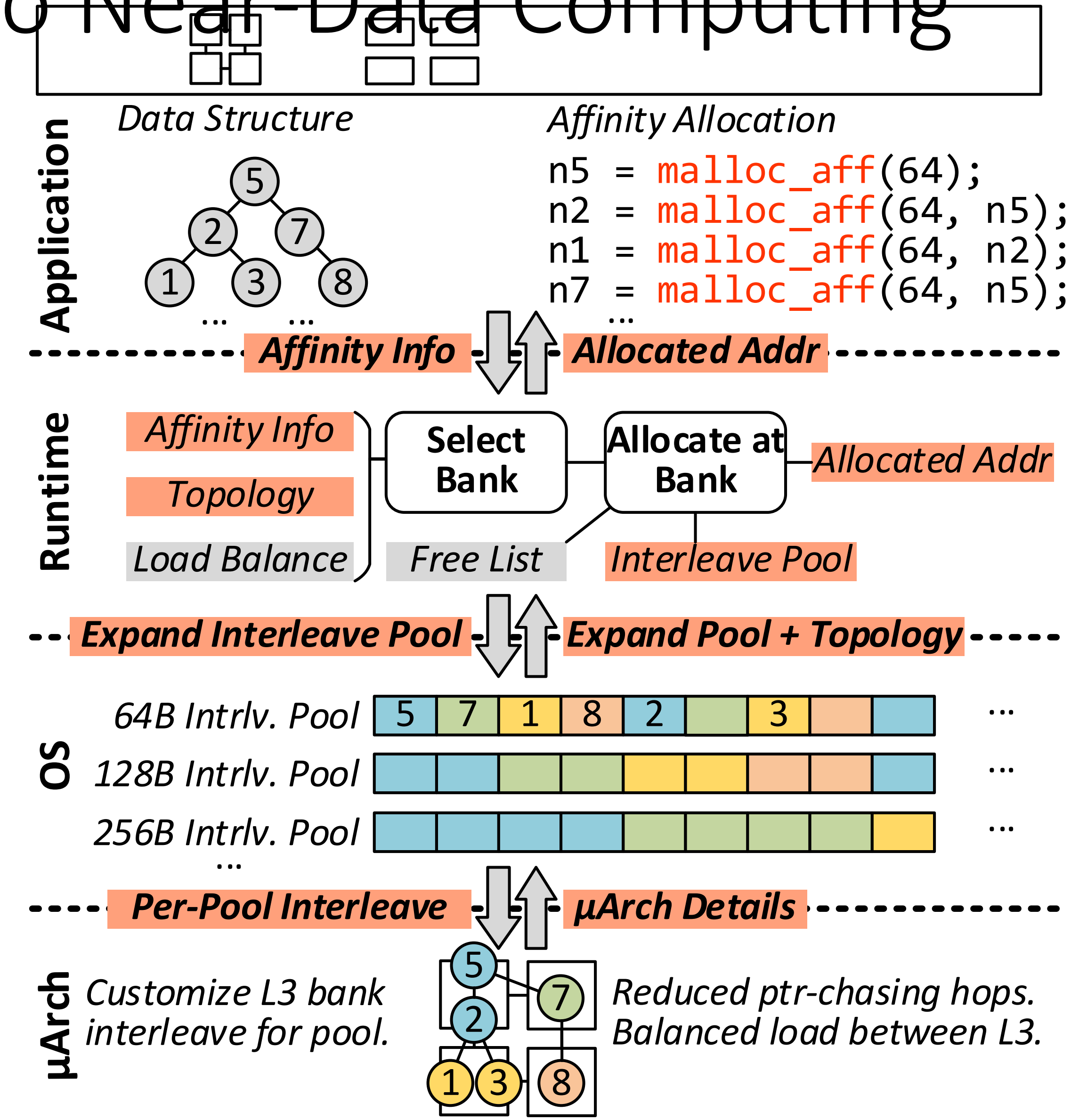


Affinity Alloc: Taming Not-So Near-Data Computing

- Clean: Each Layer exposes minimal interface.
- End-to-end: data affinity optimization.
- General: Regular & Irregular data structures.
- Unlock data structure co-optimization.



- 2.26 × speedup with 72% traffic reduction.

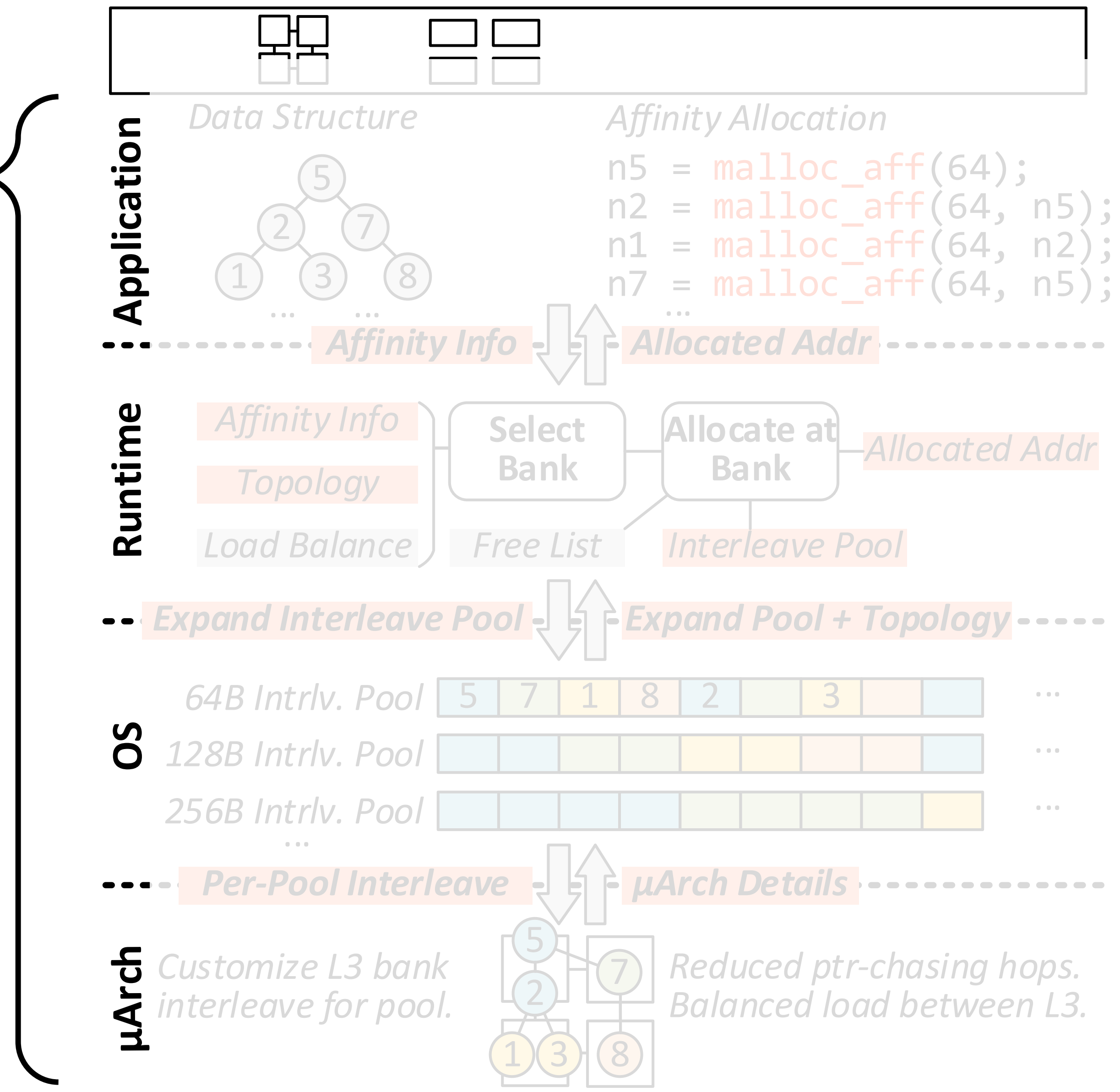


Roadmap

- Affine Data Layout
- Irregular Data Layout
- Data Structure Codesign
- Evaluation

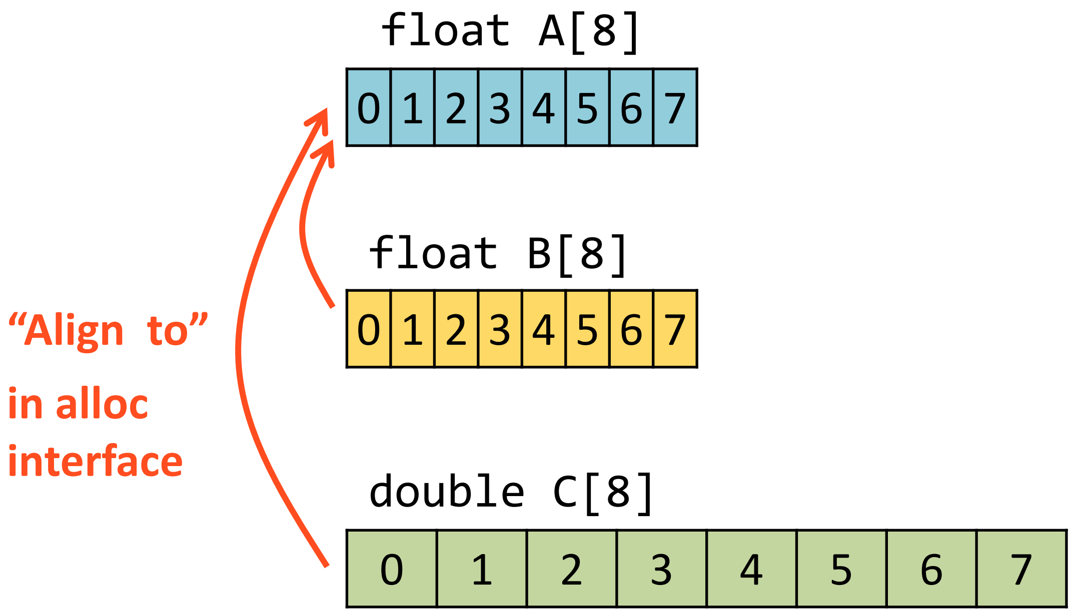
Roadmap

- Affine Data Layout
- Irregular Data Layout
- Data Structure Codesign
- Evaluation



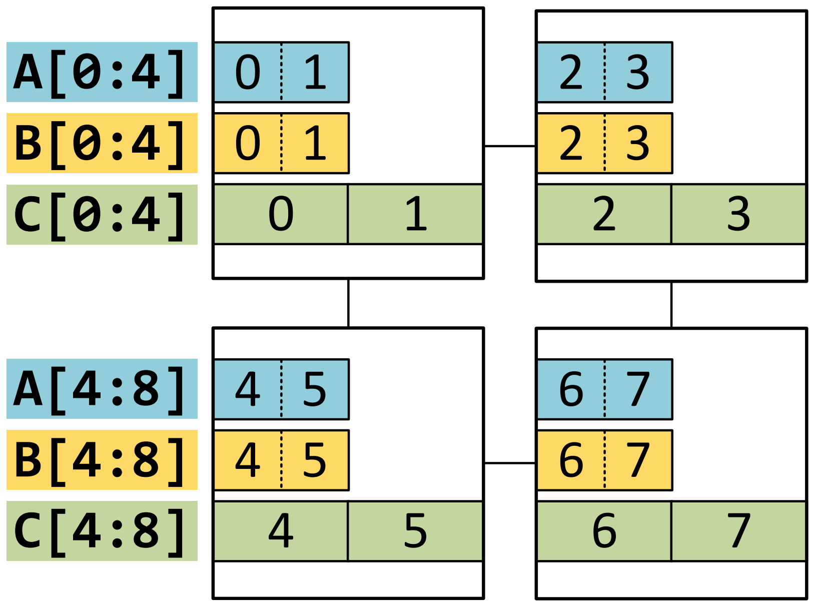
Inter-Array Affine Affinity

```
C[i] = A[i] + B[i];
double  float   float
```



Optimized Layout (8B $Line)

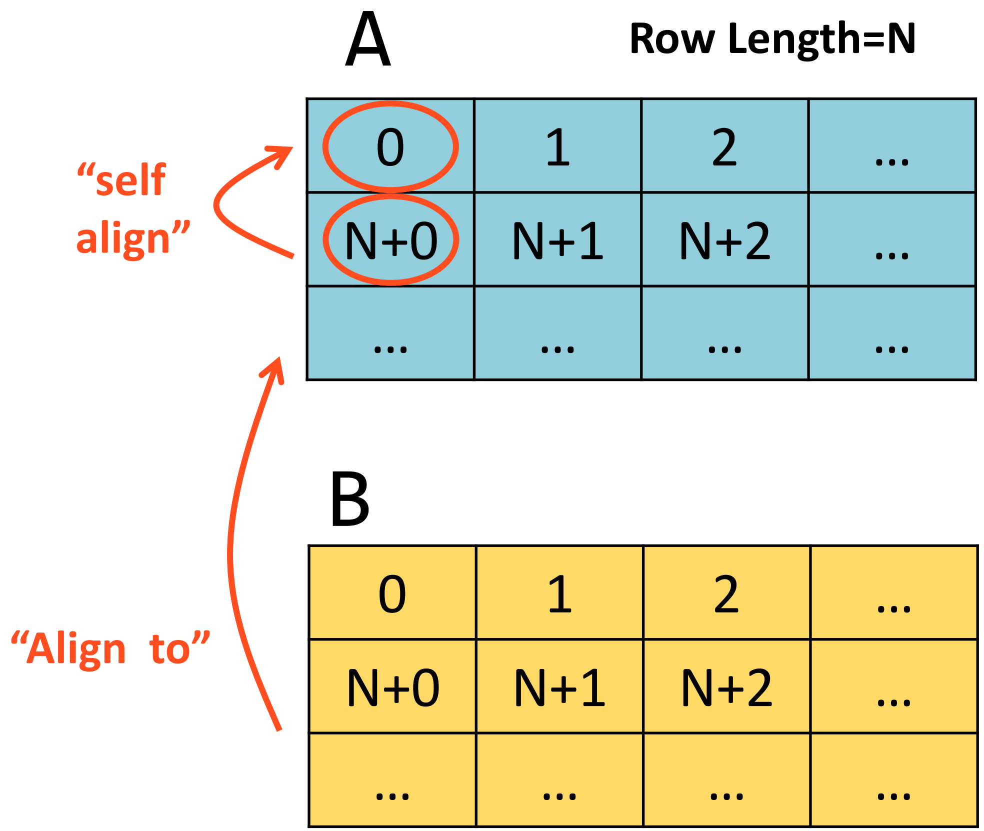
Interleave: A[] 8B, B[] 8B, C[] 16B

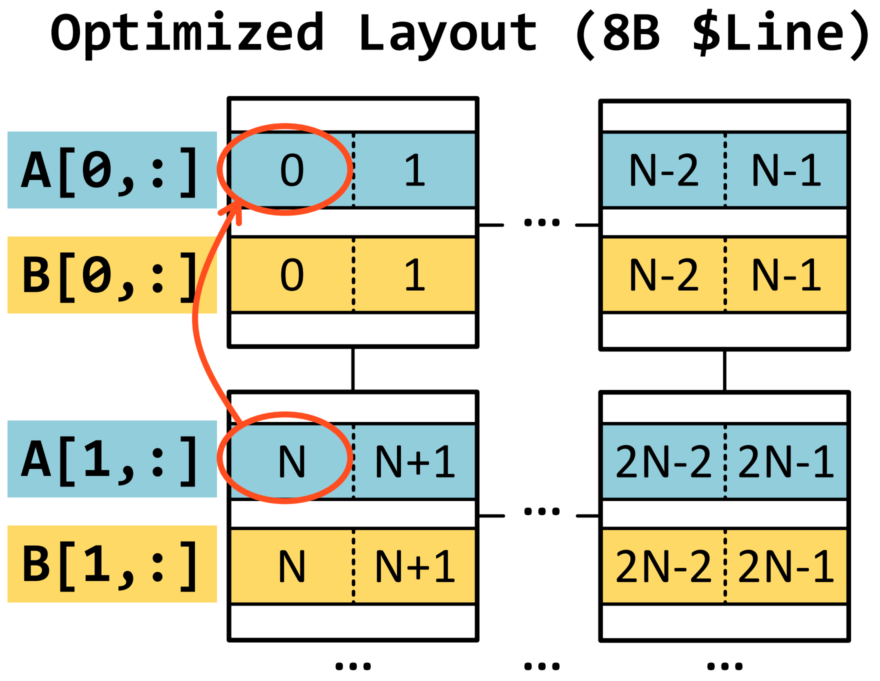


Same support required for strided access.

Intra-Array Affine Affinity

```
B[i,j] = A[i,j] + A[i+1,j]
```





Affinity Alloc Interface for Affine Data Layout

- Interface exposes affine layout transformation:

$$B[i] \rightarrow A\left[\frac{P}{Q} \times i + X\right]$$

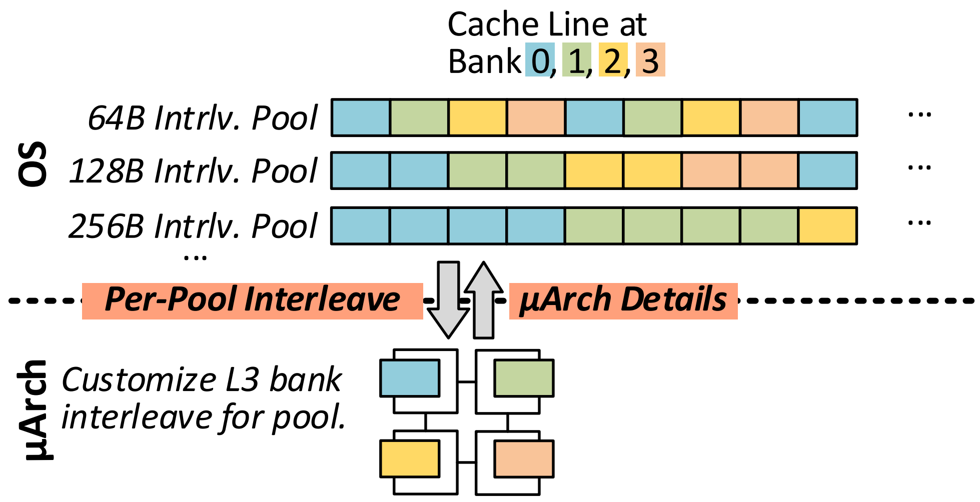
- Affine affinity alloc API:

```
struct AffineArray {
  int   elem_size; // Element size (byte).
  uint  num_elem;  // Number of elements.
  void* A;         // Pointer to the aligned affine array.
  int   P, Q;      // Interleaving Ratio
  int   X;         // Interleaving Offset
  …
};
void* malloc_aff(const AffineArray& a);
```

These parameters are independent of microarchitecture!

Mapping Virtual Addr. → LLC Banks

- Runtimes needs to be able to choose the actual cache-line interleaving and offset!
- OS Abstraction: Interleave pools
  - Set of "Direct Segment" (like Basu ISCA 2013)
  - Contiguous physical address within segment
  - Each pool is designated for power-of-2 interleaving

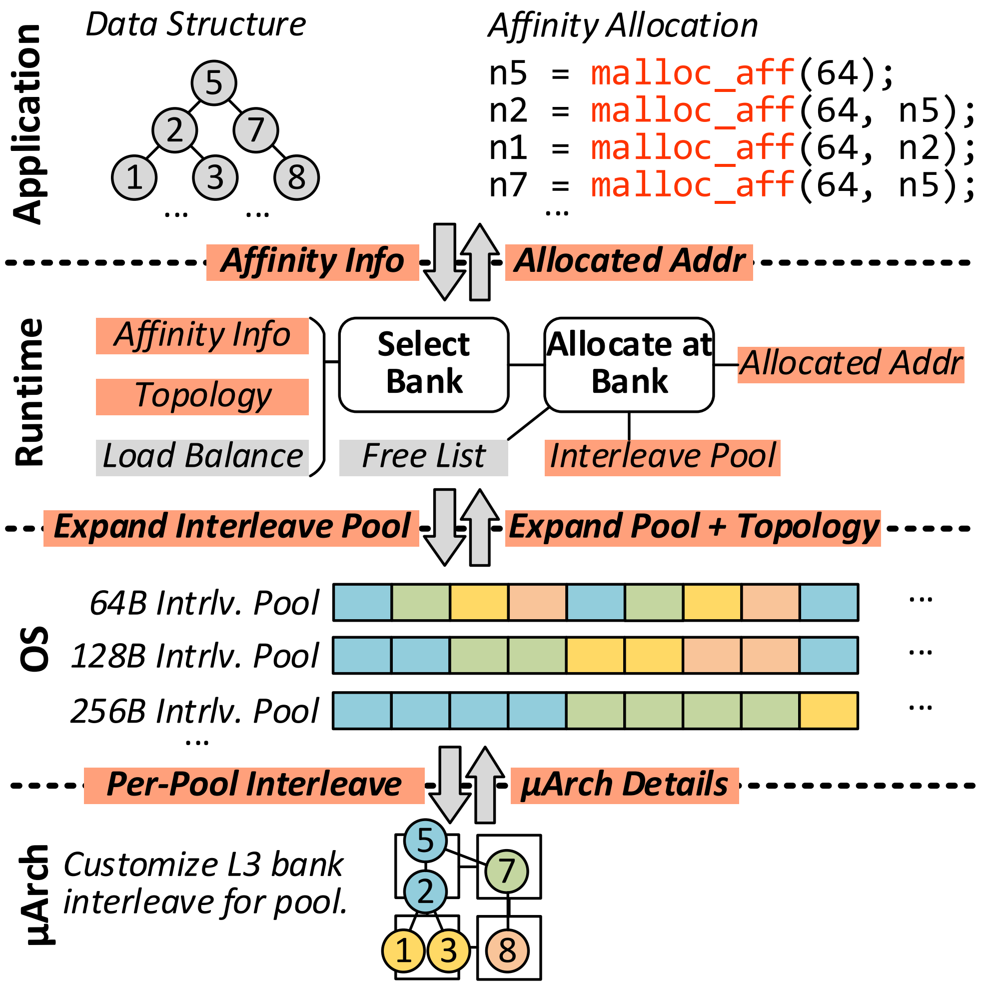


- μArch: Override cache->bank assignment

$$\mathrm{bank}(vaddr) = \left\lfloor \frac{vaddr - pool}{intrlv} \right\rfloor \pmod{N_{bank}}$$

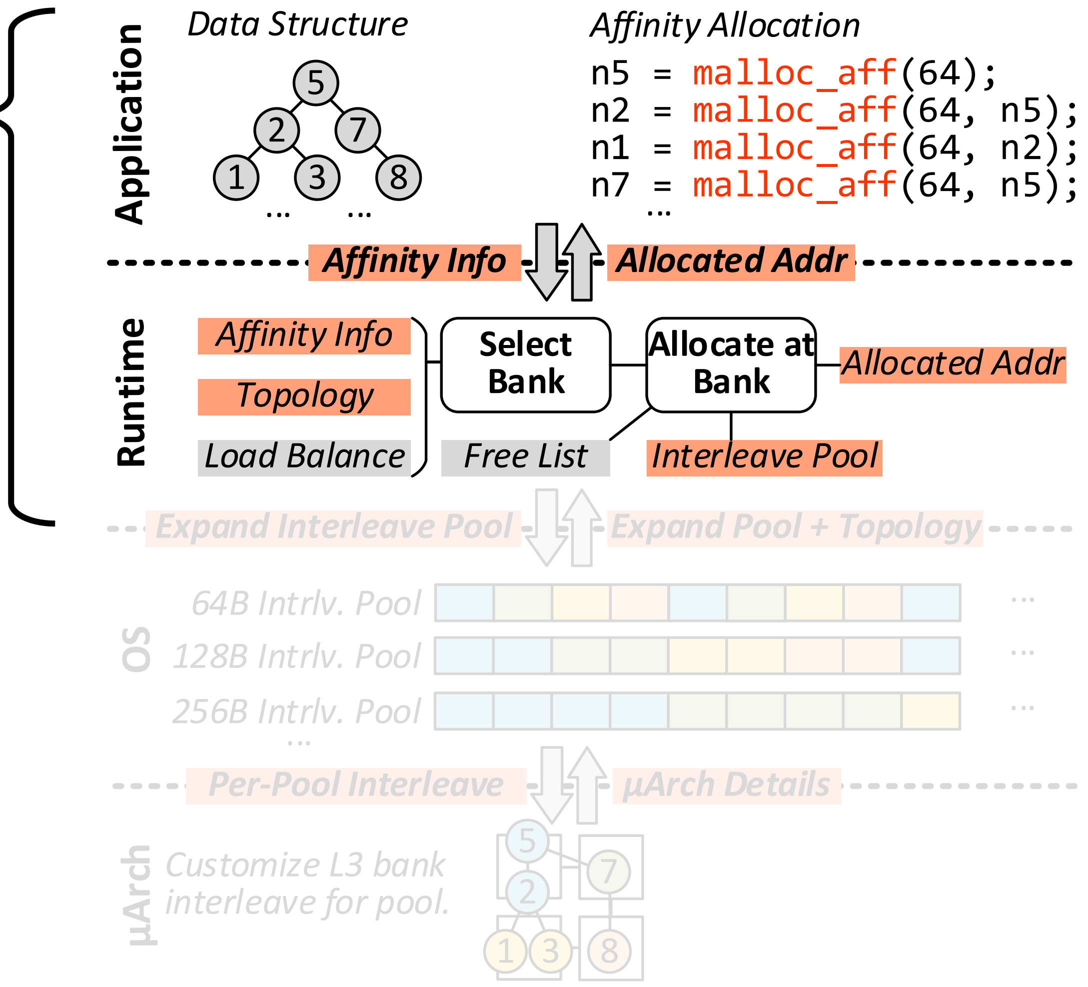
Mapping Virtual Addr. → LLC Banks

- OS: Manage interleave pools.
- μArch: Obeys interleaving of each pool.
- Application: Specify affinity relationships.
- Runtime: Choose and allocate to interleave pools.



Roadmap

- Affine Data Layout
- Irregular Data Layout
- Data Structure Codesign
- Evaluation



Irregular Data Layout

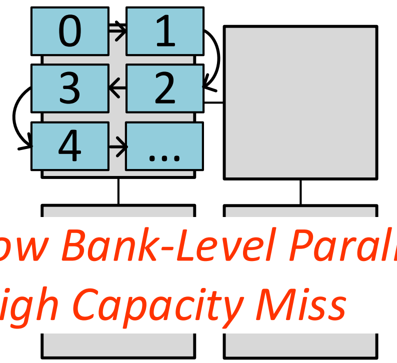
- Specify a list of irregular affinity addresses.
- Example: Linked list.
  - Random long pointer-chasing distance.
  - Affinity: Newly node → previous node.
  - Reduce the pointer-chasing distance.
- Load balancing:
  - Combine average hops and load at banks.

$$score = hops + H \times \left(\frac{load}{avg_{load}} - 1 \right)$$

  - Improve bank-level parallelism.
- ***No OS/microarchitecture overheads!***

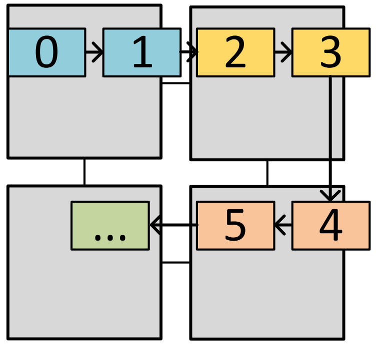
```
void* malloc_aff(uint size, // Alloc size.
    // Specify affinity addrs.
    int num_aff_addrs, void** aff_addrs);

void linked_list_append(Node *prev, T v)
    // Allocate new node near to prev.
    Node *n = malloc_aff(sizeof(Node), 1, &prev);
    n->v = v; n->nxt = prev->nxt; prev->nxt = n;
```

Unbalanced Layout



Low Bank-Level Parallelism
High Capacity Miss

Optimized Layout

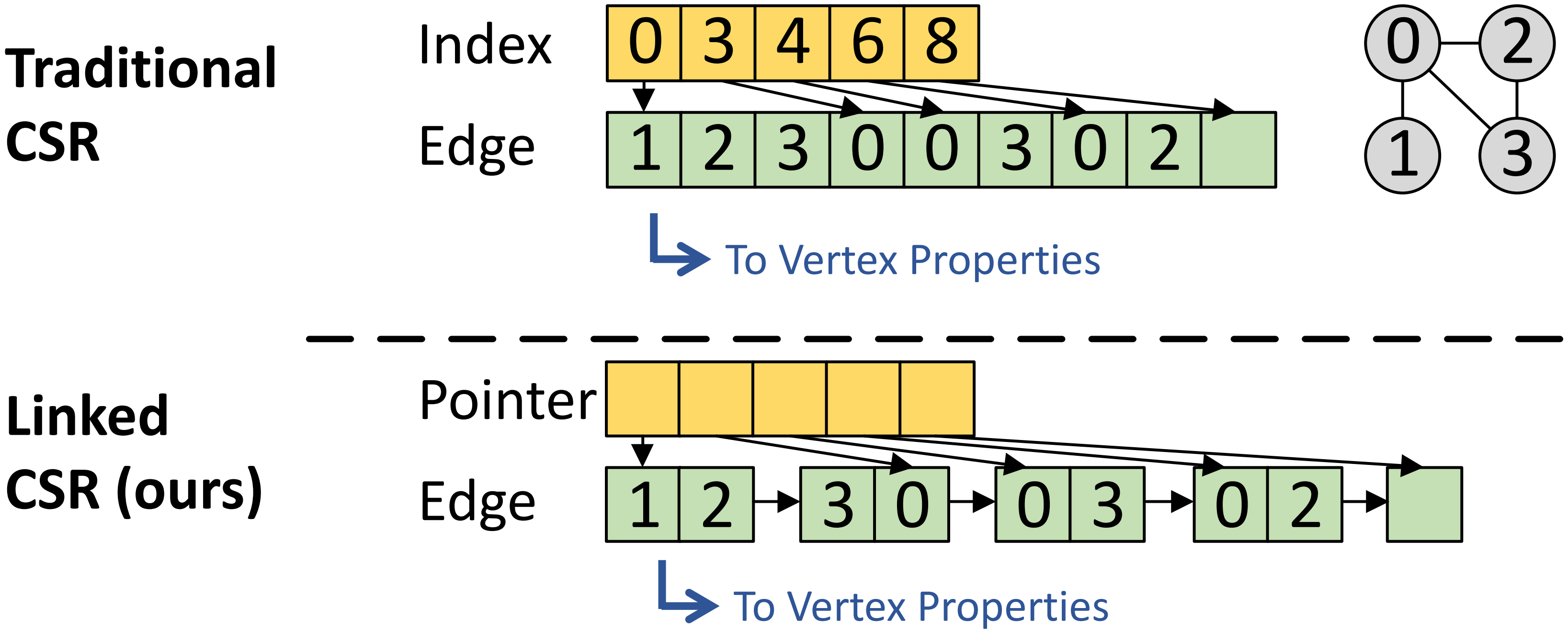


Roadmap

- Affine Data Layout
- Irregular Data Layout
- Data Structure Codesign
- Evaluation

Data Structure Codesign: Linked-CSR Format

- Original CSR uses array to store edges – inflexible for data placement.



- Linked CSR replaces the edge array with linked list.
- Each linked list node can be placed closer to outgoing vertices.

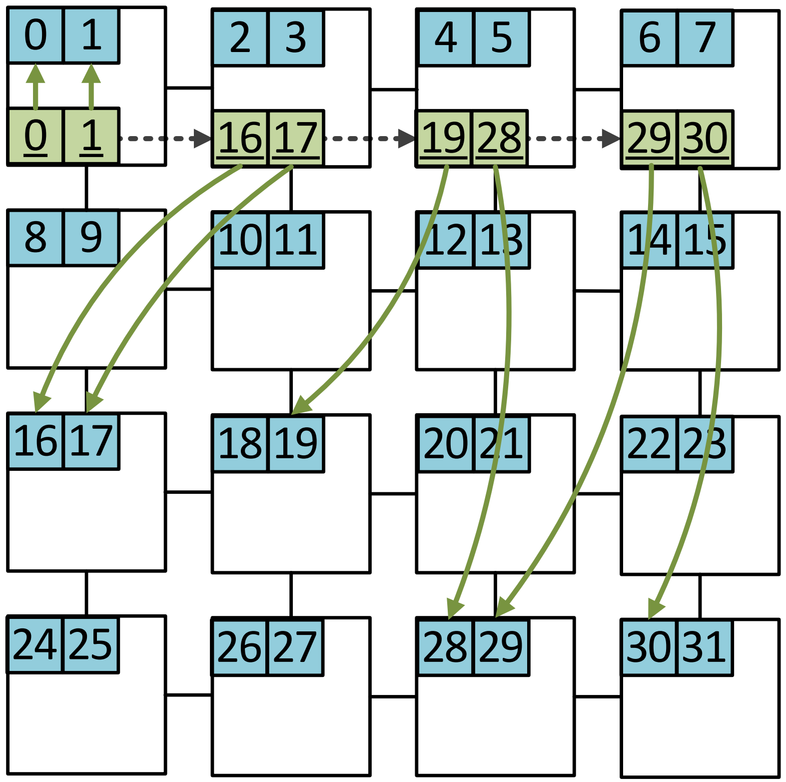
Data Layout Example: CSR Graph Traversal

4 5 — Vertex Cache Lines

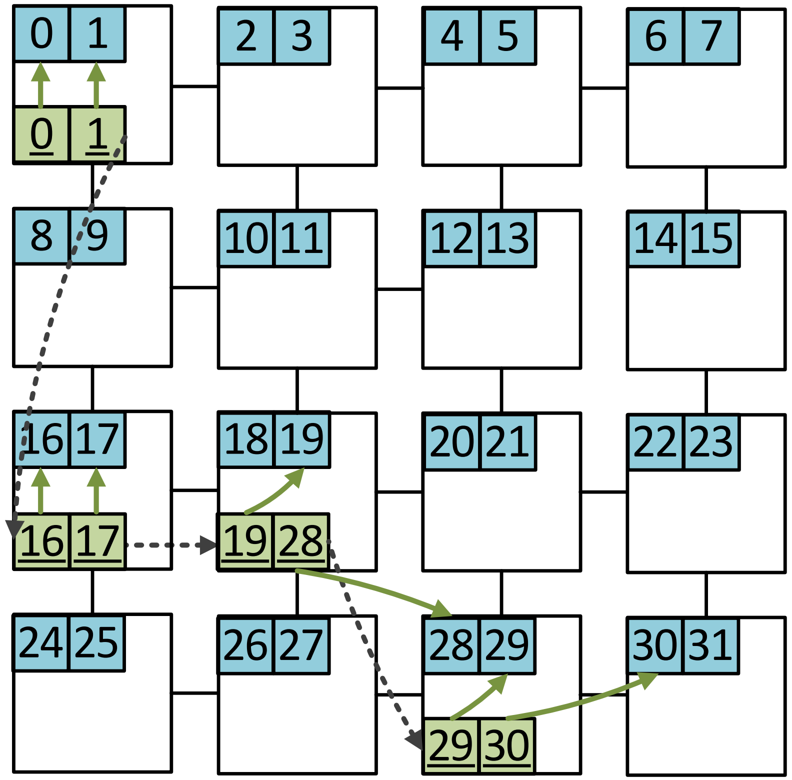
1 19 — Edge Cache Lines, V: Out Vertex

→ Indirect Request from Edge to Vertex

Indirect: 19 Hops
Migration: 3 Hops



Naïve NDC (Intrlv=1$Line)



Idea Affinity-Aware NDC

Indirect: 3 Hops
Migration: 5 Hops

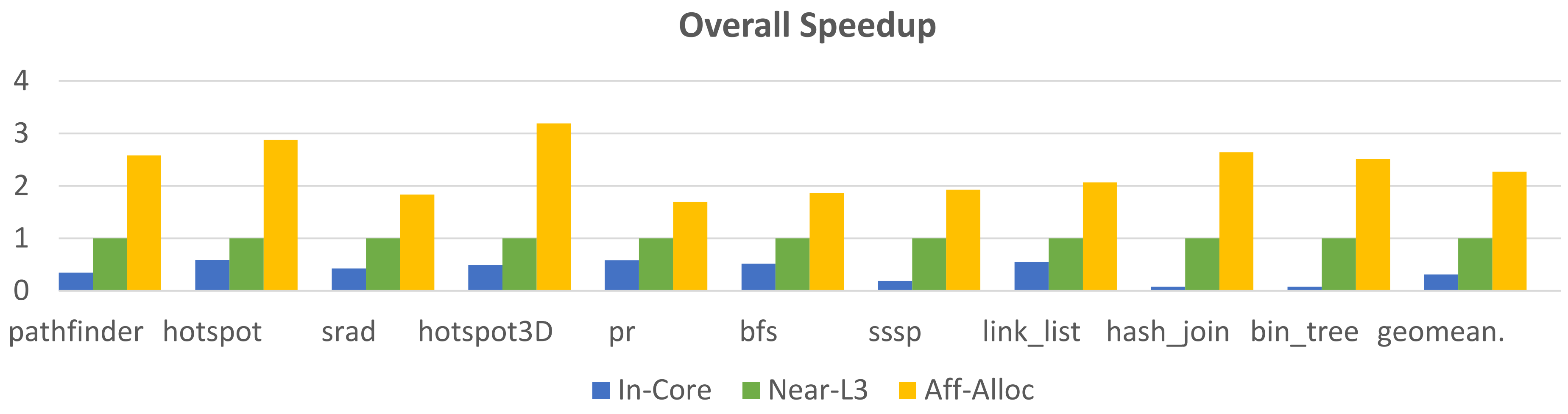
Roadmap

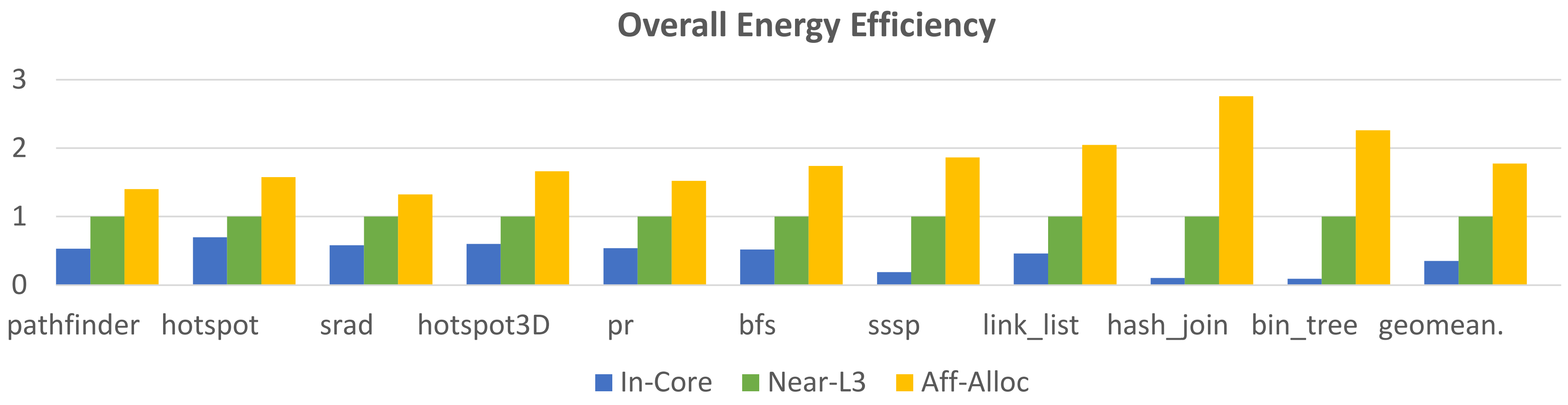
- Affine Data Layout
- Irregular Data Layout
- Data Structure Codesign
- Evaluation

Methodology

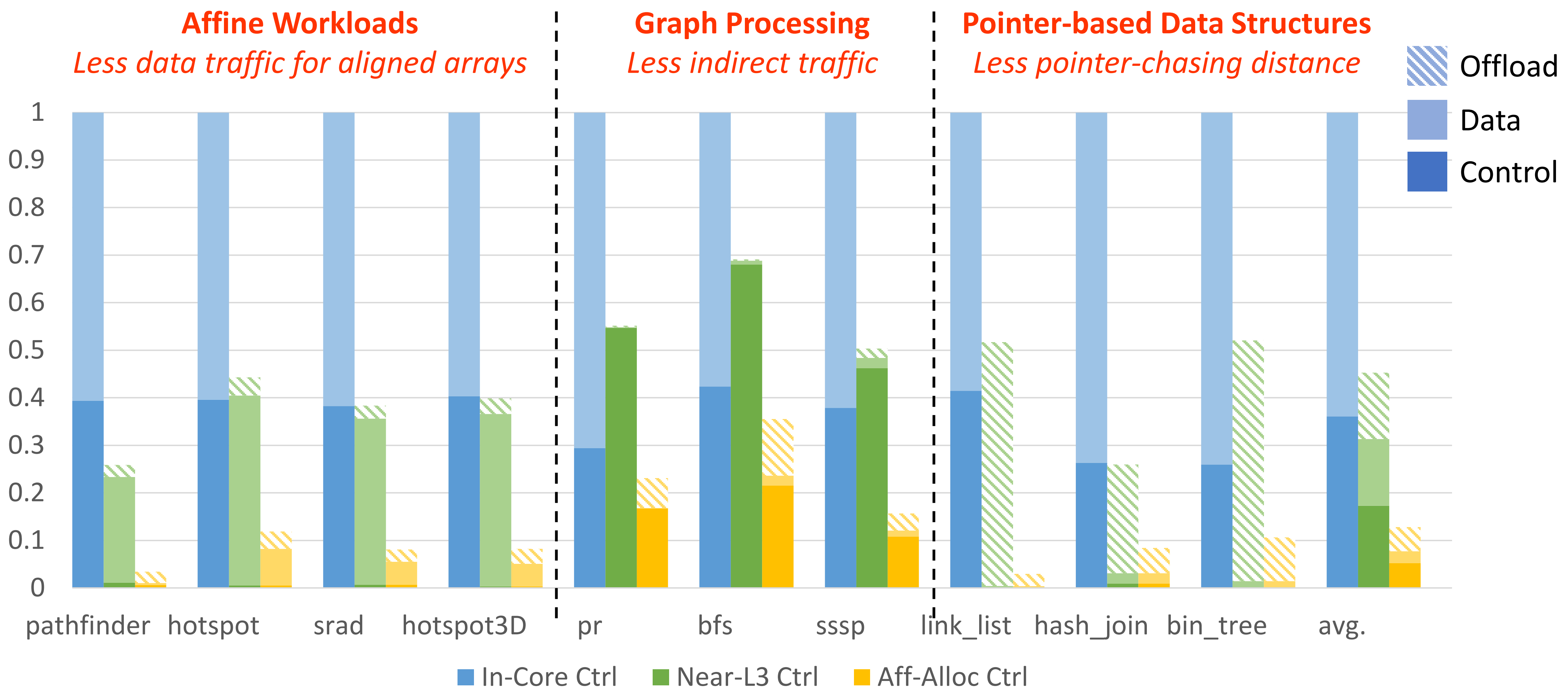
- LLVM-based Compiler
- Gem5 20.0 cycle-level execution-driven simulator.
- 10 data processing workloads from Rodinia, Gap Graph Suite and micro kernels.
  - Parallelized with OpenMP, with AVX-512 enabled.
- Configurations (see paper for details):
  - 64 Cores, 8x8 mesh topology, 3-level MESI
  - Cache Hierarchy: 32kB L1 I/D, 256kB L2, 1MB L3.
- Comparison Points
  - **In-Core:** No near-data (Bingo spatial prefetcher [HPCA2019] at L1 + stride prefetcher at L2.)
  - **Near-L3:** Near-stream Computing [HPCA ’22].
  - **Aff-Alloc:** This Work

Overall Performance and Energy Efficiency





Network Traffic



Affinity Alloc: ~~Not-So~~ *Truly* Near-Data Computing

- Minimal interface between system layers hides microarchitecture from programmer and OS.
- Express both coarse-grained and fine-grained affinity relationship in allocator.
- Automatic data layout optimization for NDC.
- Data structure co-optimization with the controlled affinity.
- 2.26× speedup and 72% traffic reduction.