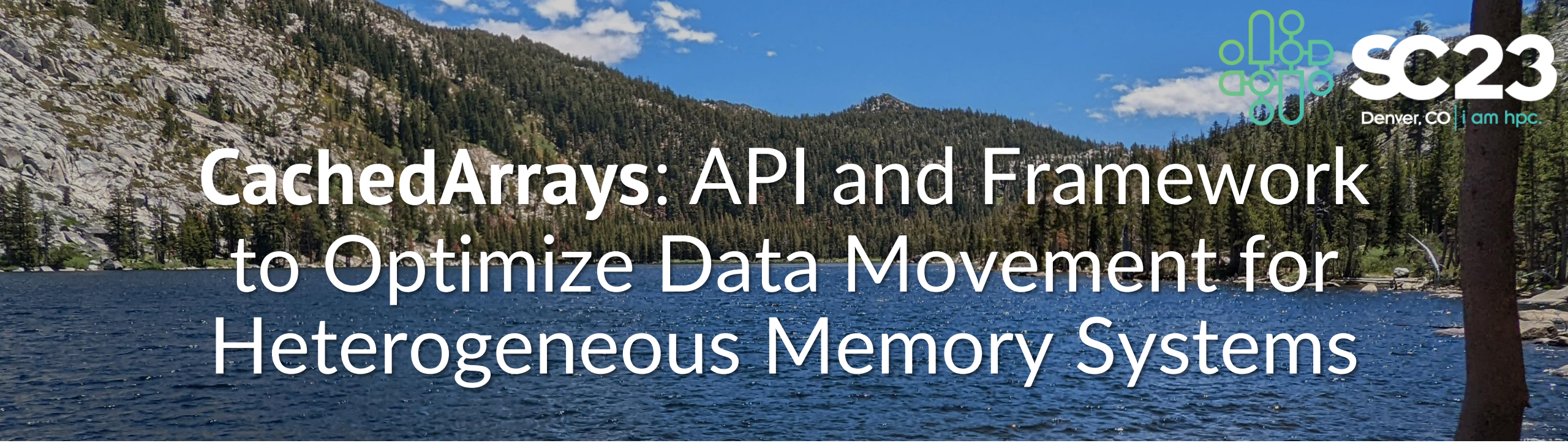


CachedArrays: API and Framework to Optimize Data Movement for Heterogeneous Memory Systems

Mark Hildebrand, **Jason Lowe-Power**, Venkatesh Akella

jlowepower@ucdavis.edu

DArchR
DAVIS ARCHITECTURE RESEARCH

UCDAVIS
COMPUTER SCIENCE

Summary

Working memory sizes growing

- Heterogeneous memory devices needed to keep up

Current data movement strategies are limited

- High overheads, inflexible, missing important optimizations

CachedArrays: Separate the mechanisms and policies

- Simple interface for programmers
- Flexible backend for framework developers

Prototype in software showing efficiency for CNNs and DLRM

UCDAVIS COMPUTER SCIENCE

Memory requirements are growing

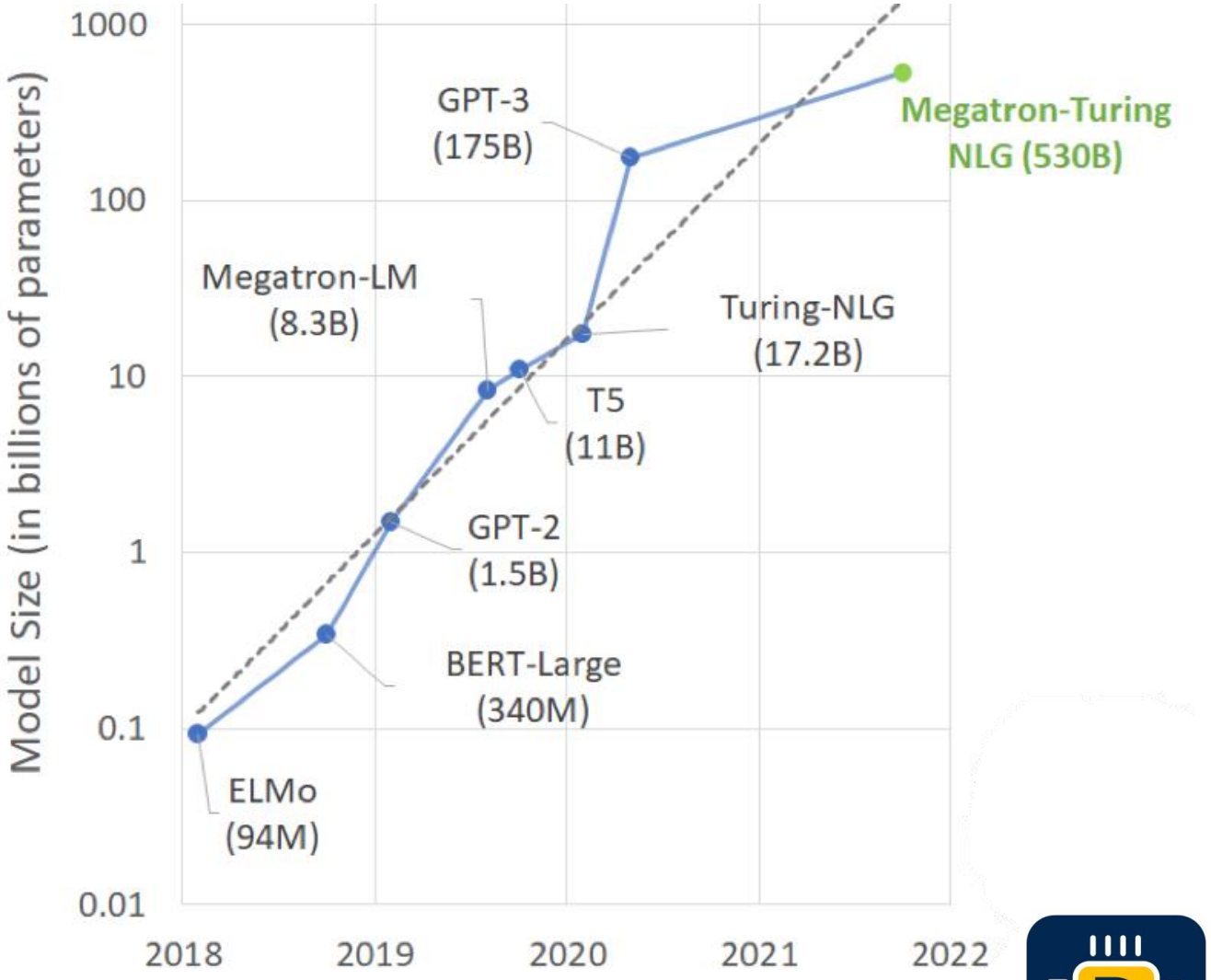
Not just deep learning

Graph analytics:

- billions of vertices
- trillions of edges

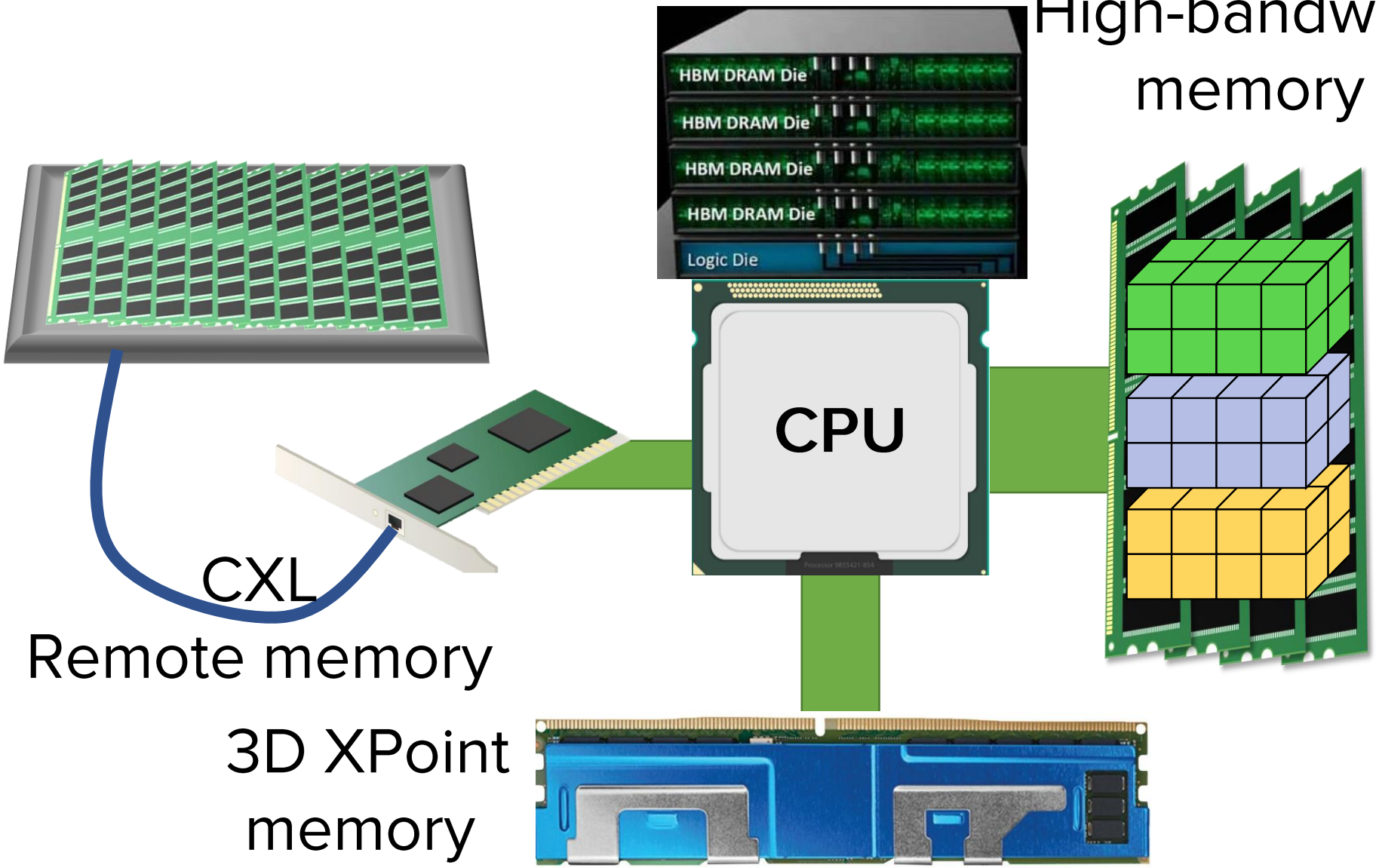
Working memory

- Byte addressable
- Low latency
- High bandwidth



UCDAVIS
COMPUTER SCIENCE

System architecture



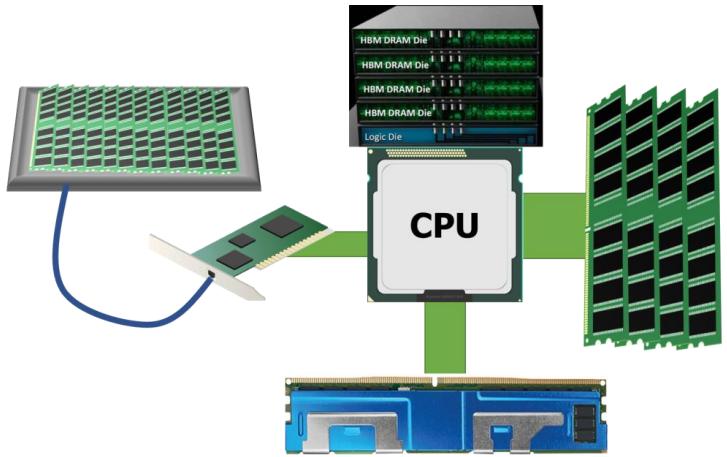
High-bandwidth memory

Traditional DRAM

Remote memory

3D XPoint memory

Why conventional techniques don’t work



CachedArrays
A solution to bridge the semantic gap

UCDAVIS COMPUTER SCIENCE

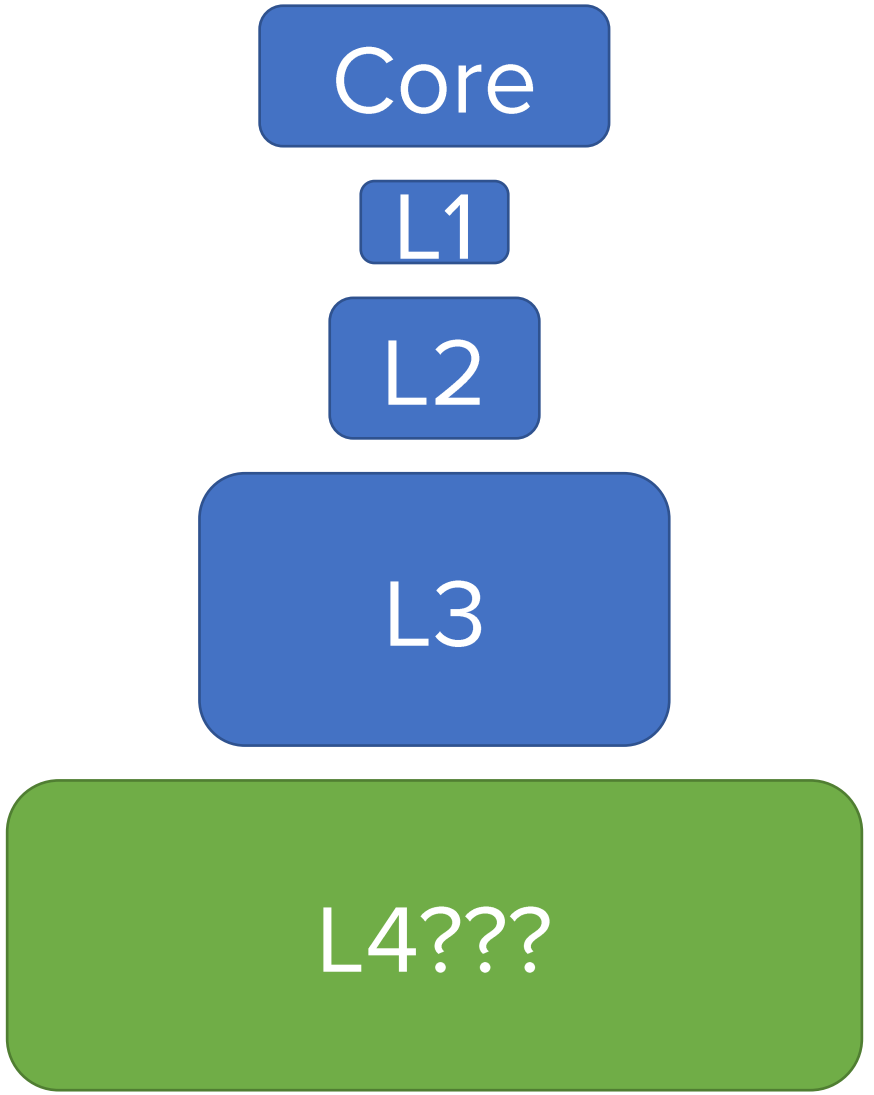
Conventional data movement

Why not just treat faster memory as a cache?

Caches have been great!

- Block-level
- Programmer transparent
- Traditionally, low overhead

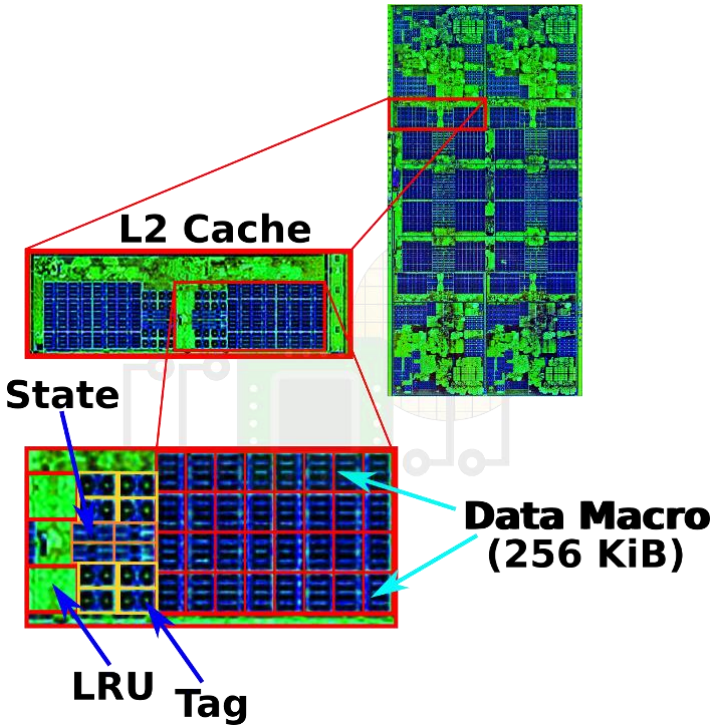
DRAM has some issues, though…



Difference between SRAM & DRAM

In SRAM caches:

Tag, LRU, etc. in different structure. High bandwidth



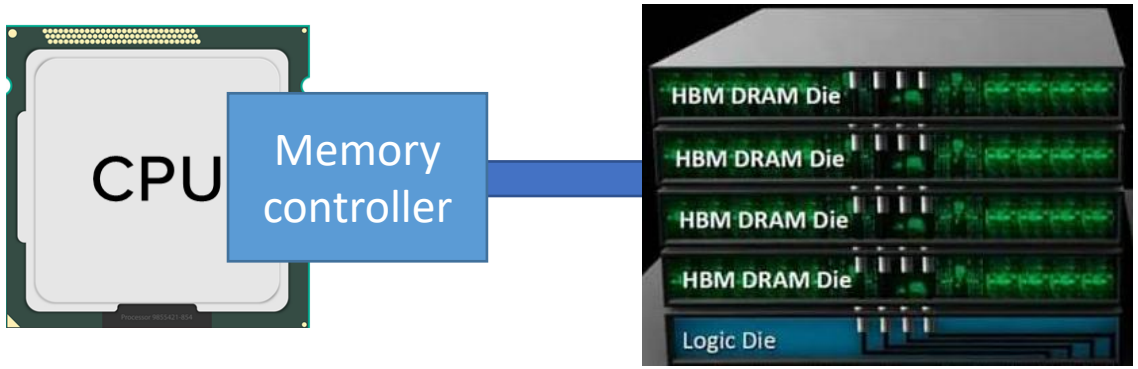
In DRAM caches:

Off-chip

Tag, LRU, etc. share bus with data

Lower bandwidth

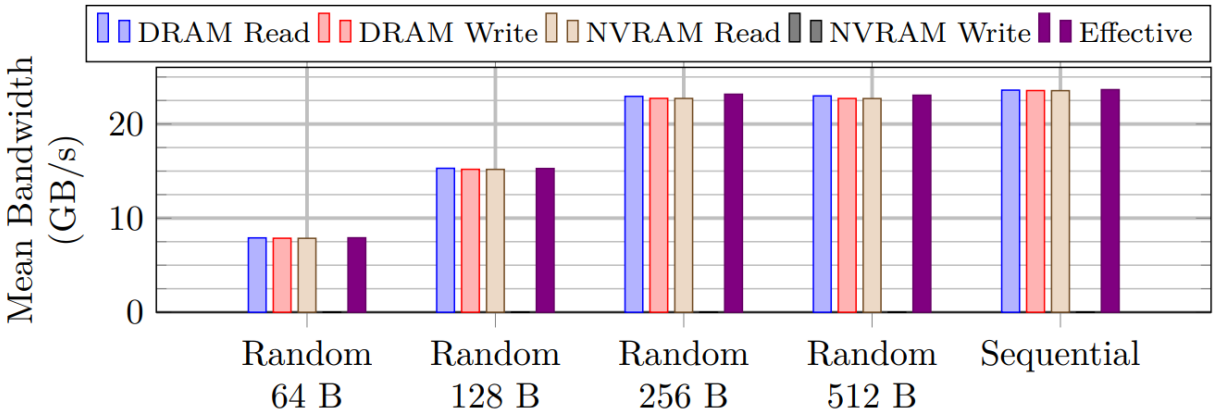
Higher latency



https://fuse.wikichip.org/news/1177/amds-zen-cpu-complex-cache-and-smu/2/

UCDAVIS COMPUTER SCIENCE

Results: Microbenchmarks on hardware



(a) *Read-only* benchmark, clean LLC read misses, 24 threads.

Each read requires **3** accesses

Each write requires **5** accesses

UCDAVIS COMPUTER SCIENCE

Option 2: Operating system paging

Many, many examples of “NUMA-style” data movement

Three problems

- Data movement granularity is 4 KiB or 2 MiB pages
- Timeliness of movement: Policy doesn’t have insight into dynamic access
- Policy has no information on semantics of data use

OK for some workloads (e.g., cloud/VM)

Inefficient for many others

Requirements for efficient data movement

Transparent to the programmer (Mostly)

- Hints are OK
- Library (e.g., PyTorch) changes OK

Semantic information of data use to drive movement

- Sage, vDNN, AutoTM, ZeRO-Offload, etc.

Important optimizations

1. Initial access data placement in fast memory
2. Elide dead data writebacks
3. Move data at *right* granularity
4. Avoid polluting fast memory

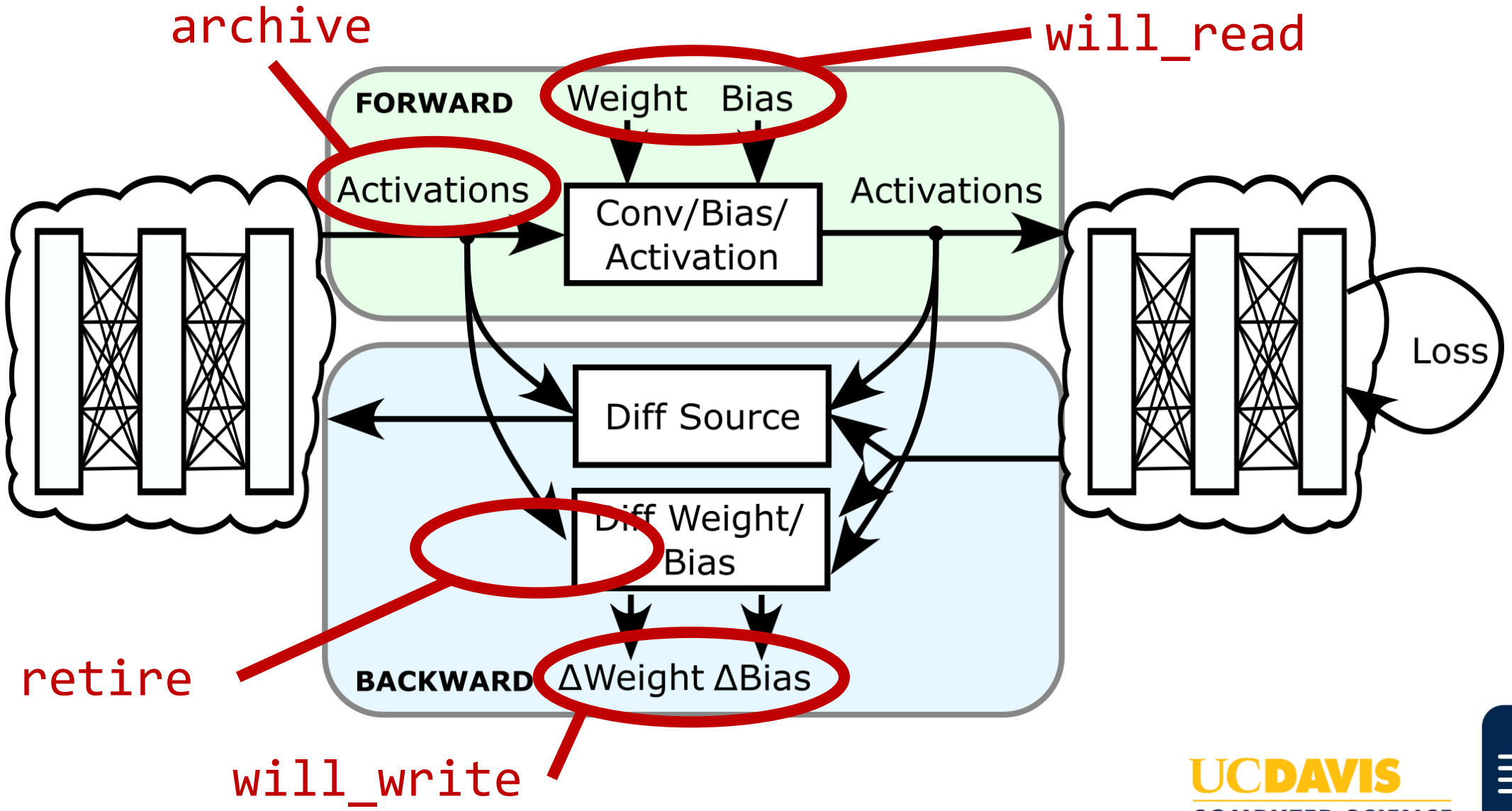
CachedArrays API for data movement

Exposes the following *hints* to the programmer/library developer

Use ***object*** granularity. Not page, block, etc.

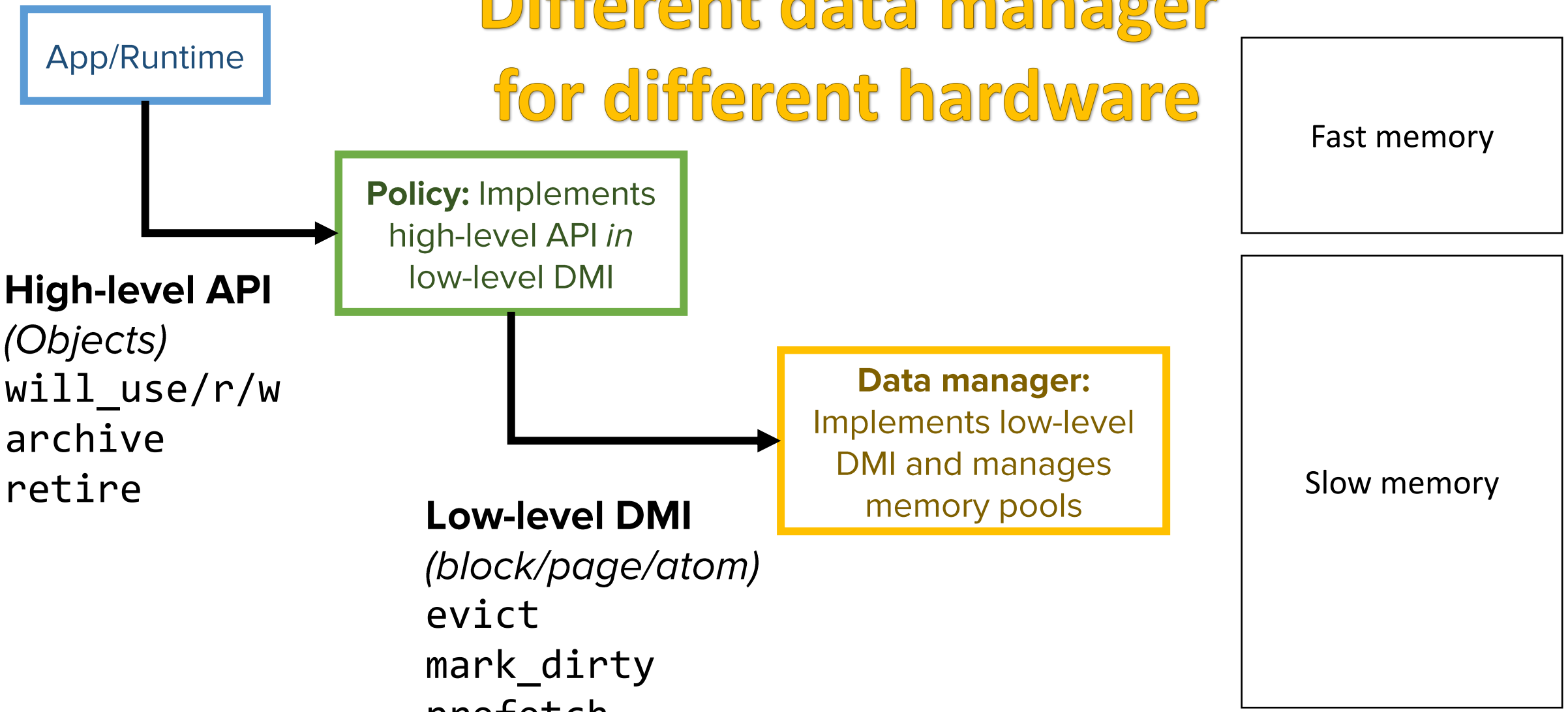
`will_use`	I am going to read and/or write this object
`will_read`	I am going to access object read-only
`will_write`	I am going to write and/or read this object
`archive`	I am not going to use this object for a while
`retire`	I am never going to access this object again

Using the CachedArrays API for CNNs



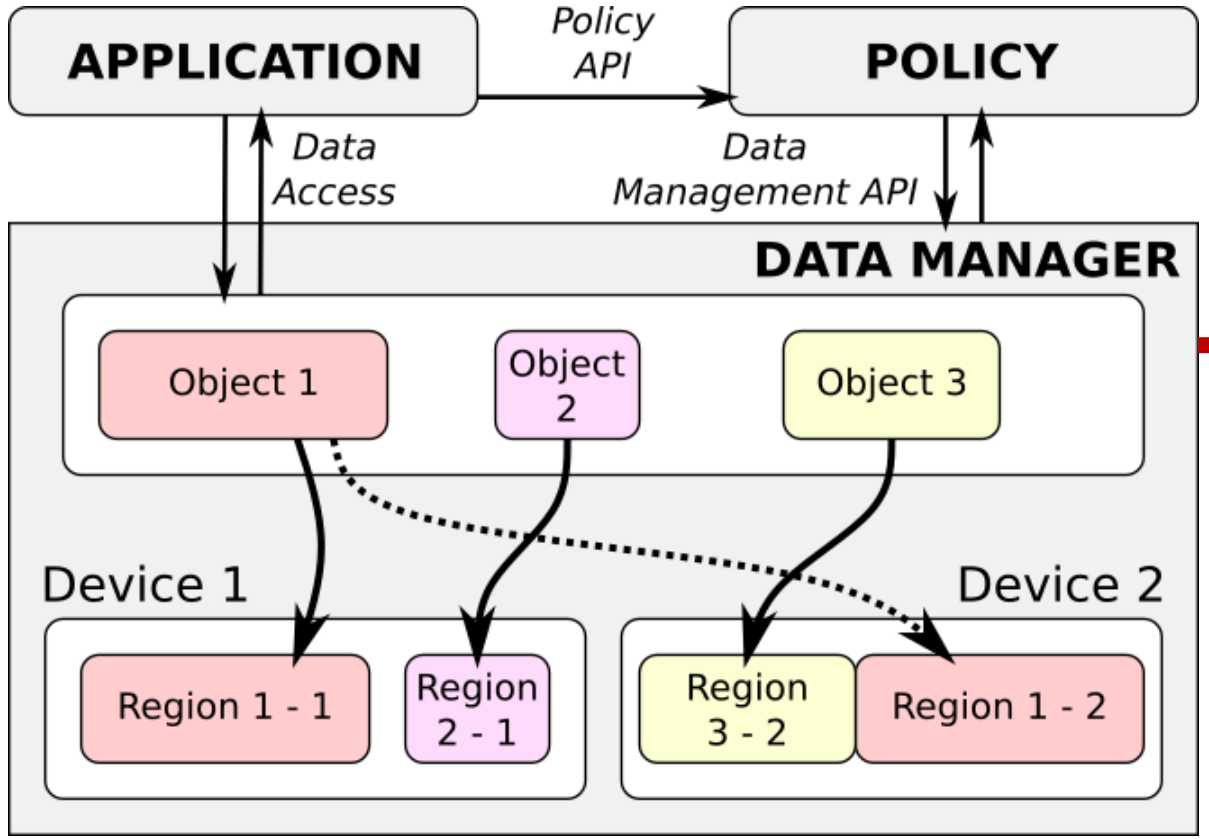
UCDAVIS
COMPUTER SCIENCE

Different data manager for different hardware



UCDAVIS COMPUTER SCIENCE

Example of low-level DMI




```
function prefetch(
  policy, object, force::Bool = false)
  x = DM.getprimary(object)
  if DM.in(x, SLOW)
    sz = DM.sizeof(object)
    y = DM.allocate(FAST, sz)
    if isnothing(y) && force
      start = find_region(policy)
      DM.evictfrom(FAST, start, sz) do region
        evict(policy, DM.parent(region))
      end
      y = DM.allocate(FAST, sz)::Region
    else if isnothing(y) && !force
      return
    end
    DM.copyto(y, x)
    DM.link(x, y)
    DM.setprimary(object, y)
  end
  return
end
```

UCDAVIS COMPUTER SCIENCE

Evaluation platform

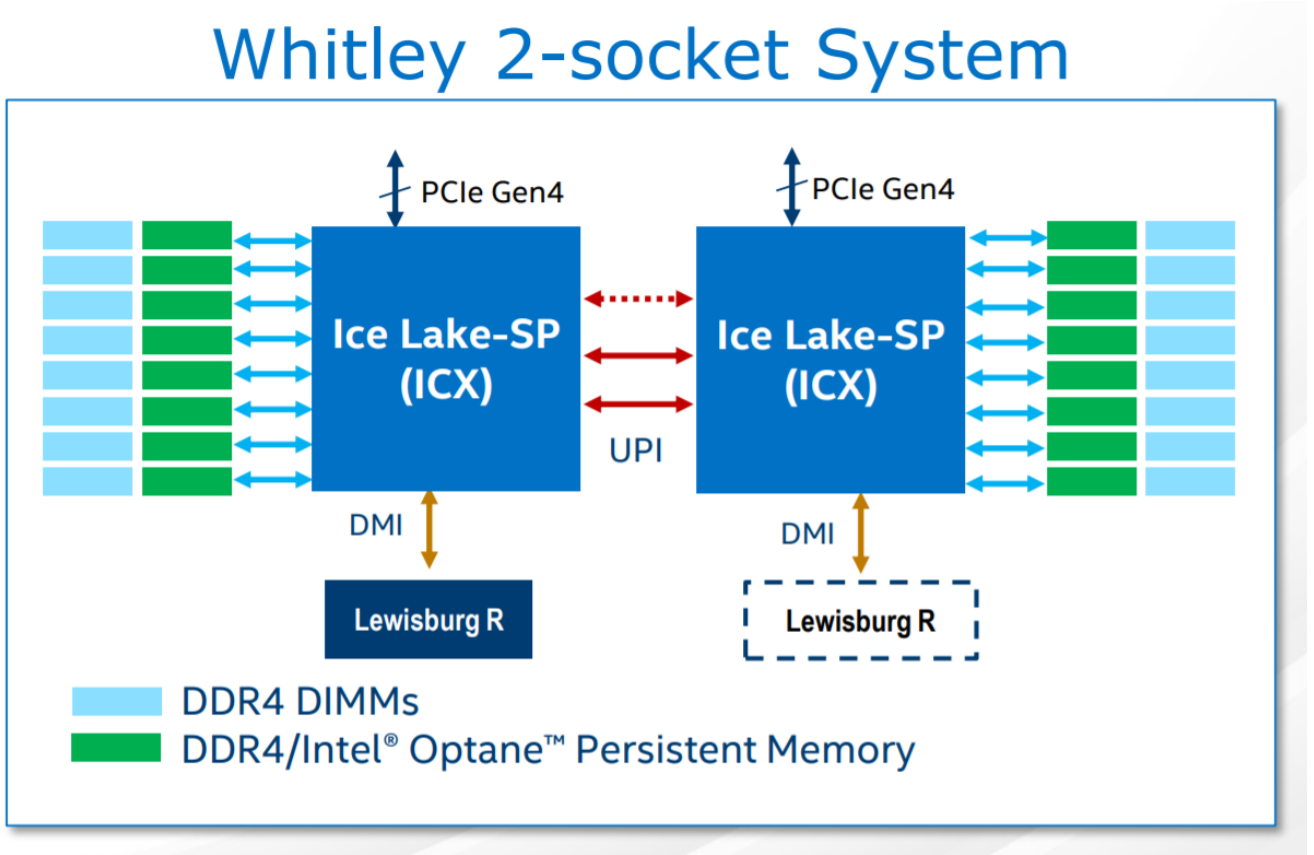
Optane and DRAM share bus

Optane DIMMs: >512GB

DRAM DIMMs: >64-128GB

Total memory per node:
3-4TB Optane
384-512GB DRAM
(SRAM <64MB)

Compare to DRAM Cache (2LM)



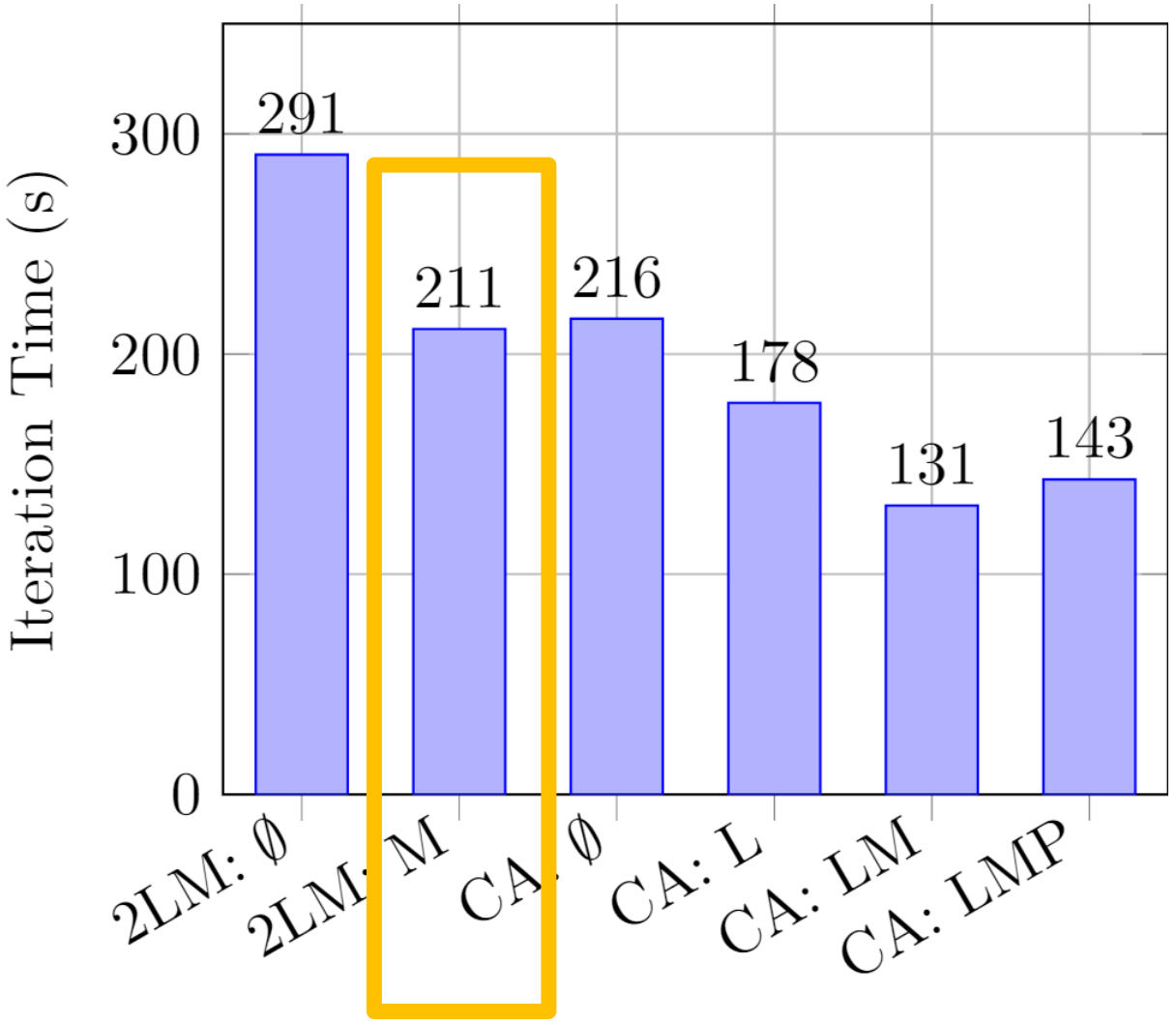
Large Networks			Small Networks	
Model	Batchsize	Footprint	Model	Batchsize
DenseNet 264	1536	526 GB	DenseNet 264	504
ResNet 200	2048	529 GB	ResNet 200	640
VGG 416	256	520 GB	VGG 116	320

UCDAVIS COMPUTER SCIENCE

Results for DNNs

Three optimizations:

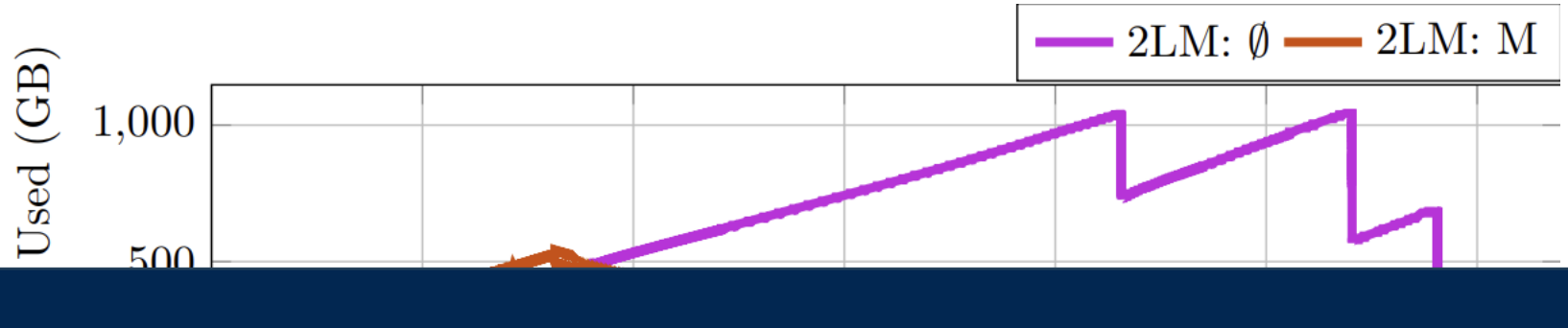
- Memory freeing (M)
  `retire`
- Allow local accesses (L)
- Prefetching (P)



DenseNet 264

UCDAVIS COMPUTER SCIENCE

Benefits of **`retire`** (memory freeing)



Take away: Adding semantic information about memory use improves efficiency

Reu

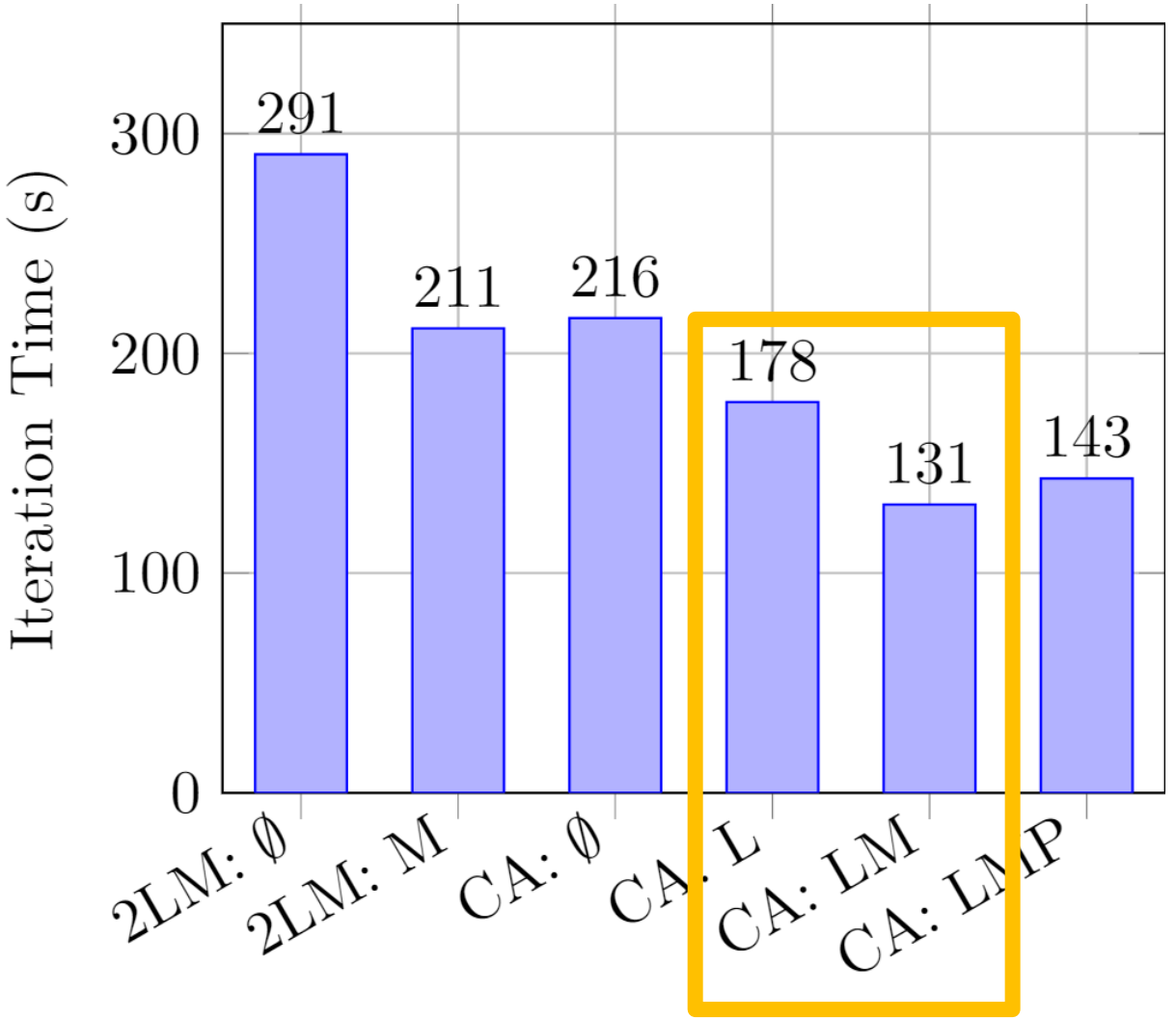
Memory reclaimed at lower cost

UCDAVIS COMPUTER SCIENCE

Results for DNNs

Three optimizations:

- Memory freeing (M)
  - **`retire`**
- Allow local accesses (L)
- Prefetching (P)



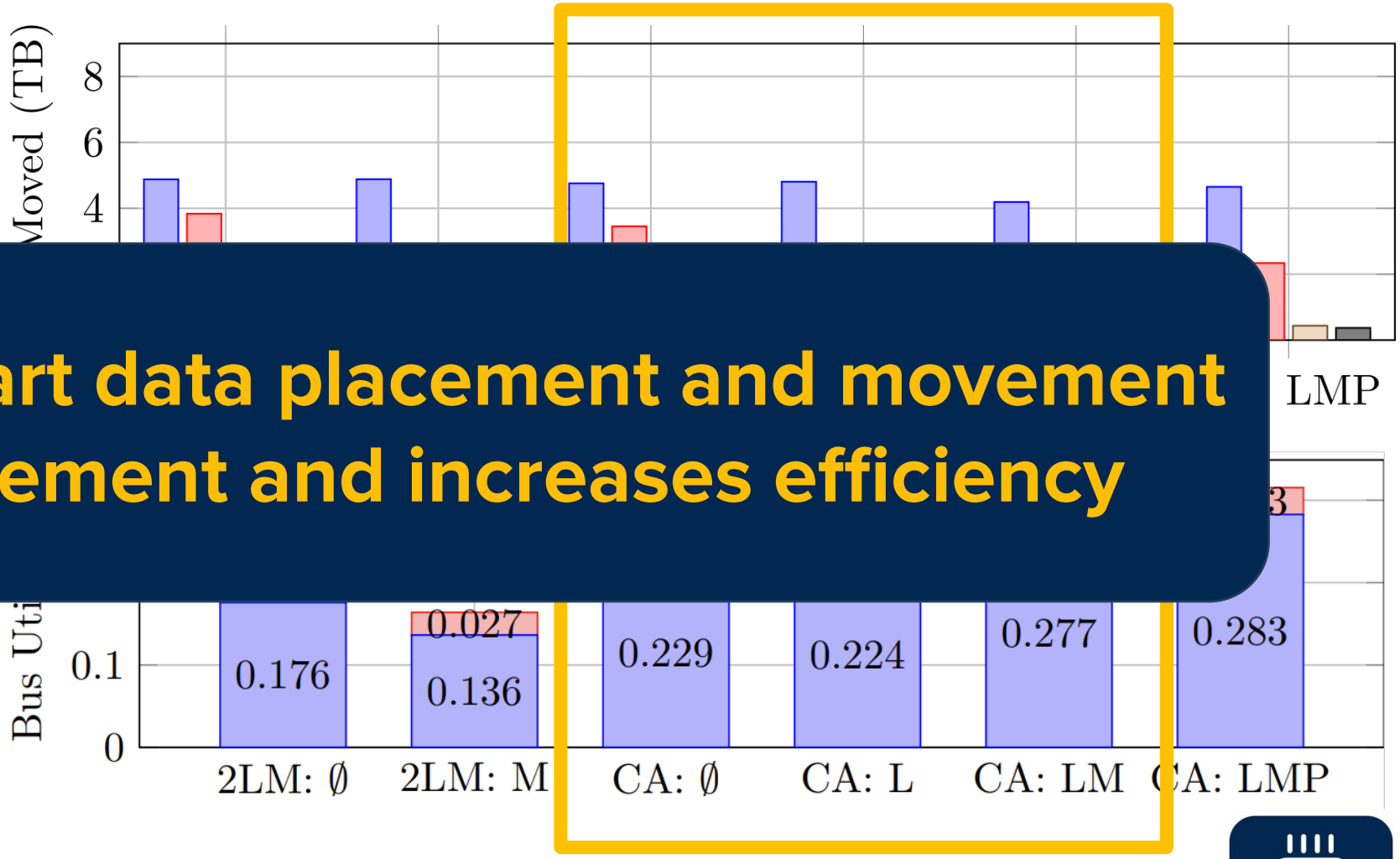
DenseNet 264

UCDAVIS COMPUTER SCIENCE

Data placement and object-based movement

Placing data in fast/local memory reduces movement

Us
movement improves efficiency



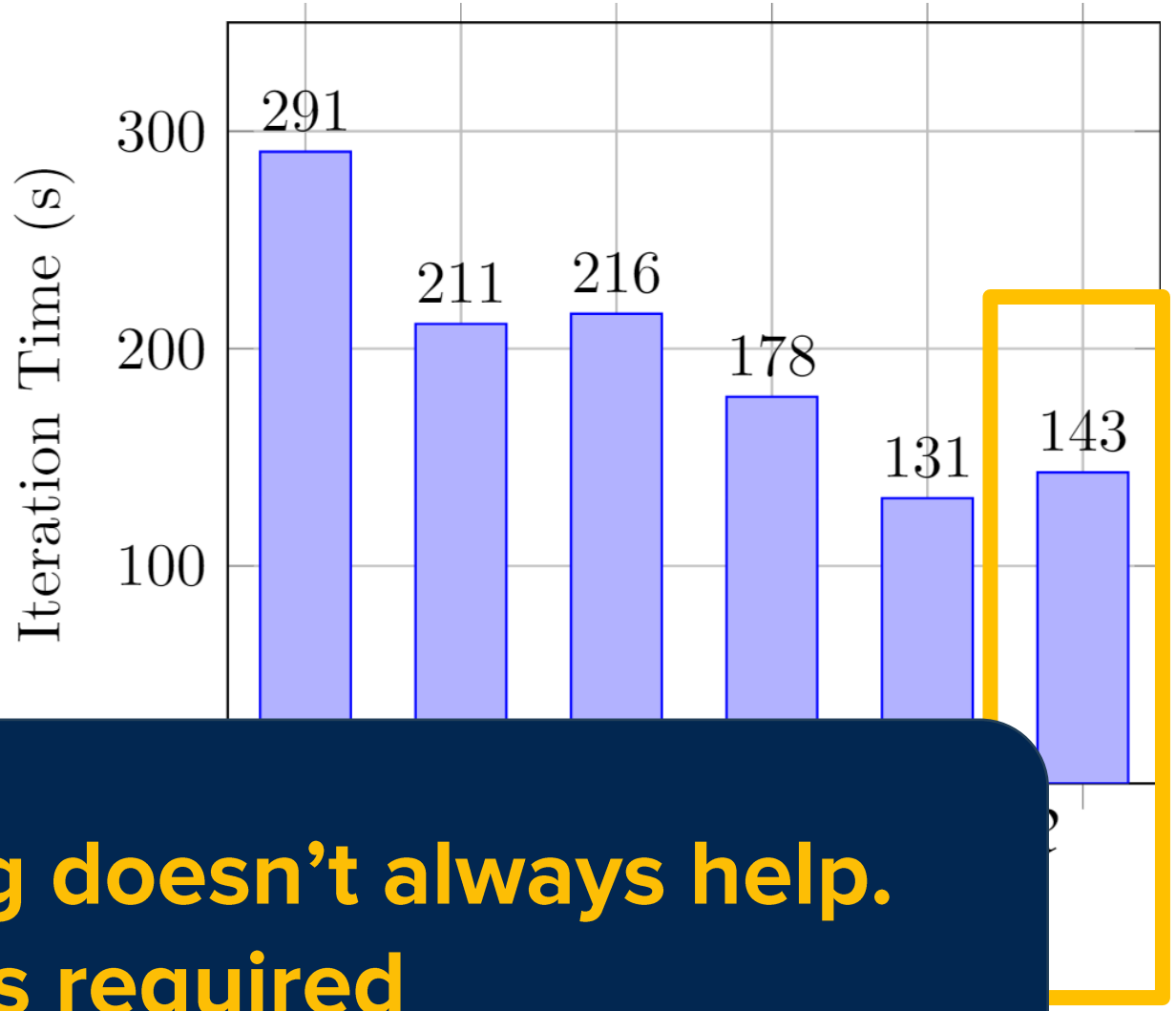
Take away: Smart data placement and movement reduces movement and increases efficiency

UCDAVIS COMPUTER SCIENCE

Results for DNNs

Three optimizations:

- Memory freeing (M)
- **`retire`**
- Allow local accesses (L)
- Prefetching (P)



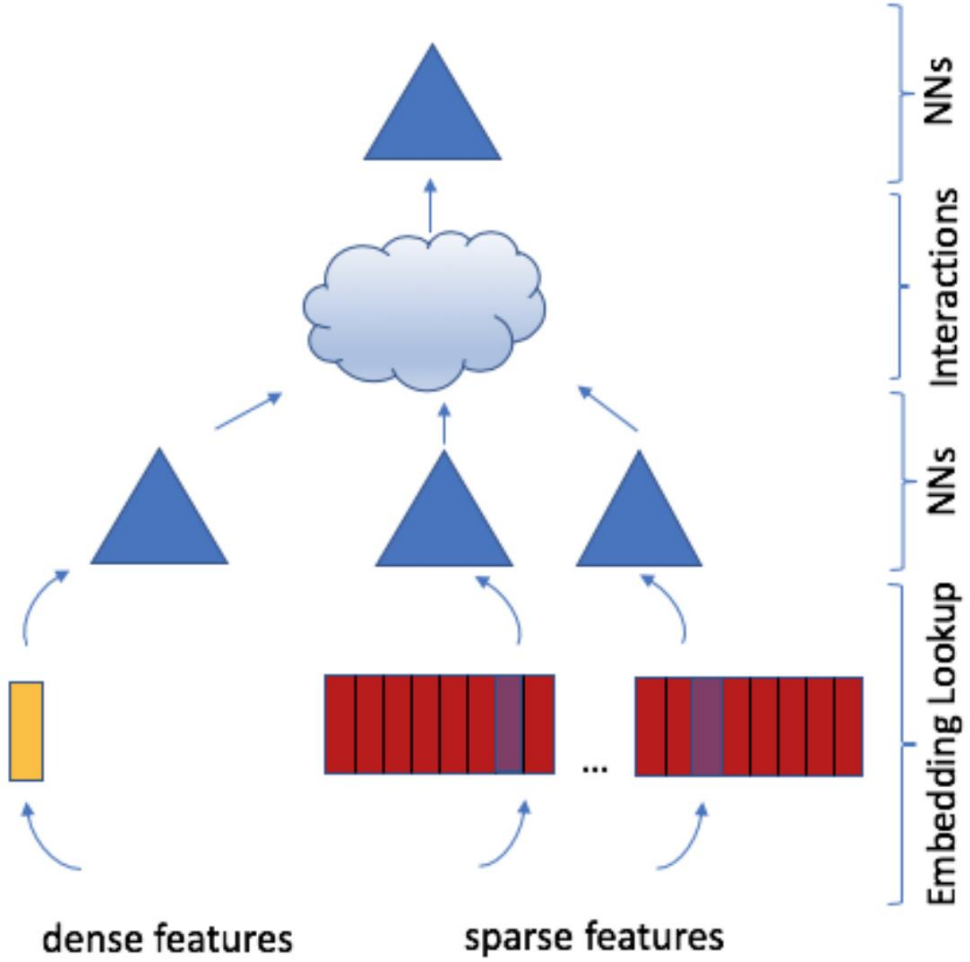
Take away: Prefetching doesn't always help. Flexibility is required

UCDAVIS COMPUTER SCIENCE

Results for DLRM (sparse accesses)

DLRM: Deep learning recommendation model

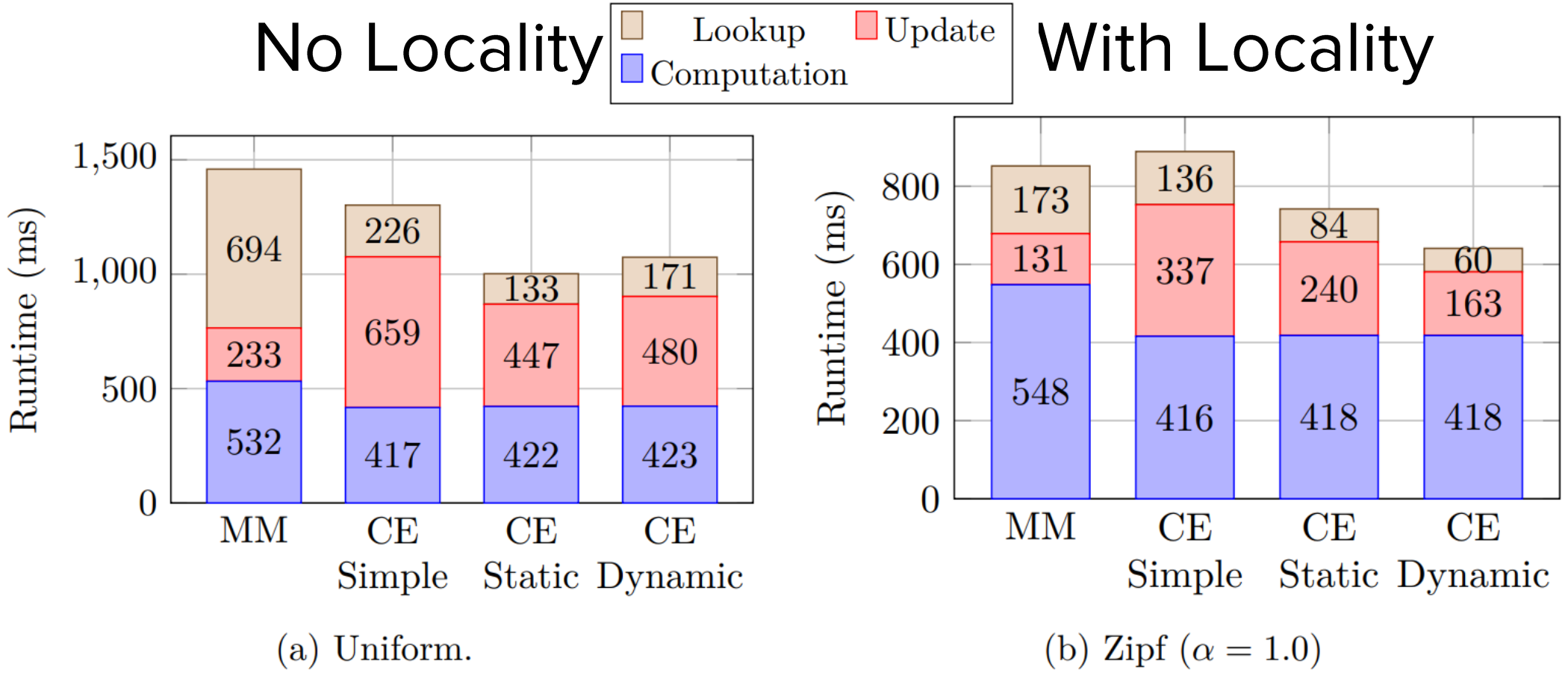
Very large embedding tables which are sparsely accessed



Hildebrand et al. Efficient Large Scale DLRM Implementation on Heterogeneous Memory Systems https://doi.org/10.1007/978-3-031-32041-5_3

UCDAVIS COMPUTER SCIENCE

Results for DLRM (sparse accesses)



Hildebrand et al. Efficient Large Scale DLRM Implementation on Heterogeneous Memory Systems https://doi.org/10.1007/978-3-031-32041-5_3

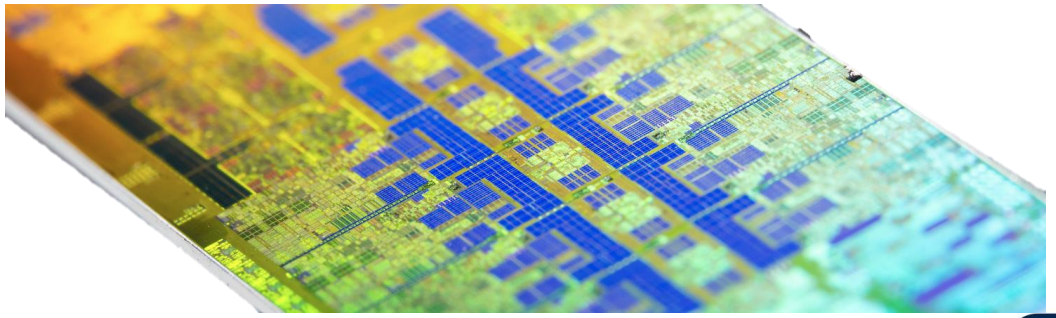
UCDAVIS COMPUTER SCIENCE

Future work

Apply ideas to remote memory

Implement in PyTorch instead of Julia

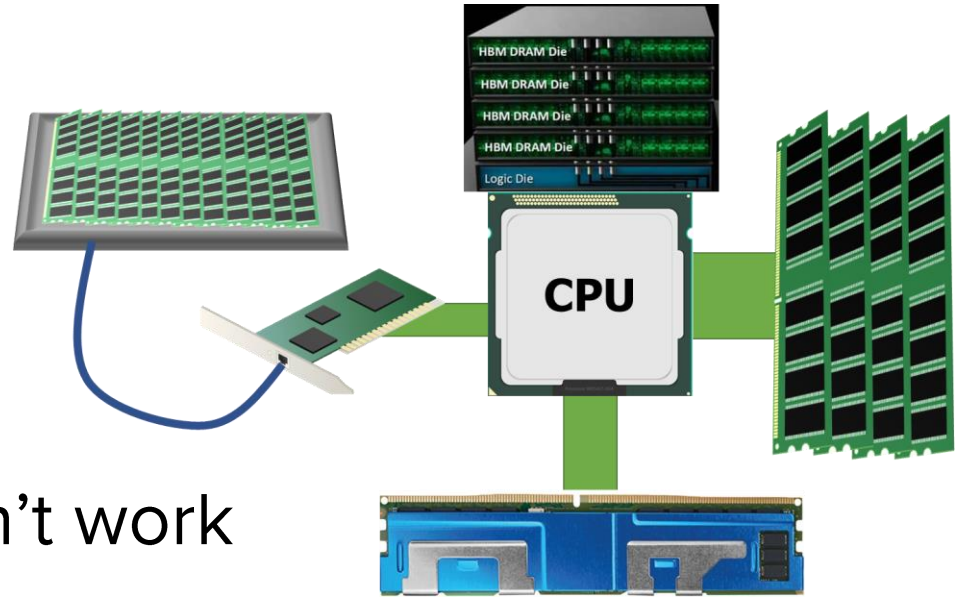
Hardware support and acceleration



UCDAVIS
COMPUTER SCIENCE

Conclusions

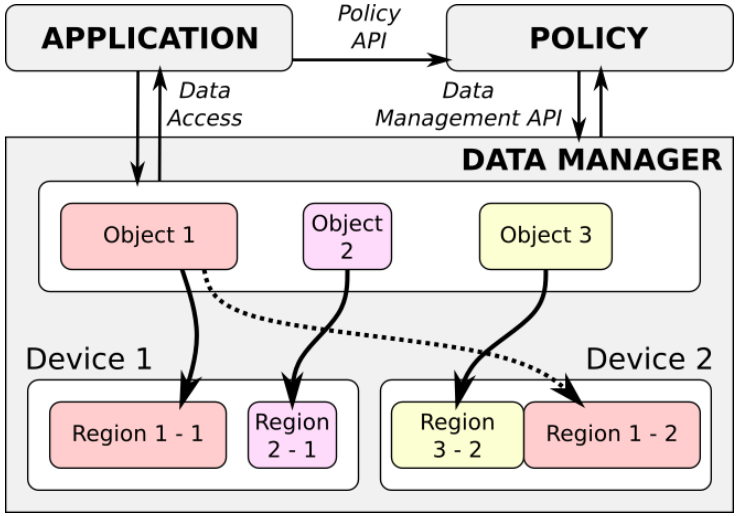
Memory systems are heterogeneous



Naively applying yesterday's solutions doesn't work

Hardware-software co-design is the future

- Involving the program, not the programmer
- Co-design data movement and placement



Just the first step: Many optimizations possible!

UCDAVIS COMPUTER SCIENCE

Why AutoTM wins Or, why DRAM caches lose

1. DRAM cache is direct-mapped
   - Misses miss for no good reason
   - “Conflict” misses, but increasing associativity is hard

2. Bandwidth is poorly utilized on misses
   - Extra accesses
   - Traffic is poorly “shaped”

3. AutoTM can skip dead writebacks
   - Temporary data that will be written before it is read

Software to the rescue

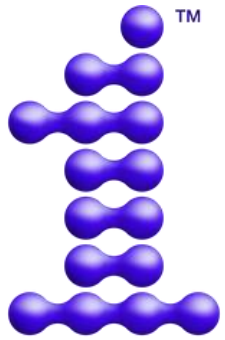
Key idea: "programmers" know a lot about their workloads

↳ Read: compilers, runtimes, experts…

Hardware-managed caches suffer from
frequent inefficient management
fine-grained unpredictable accesses

Programs operate on **objects**

Can exploit statically **predictable** accesses



oneAPI

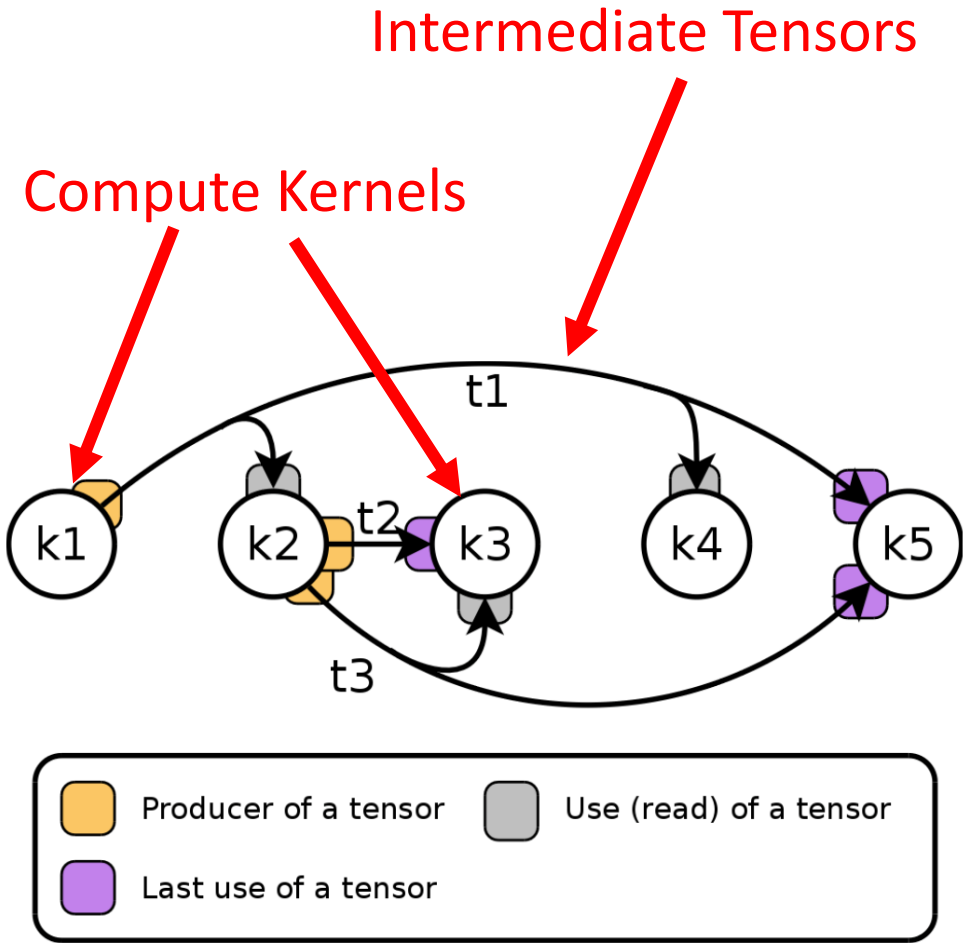
UCDAVIS
COMPUTER SCIENCE

AutoTM

Goal: Minimize Execution Time
- Arbitrary computation graph.
- Size constraint on **fast** memory.

How
- Place tensors in **fast** or **slow** memory.
- Optimal tensor **movement.**

Strategy
- Profile kernel performance
- Model tensor assignment as ILP



Experiments!

Software

- Modified the *ngraph*[1] compiler.
- Julia's *JuMP*[2] package for ILP modeling.
- Gurobi[3] as the ILP solver.

Hardware

- 1.5 TB Optane™ DC PMM
- 384 GiB DRAM

Workloads 

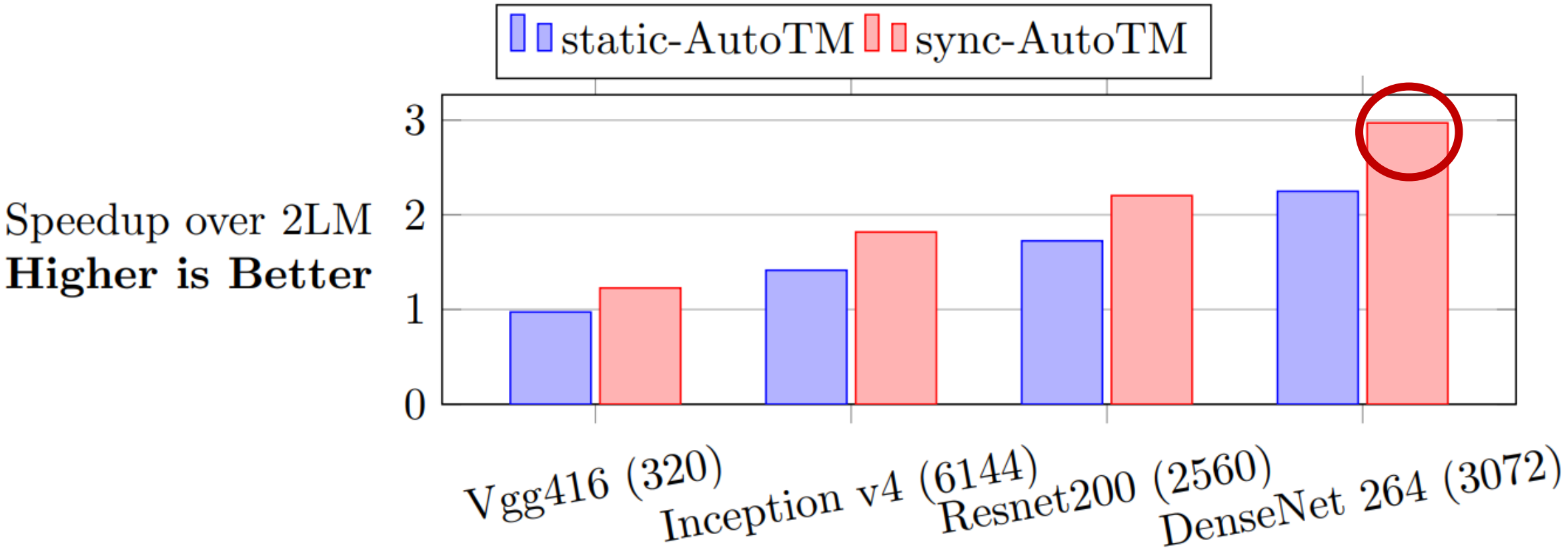
Conventional	Batchsize	Memory (GB)
Inception v4	1024	111
Vgg 19	2048	143
Resnet 200	512	132
DenseNet 264	512	115

Large	Batchsize	Memory (GB)
Inception v4	6144	659
Vgg 416	128	658
Resnet 200	2560	651
DenseNet 264	3072	688

UCDAVIS COMPUTER SCIENCE

Perf compared to DRAM Cache

Up to 3x performance over hardware cache!



UCDAVIS COMPUTER SCIENCE