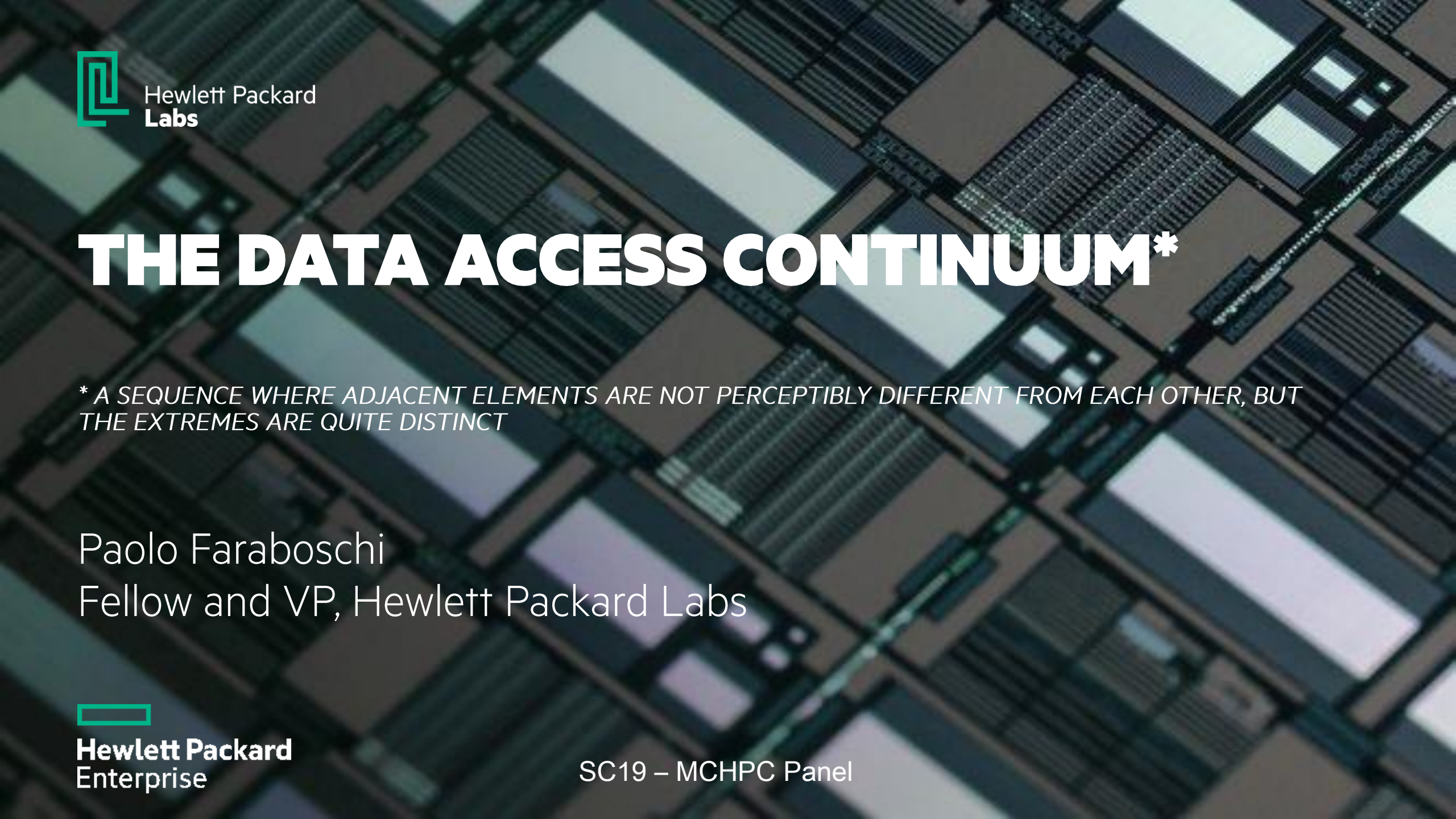


Hewlett Packard Labs

THE DATA ACCESS CONTINUUM*

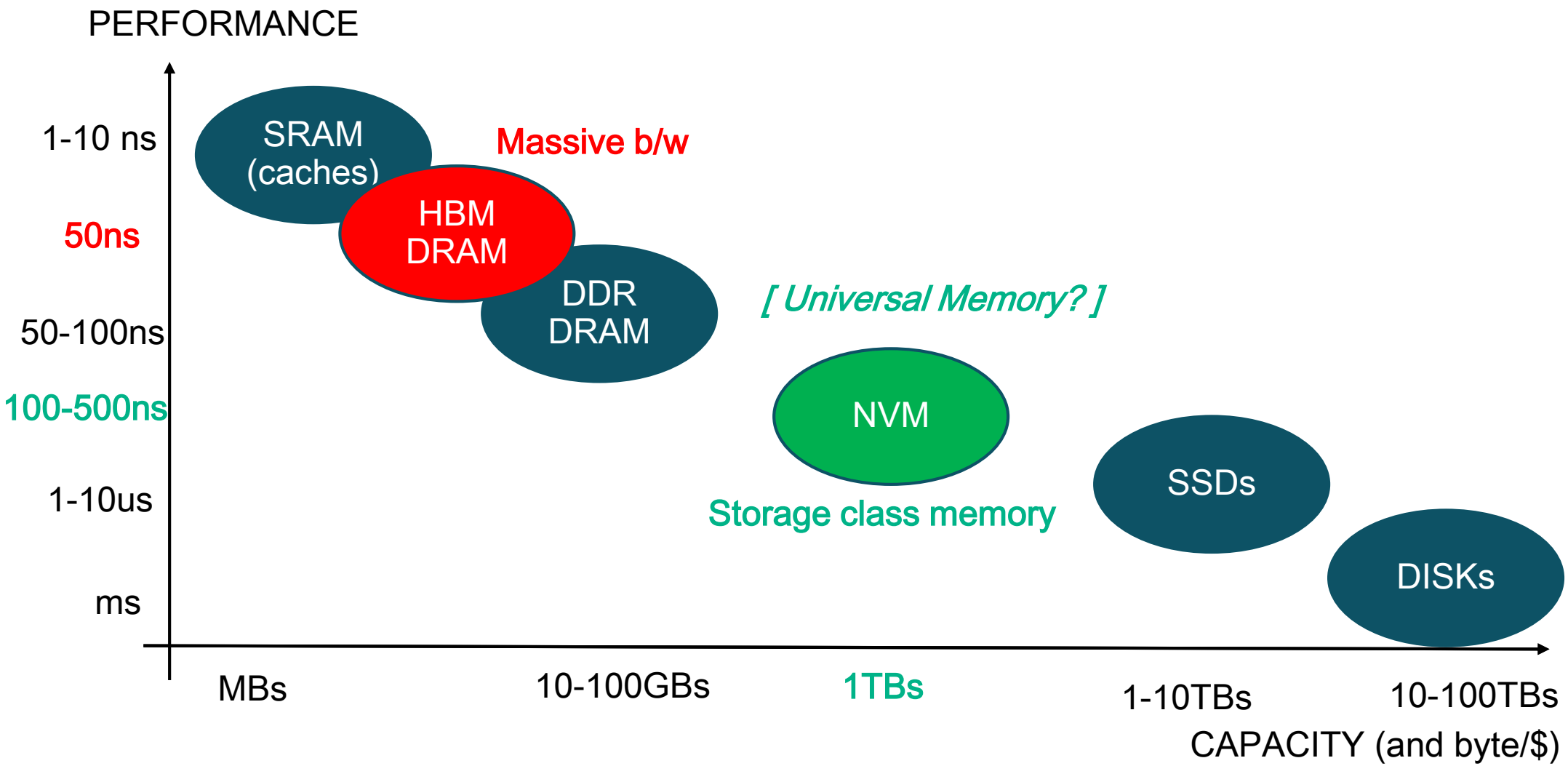
** A SEQUENCE WHERE ADJACENT ELEMENTS ARE NOT PERCEPTIBLY DIFFERENT FROM EACH OTHER, BUT THE EXTREMES ARE QUITE DISTINCT*

Paolo Faraboschi
Fellow and VP, Hewlett Packard Labs

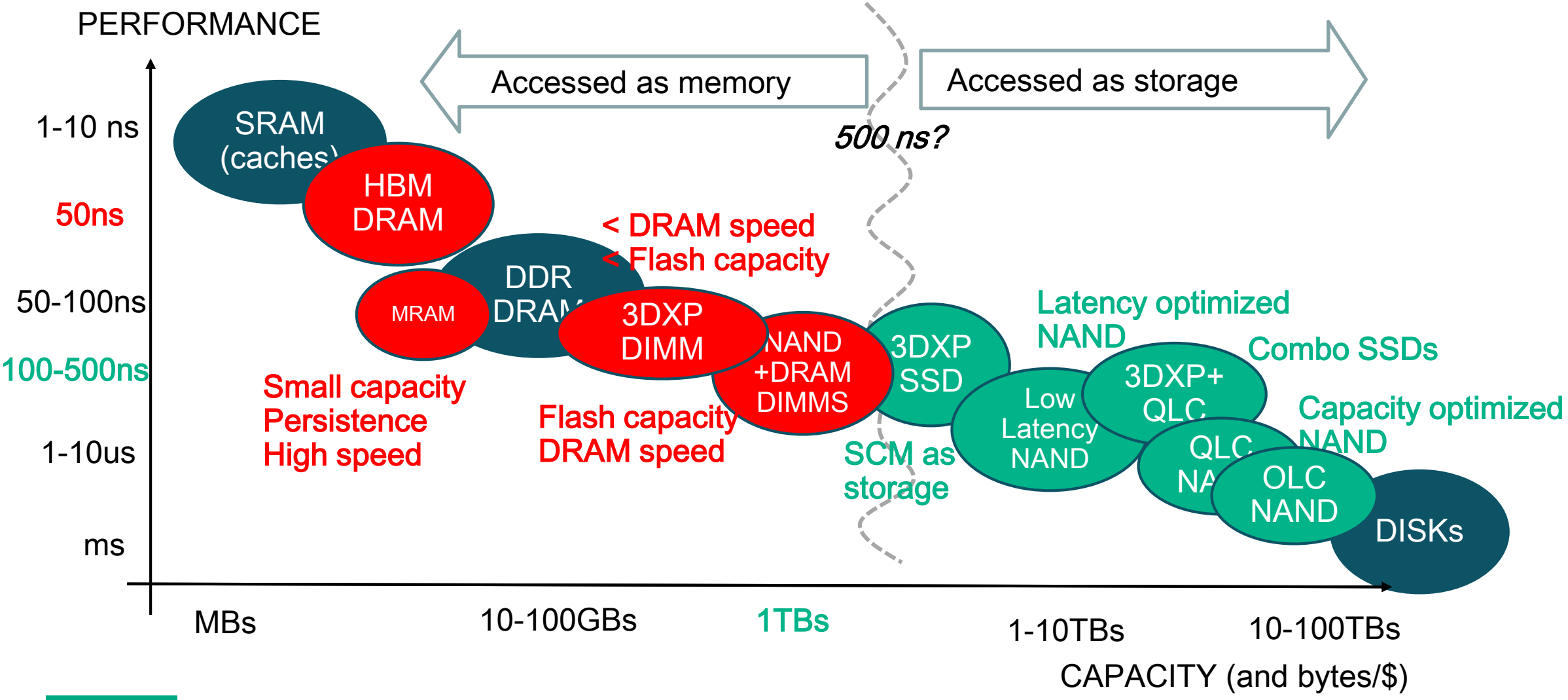
Hewlett Packard Enterprise

SC19 – MCHPC Panel

DATA ACCESS HIERARCHY CA. 2014 – TWO NEW ENTRIES



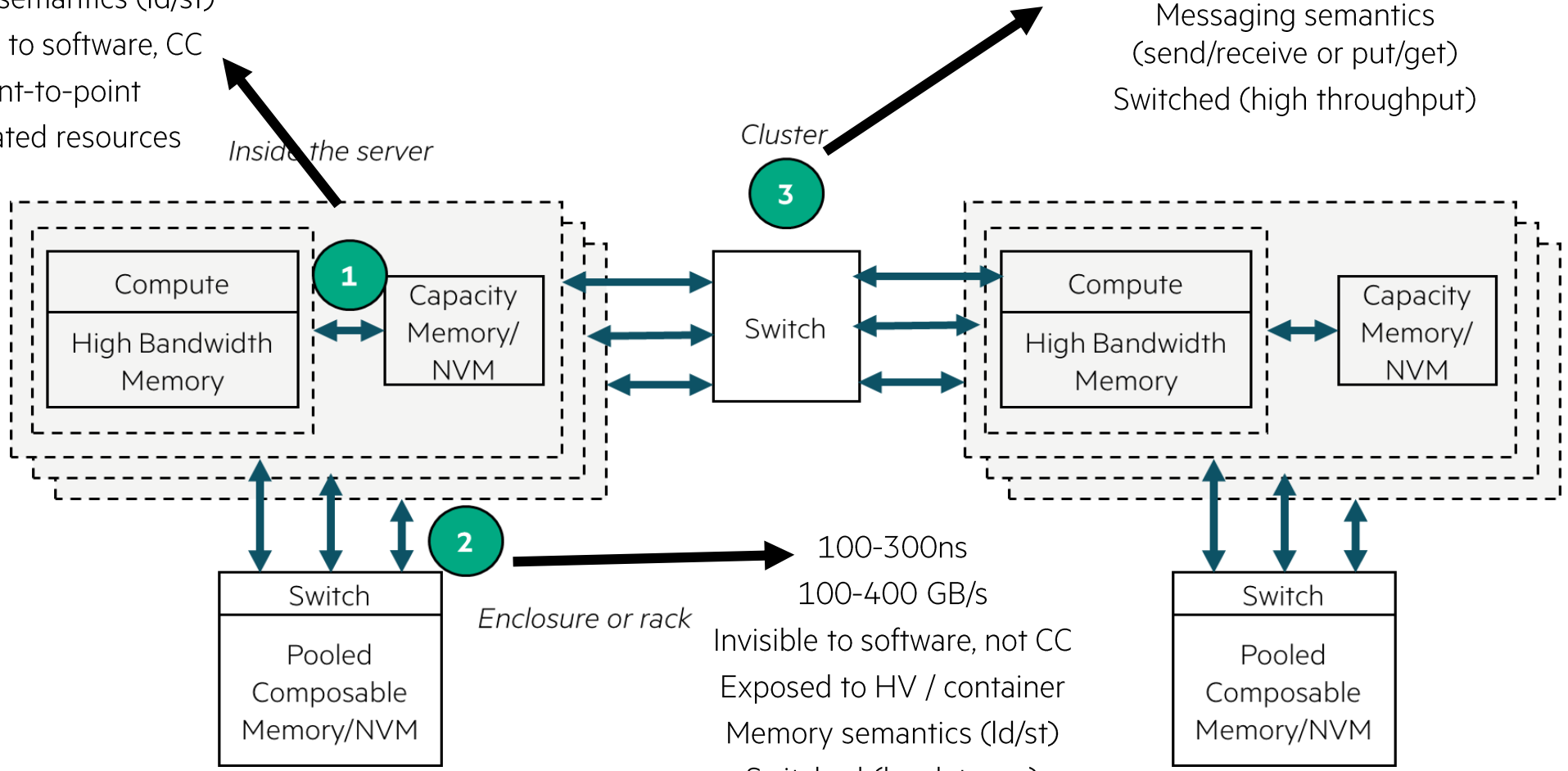
DATA ACCESS HIERARCHY IN 2019 – GETTING WORSE!



DATA ACCESS INTERCONNECTS

30-100ns
200-500 GB/s
Memory semantics (ld/st)
Invisible to software, CC
Point-to-point
Dedicated resources

300ns-1,000ns
25-100 GB/s
Software-exposed (libfabrics)
Messaging semantics
(send/receive or put/get)
Switched (high throughput)



100-300ns
100-400 GB/s
Invisible to software, not CC
Exposed to HV / container
Memory semantics (ld/st)
Switched (low latency)
Shared resources

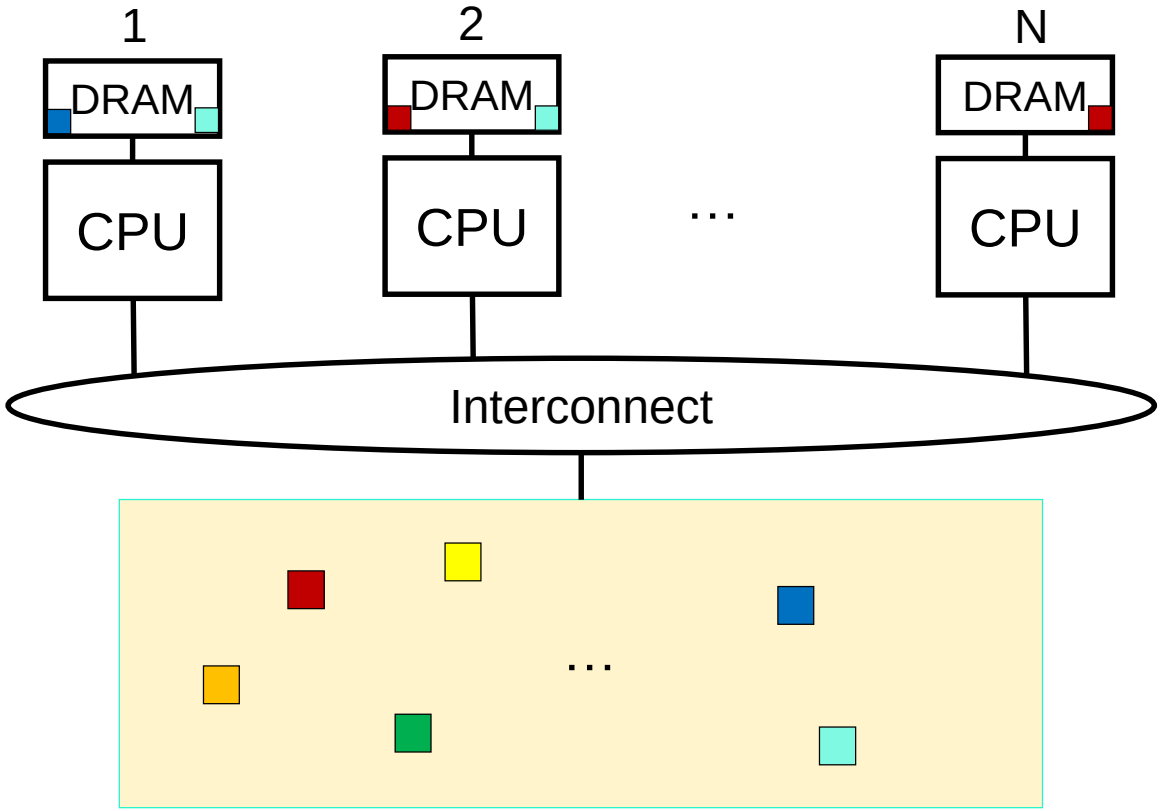
TRADITIONAL DATA STRUCTURES ARE ILL-SUITED

- How to design data structures and algorithms for the “data continuum”?
- Memory tiers provides ample capacity, but are further away
- Concurrent cached accesses without coherence may cause stale data
- Near memory data structures
  - Assume small, homogeneous access time to memory
- NUMA-aware data structures
  - Discriminate only between local memory, rely on hardware cache coherence
  - Simplifying assumptions (e.g., two levels, cache coherence) are short lived
- Traditional distributed data structures
  - Assume memory is operated on by local processor
  - Remote processors access memory via two-sided accesses (RPCs)
- Need to design new “distance-avoiding” data structures for far memory

FAR MEMORY DATA STRUCTURES

1. *Far data: truth in far memory*
   - Core content of data structure in far memory, (possibly) persistent
2. *Caches at data-structure clients*
   - (Ephemeral) data structure-specific caches
3. *Algorithm for operations*
   - Executed by clients to access data structure, using hardware (not RPCs)

- Goal: O(1) far memory accesses
  - Target: single round trip per operation
- Idea: trade far accesses for near accesses
  - Optimize pointer chasing in far memory
  - Cache some data intelligently at clients
  - Efficiently support data sharing with caching

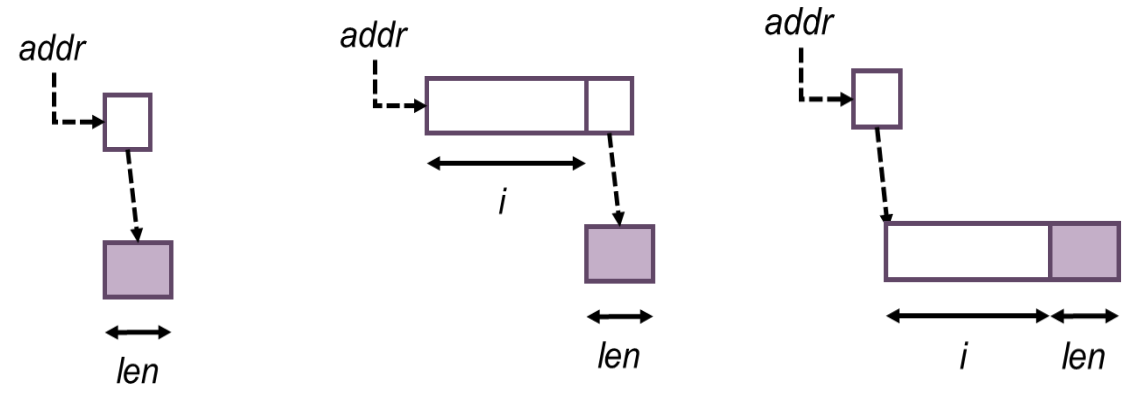


Far memory data structures

M. Aguilera, K. Keeton, S. Novakovic, S. Singhal
Designing Far Memory Data Structures: Think Outside the Box
HotOS 2019

SIMPLE HARDWARE EXTENSIONS CAN HELP

Indirect Addressing




```
tmp = *addr;
```

```
tmp = *(addr + i);
```

```
tmp = *(addr) + i;
```

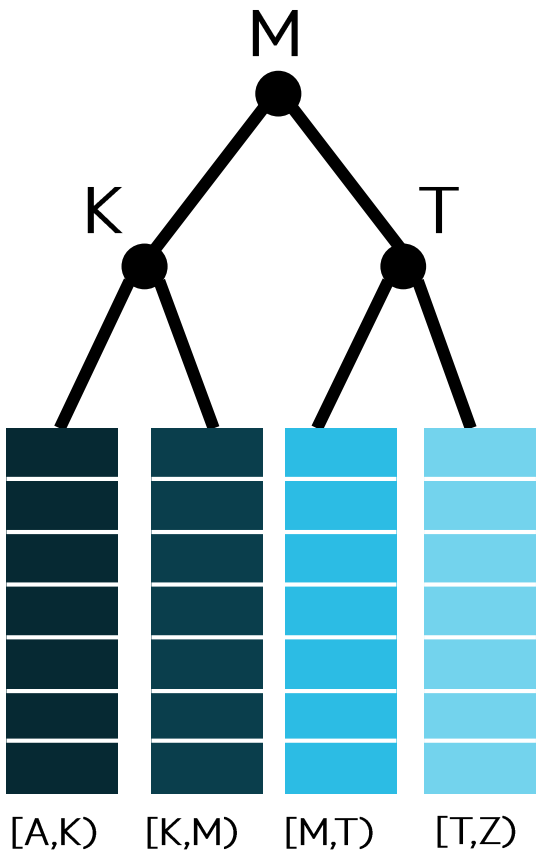
- `load(addr, len) { return *tmp; }`
- `store(addr, val, len) { *tmp = val; }`
- `add(addr, val, len) { *tmp += val; }`

Notifications

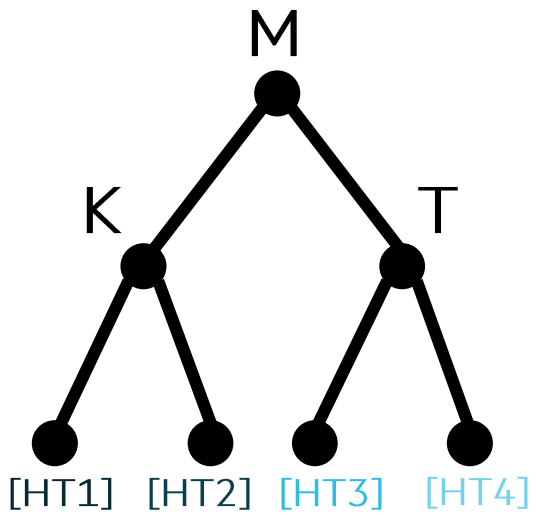
- Callback triggered when data changes, avoids remote probing
  - Useful for invalidating or updating cached data or metadata at clients
  - `notify0(addr, len)`: signal change in *`[addr, addr + len)`*
  - `notifye(addr, val, len)`: signal when *`addr`* set to *`val`*
  - `notify0d(addr, len)`: signal change in *`[addr, addr + len)`*, return data

EXAMPLE: HASH TABLE TREE

#1 Far memory:
tree of hash tables



#2 Near memory:
tree only



#3 Use local tree to
find hash table
Go to hash bucket
in far memory

Advantages

- Quick split of hash table
- One far access per read

DISCUSSION

- The “data continuum” space is fragmenting, driven by economics
- New media and new access protocols increase heterogeneity
- Simple assumptions (e.g., coherence or 2-level mem) are short lived

- Far memory needs new data structures
- New hardware primitives can help
- Goal: one “far access” per operation

- Think outside the box!