Lecture 25:

Retrieval using binary codes

Visual Computing Systems
CMU 15-869, Fall 2013

Bitcode representation of images *

Query image

→ *b*-bit binary descriptor

* Actually: images, image tiles, or keypoints, etc.

Simple example: Hamming embedding using locality sensitive hashing

- Step 1: compute <u>full descriptor</u>
  - Examples:
    - BOW representation, HOG, SIFT, etc.
    - Full-image descriptors: tiny images, GIST, etc.

- Step 2: embed descriptor in *b*-bit hamming space using *b* random projections
  - For each input query, compute 1 bit per projection (e.g., side of hyper-plane)
  - Query is now represented as a *b*-bit string

Note: a better way to determine a better set of hash functions than random projection is to learn them from the database

Fast image retrieval using bitcodes

Query image → Compute full descriptor → Compute *b*-bit binary descriptor (embedding in hamming space) → search database of binary descriptors

Benefits of NN search in hamming space

1. **Efficient distance computation:**
   - Hamming distance: number of bits that differ between two *b*-bit codes

```
int hamming_distance(bitstring x, bitstring y) {
    return count_bits( xor(x, y) );
}
```

2. **Compact database representation:**
   - *bn* bits to store bitcodes for *n* images in database
   - Recall SIFT descriptor: 512 bits per keypoint, hundreds/ thousands of keypoints per image!

K-NN search (K=5) in hamming space:

- 12.9M elements in database
  - Each element corresponds to full-image descriptor
- Quad-core CPU

[Torralba et al. 2008]

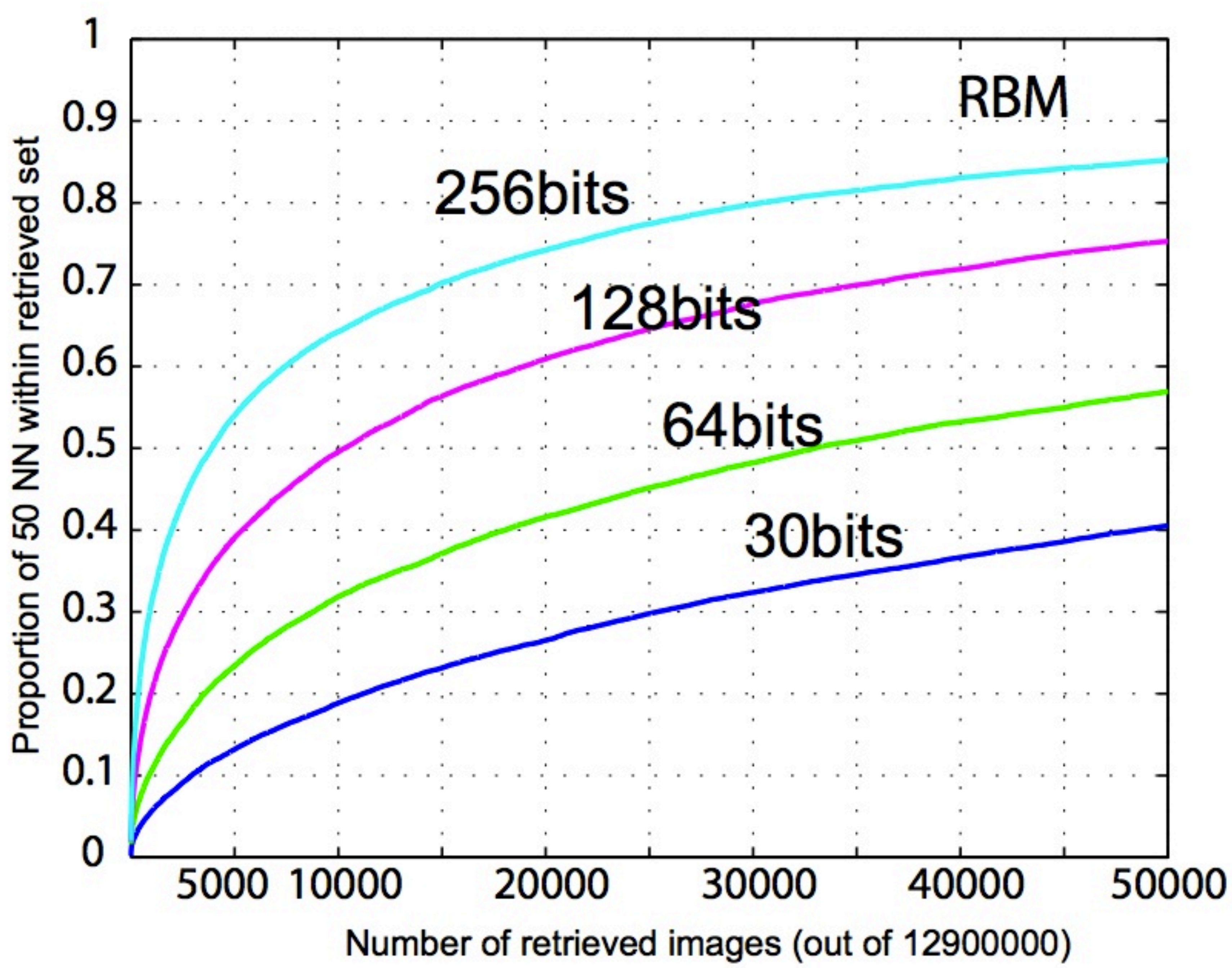
- <u>Brute-force</u> search for top 5 nearest neighbors:
  - 30-bit codes: 400 MB of memory, 74 ms
  - 256-bit codes: 3.2 GB of memory, 0.23 sec

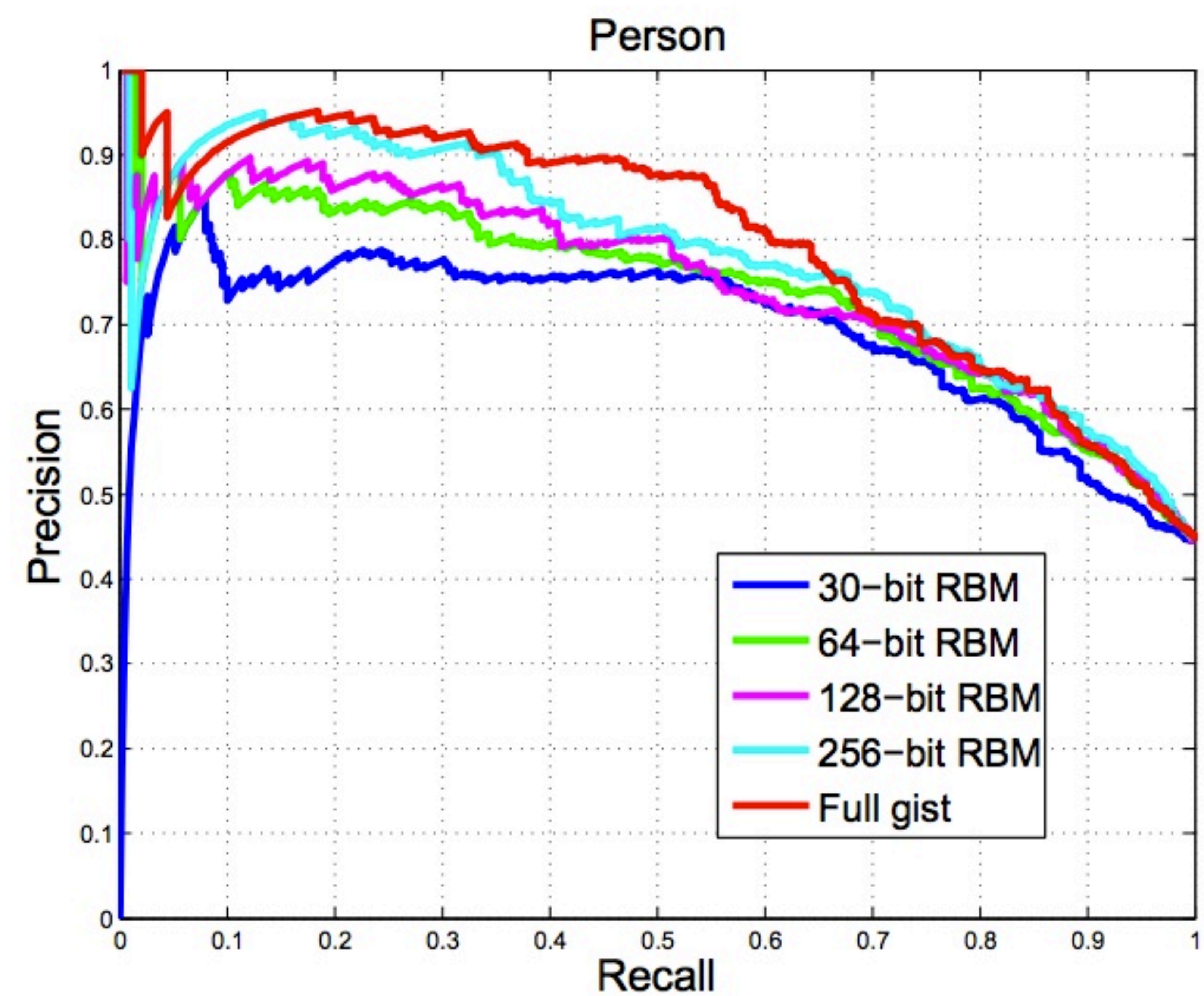
- Two orders of magnitude faster than brute force (and also K-NN tree search) on database containing full-representation GIST descriptors *

* Unfair comparison: should have compared to approximate k-NN implementation to be more fair since bitcode search results are not the same (see next slide)

Bitcode search “performance”

- Baseline: GIST full image descriptor (384 floats)
- Experiment (left): compute top 50 NN in GIST-space, then measure how many of these NN appeared in the NN results in hamming space
- Experiment (right): object detection by transferring class label (person) from NN’s to query image (does query picture contain a person?)





[Torralba et al. 2008]

Benefits of NN search in hamming space

1. **Efficient distance metric computation:**
   - Hamming distance: number of bits that differ between two *b*-bit codes

2. **Compact database representation:**
   - *bn* bits to store bitcodes for *n* images in database

3. **Potential for using binary code directly as hash table index for O(1) search**

Simple problem formulation

- Find all images within hamming distance *r* from query
- Search process: (assume 2^b indices in hash table)

```
Compute b-bit key for query
For all indices within distance r from query:
    Add images in hashtable[index] to result set
```

- Simple example: $r=0$, just check one bucket

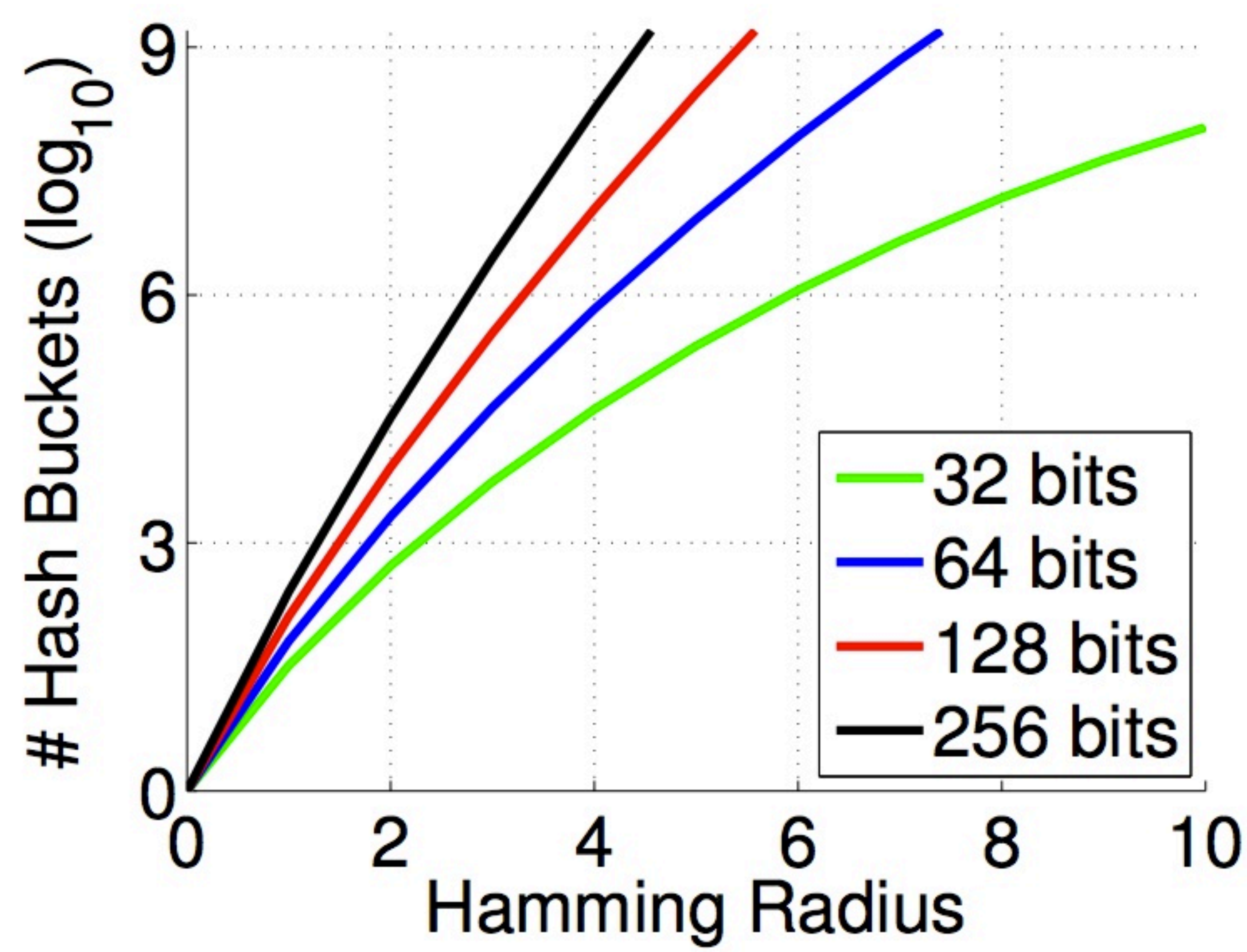
Problem

- **Number of buckets to check increases rapidly with *r***
  - Volume of the "hamming ball" of radius *r*

- **Number of candidate buckets:**

$$L(b, r) = \sum_{k=0}^{r} \binom{b}{k}$$

[Norouzi et al. 2012]



- **Example: *b*=64, then about 1B buckets for *r*=7**
  - If database is smaller than 1B elements, most of these indices will be empty
  - Consider database of millions of elements: faster to just run brute-force linear search through database!

Multi-index hashing to improve k-NN search in hamming space

[Norouzi et al. 2012]

- **Basic intuition:**
  - Divide query bit string into *m* disjoint *b*/*m*-bit substrings
  - Bit strings that are close in one of the substrings might be close overall

- **Key idea:**
  - If binary codes *x* and *y* differ by less than *r* bits, then in one of their *m* substrings they must differ by less than floor(*r*/*m*) bits.
  - Proof by pigeon-hole principle (if they differed by more than *r*/*m* bits in each substring, then overall *x* and *y* must differ by more than *r* bits

Efficient k-NN using multi-index hashing

- For each set of length-*m* substrings, find substrings of within Hamming radius of floor(*r*/*m*)

- This is a much easier problem!
  - Previously: search needed to examine $L(b, r)$ hash buckets
  - Now need to examine only $L(b/m, \lfloor r/m \rfloor)$ buckets in *m* different hash tables
  - E.g., *r*=7, *m*=4, then only need to search with radius 1 in each of the substrings

Full algorithm

- Build m hashtables using the length b/m substrings of elements in the original database

- Given b-bit query:
  - For each of the m substrings of the query:
    - Find radius floor(r/m) neighbors and add them to <u>candidate set</u> (using hashtable corresponding to current substring)
  - The candidate set is a superset of the true set of elements within hamming distance r, so compute actual set by executing full Hamming distance computation for all elements in candidate set (brute force linear scan)

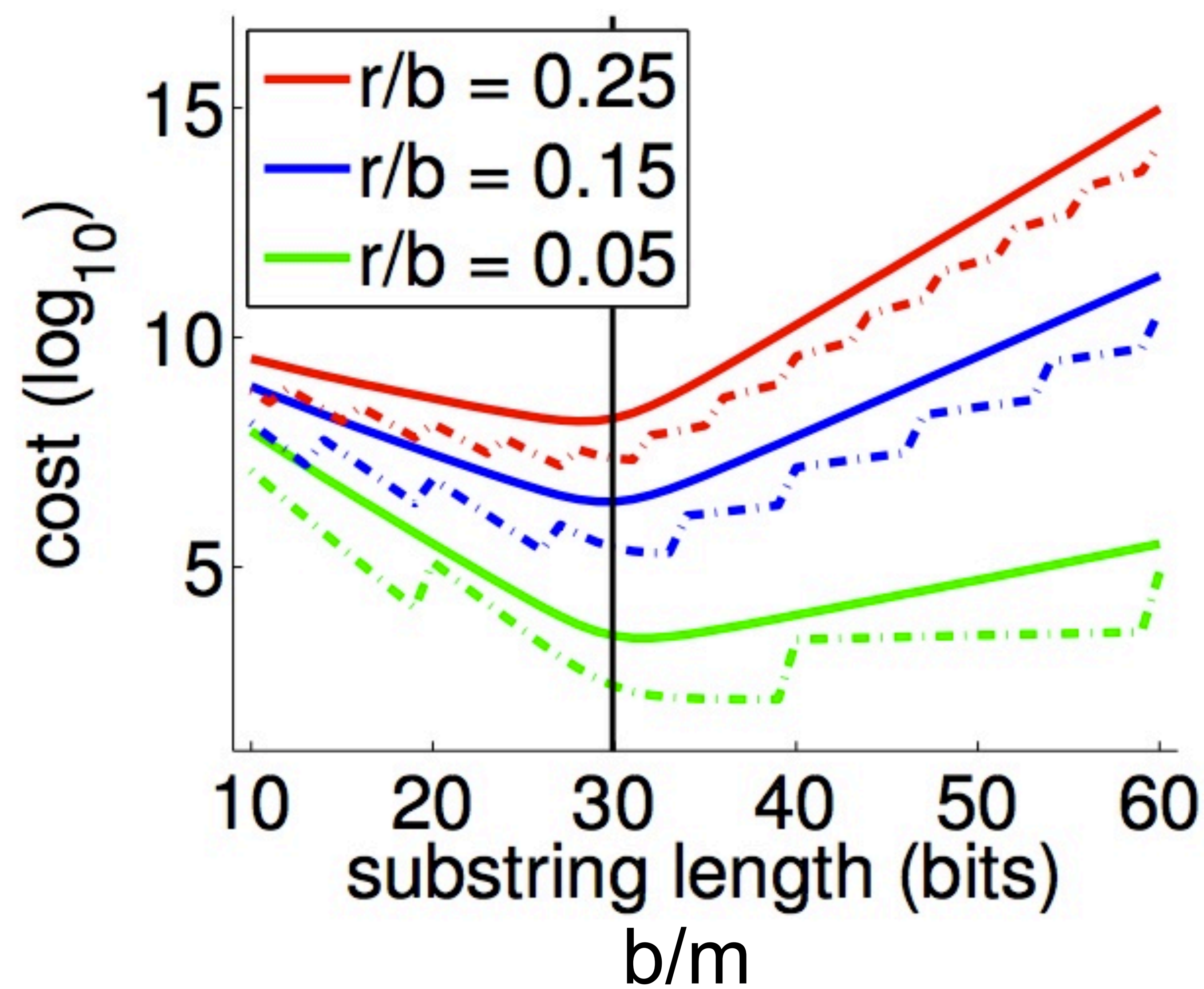
- Storage cost:
  - bn bits to represent all descriptors in hash table
  - m hash tables referring to these descriptors ($mn\lg_2 n$)
  - In practice, optimal $m=b/\lg_2 n$ so overall storage cost near linear in n

How to choose *m*?

- Trade-off between having large substrings (and thus a tight candidate set, but many bucket lookups in substring searches) and having small substrings (cheap substring search but very loose candidate set)
  - Consider *m*=*b*, substrings are of length 1, but all neighbors in candidate set!

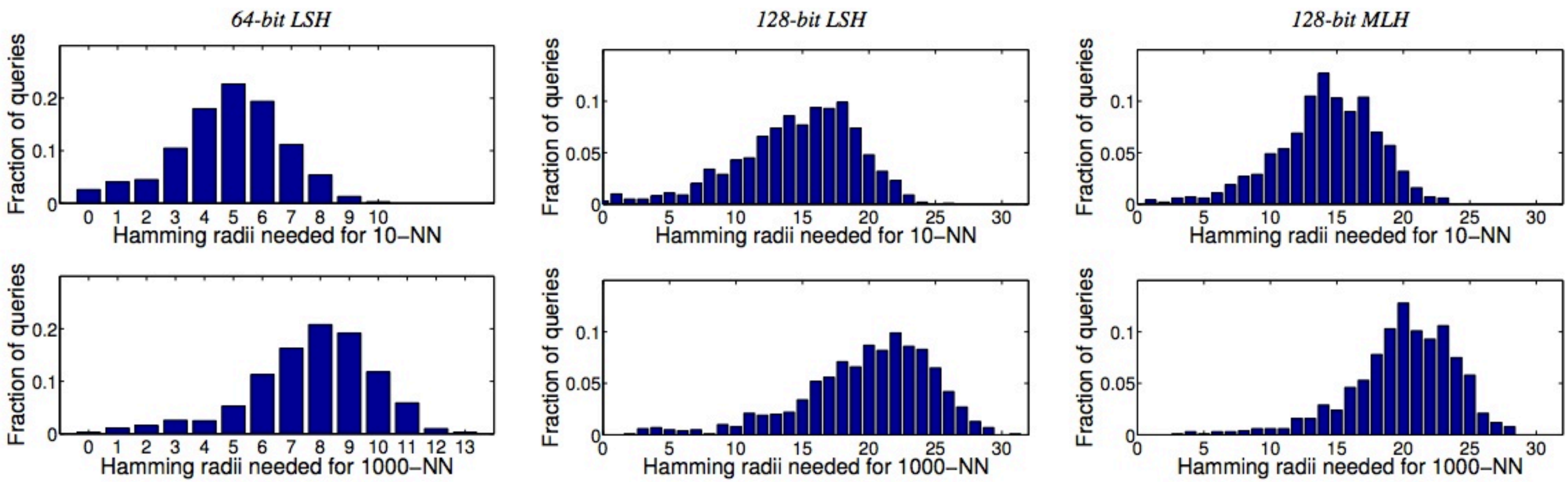
Figure at right:

- Database size: 1B descriptors
- 128-bit codes (*b*=128)



How to determine *r* from *k*?

- **Algorithm finds all database elements within Hamming distance *r*, but we often want *k* nearest neighbors to a query (not all elements within a fixed distance)**
- **Problem: binary codes not uniformly distributed across Hamming space, so cannot just pick an *r* corresponding to *k* (*r* required to contain knn depends on query)**



- **Solution: progressively increase *r* until k-NN are found.**

Fast image retrieval using bitcodes

Query image

Compute full descriptor

SIFT
SURF
HOG
etc.

Compute *b*-bit binary descriptor (embedding in hamming space)

Matrix-vector multiplication

Neural network evaluation
etc.

search database of binary descriptors

10's of thousands of hamming distance computations

Compute intensive

memory intensive

Accelerating binary code generation

- Option 1: use faster-to-compute full descriptors: e.g., SURF
- Option 2: compute binary code directly from image (not via binarization of full descriptor)

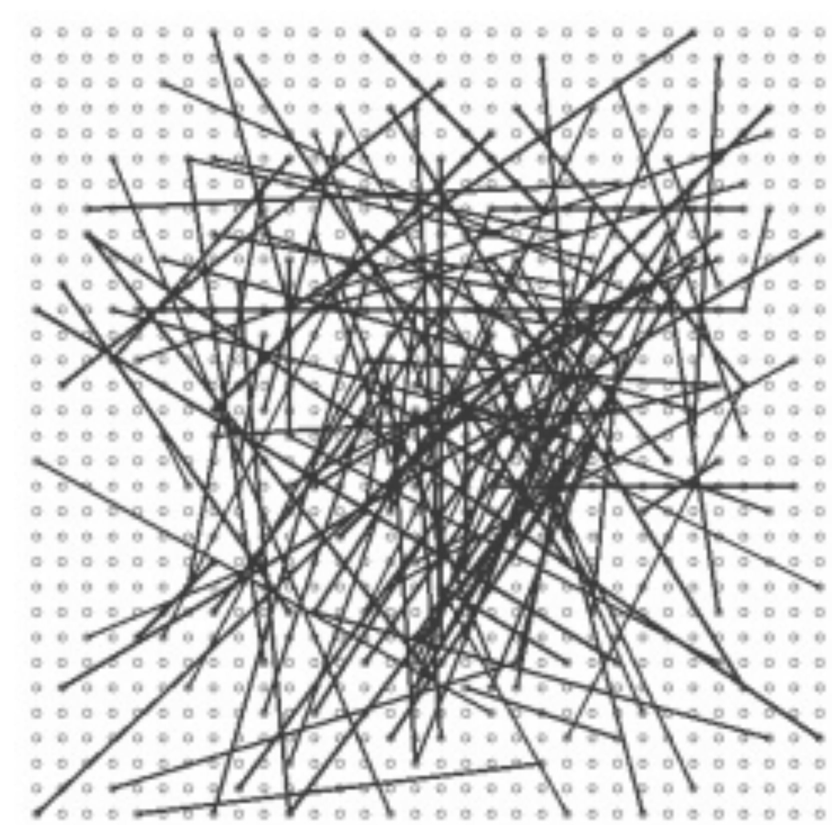
Direct computation of binary code

Query image → Compute full descriptor → Compute *b*-bit binary descriptor (embedding in hamming space) → search database of binary descriptors

BRIEF descriptor

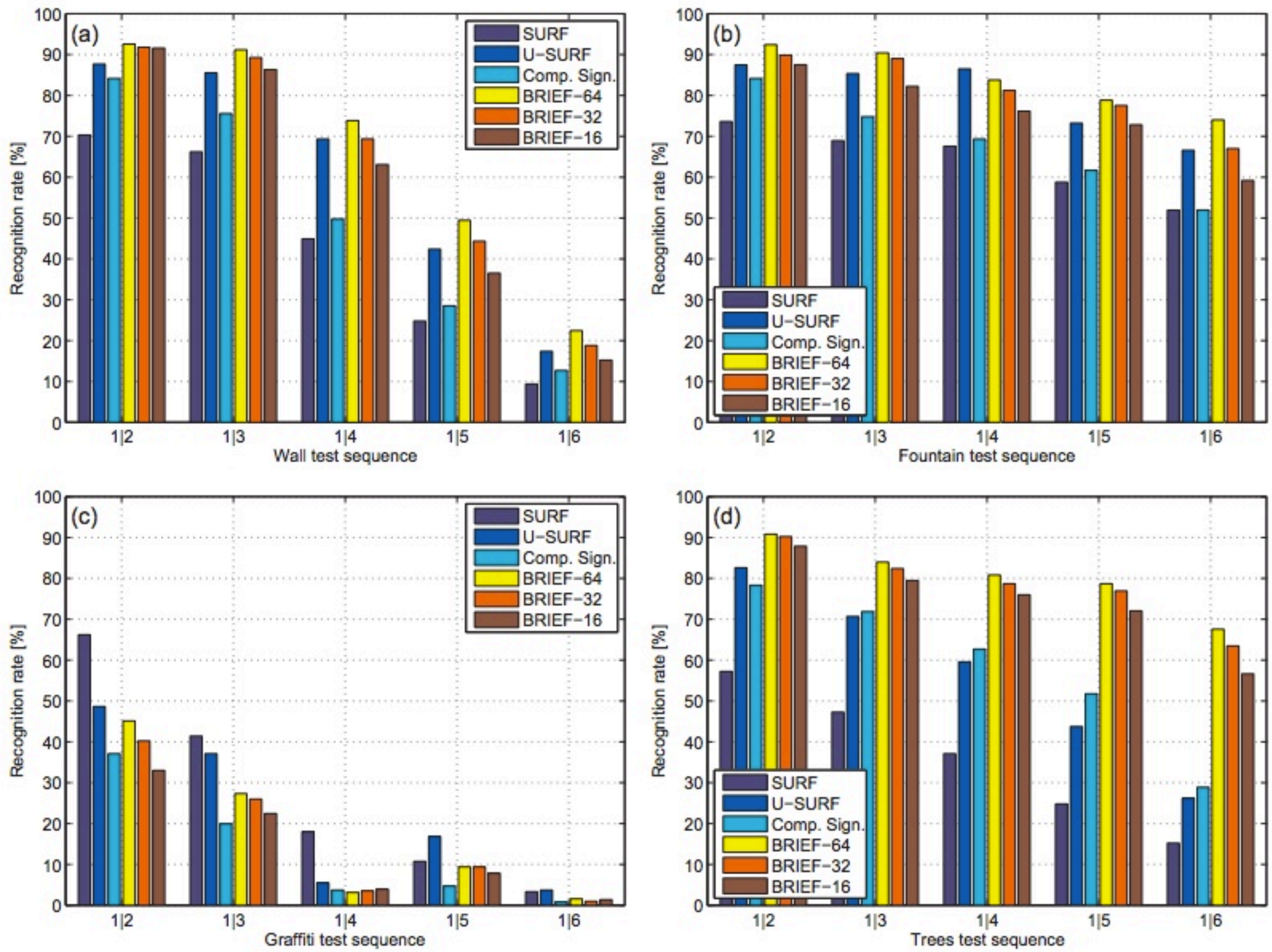
[Calonder 2010]

- Idea: compute binary descriptor for image directly
  (rather than binarize a full descriptor)
- Want descriptor computation to be fast (avoiding cost of full descriptor computation is the motivation for direct computation)
- BRIEF is a patch-based descriptor:
  - For each S x S image patch p, consider binary function $f(p, x, y)$
    - x and y are pixel coordinates in patch
    - $f(p, x, y) = 1$ if $p(x) < p(y)$, 0 otherwise
  - Algorithm:
    - Step 1: smooth image patch using 9x9 pixel gaussian kernel
    - Step 2: to compute each bit b, evaluate $f(p, x_b, y_b)$
      - $(x,y)_b$ point pairs chosen at random from gaussian distribution centered at patch center



BRIEF "performance"

- Experiment: find % of NN that match ground truth NN
- Note: BRIEF-64 is eight times more compact than full SURF descriptor (64 floats)



Fast image retrieval using bitcodes

Query image

Compute full descriptor

Compute *b*-bit binary descriptor (embedding in hamming space)

search database of binary descriptors

10's of thousands of hamming distance computations

memory intensive

- Example: Hamming distance for 64-bit code
  - 64 bit xor
  - 64-bit pop count (popcnt)
  - 16 bytes of input, 2 CPU instructions
- 4 cores at 3 GHz: 6B distance computations per second
  - 96 GB/sec of required bandwidth

Bandwidth cost of search

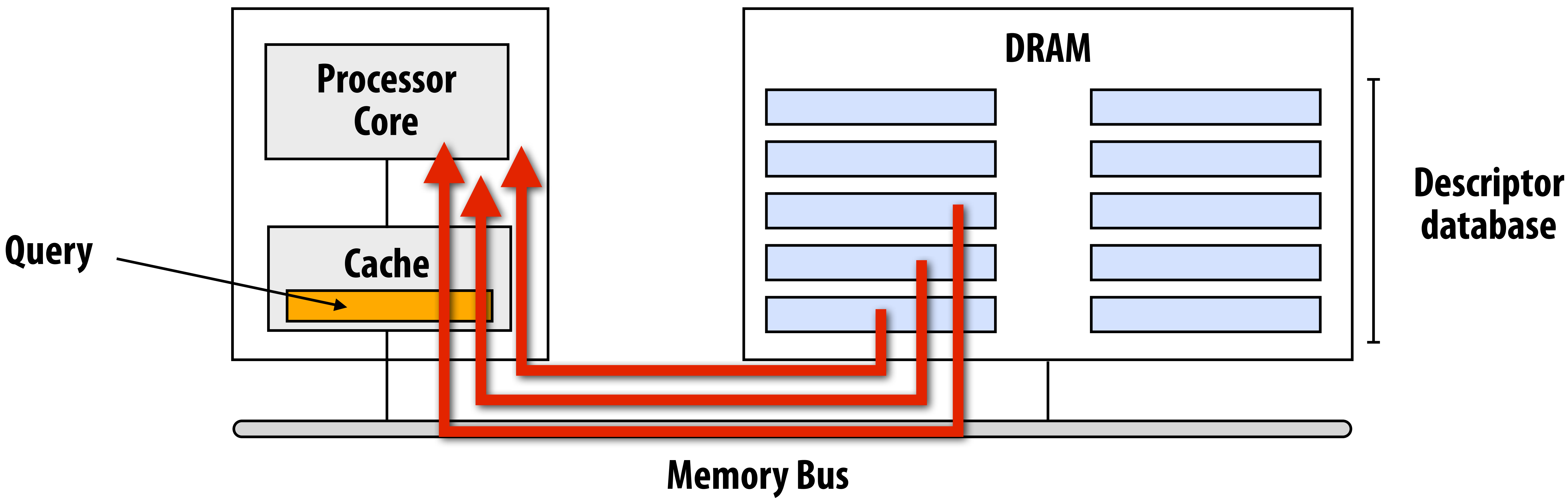
- Back-of-the-envelope calculation
  - 30 fps video, 1000 descriptors per frame
  - 64 bit descriptors (small)
  - 100M element database (800MB database)
  - 100,000 Hamming distances per frame (.1% of database touched per query)

- System must compute 3B hamming distances per second
  - 8 bytes per distance computation (assume query is cached)
  - 24 GB/sec of bandwidth
  - 150 pJ per byte (LPDDR memory)
  - Approximately 3.84 watts just to read the data! (not counting cost of Hamming distance math or math to compute the query bitcode)
    - Modern smartphone: 5.5 watt-hour battery
    - Typically budget for mobile GPU: ~ 1 watt

Optimizations

- Caching: Exploit locality of queries
  - Hopefully back-to-back access same hash buckets (cache benefit)
- Algorithmic: batch queries
  - Execute N queries at a time
  - Requires database reuse across queries (certainly true for brute-force search, less clear when hashing techniques uses -- see point 1 above)
  - Increases query latency, to gain higher overall throughput
- Improved machine organizations?
  - Move computation closer to memory

Basic machine organization

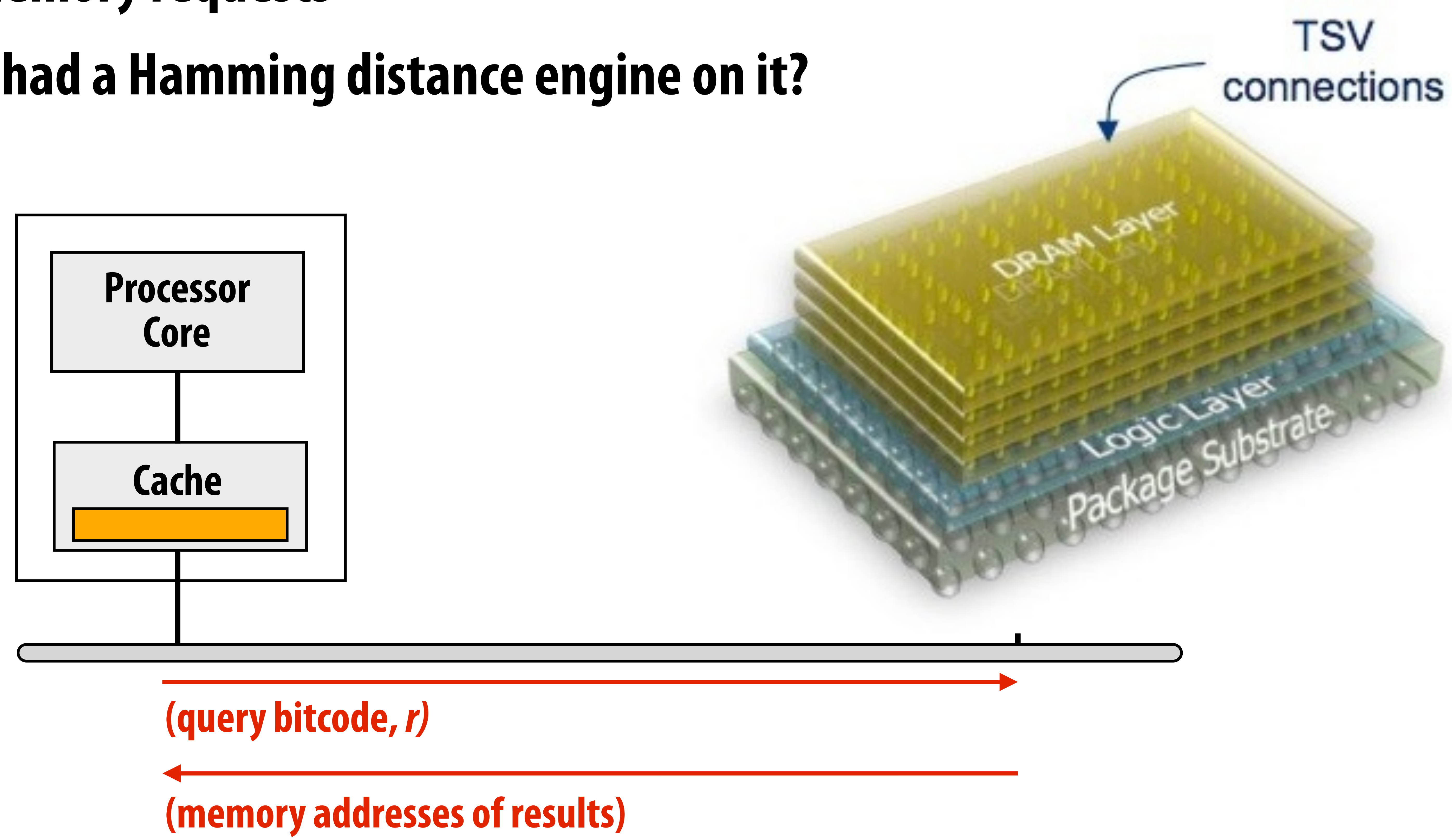


Then repeat for next query. (transfer entire database to CPU)

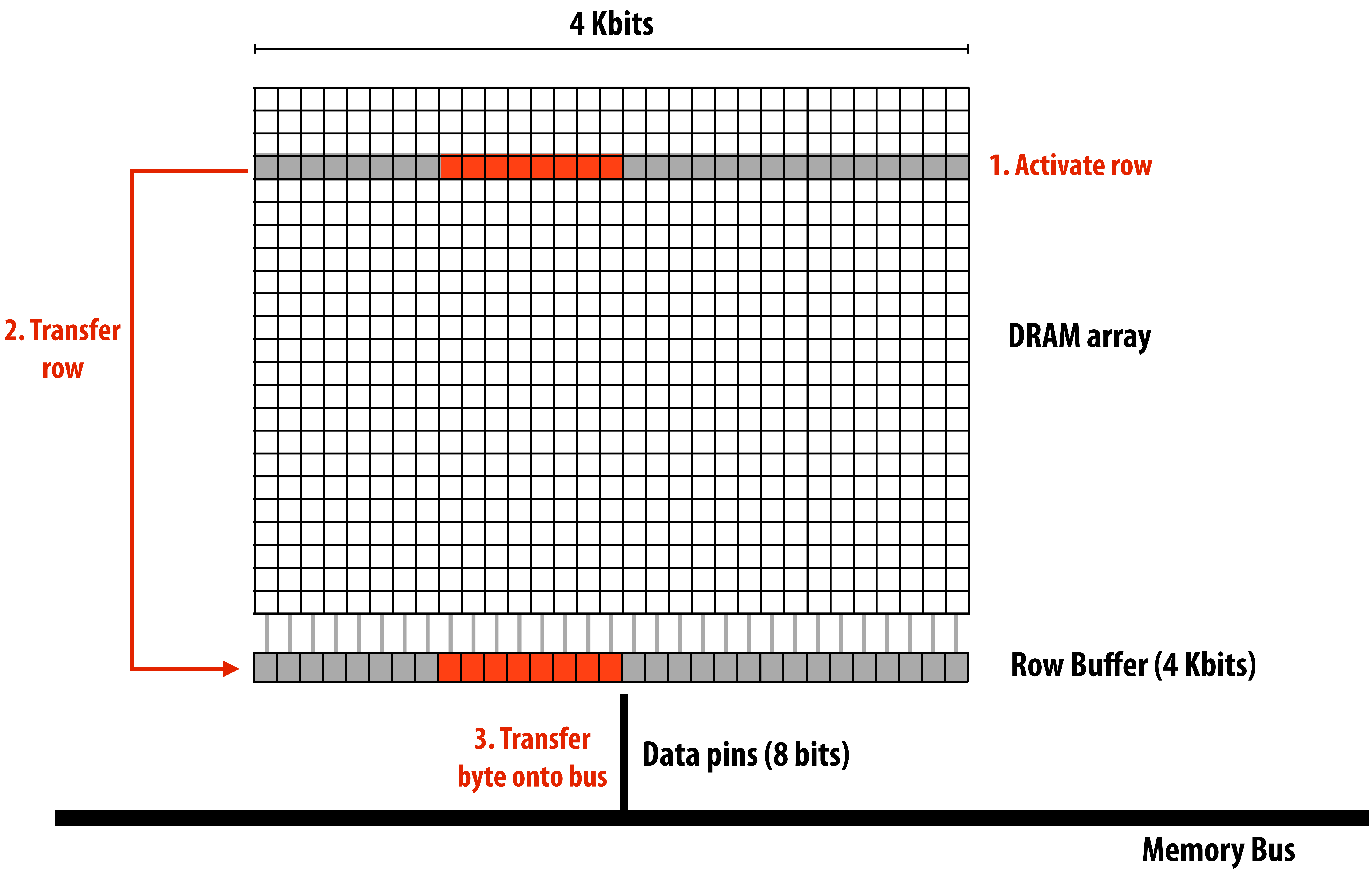
Increasing interest in avoiding transfer of all this data to CPU

Example: hybrid memory cube

- Stacked layers of DRAM
  - Through-silicon vias (TSV) connect layers
- Bottom layer is logic layer
  - Currently handles logic for managing memory cube, serving memory requests
  - What if it had a Hamming distance engine on it?



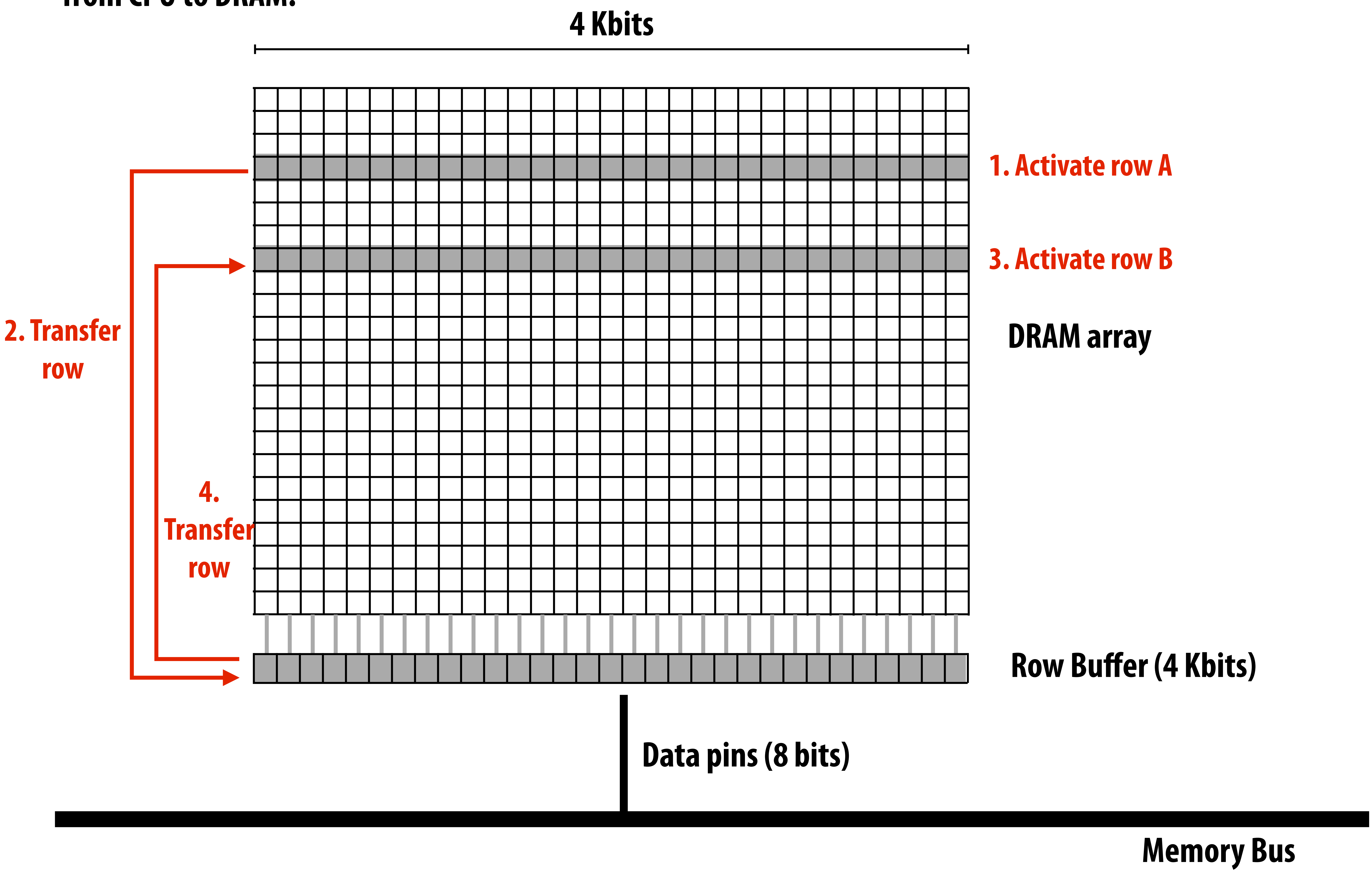
Tutorial: DRAM operation (load one byte)



RowClone: in-DRAM operations

[Seshadri et al 13]

Idea: offload simple bandwidth-heavy operations (bulk data copy and bulk data initialize) from CPU to DRAM.

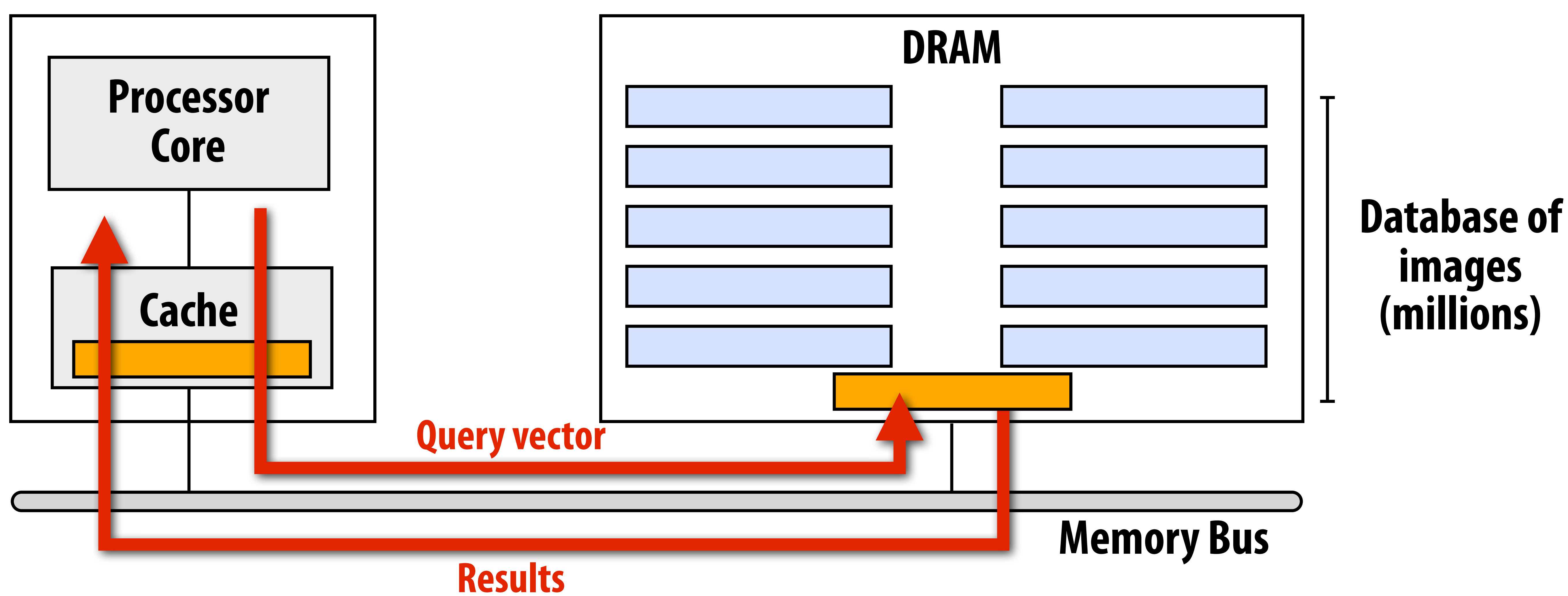


RowClone: in-DRAM operations [Seshadri 13]

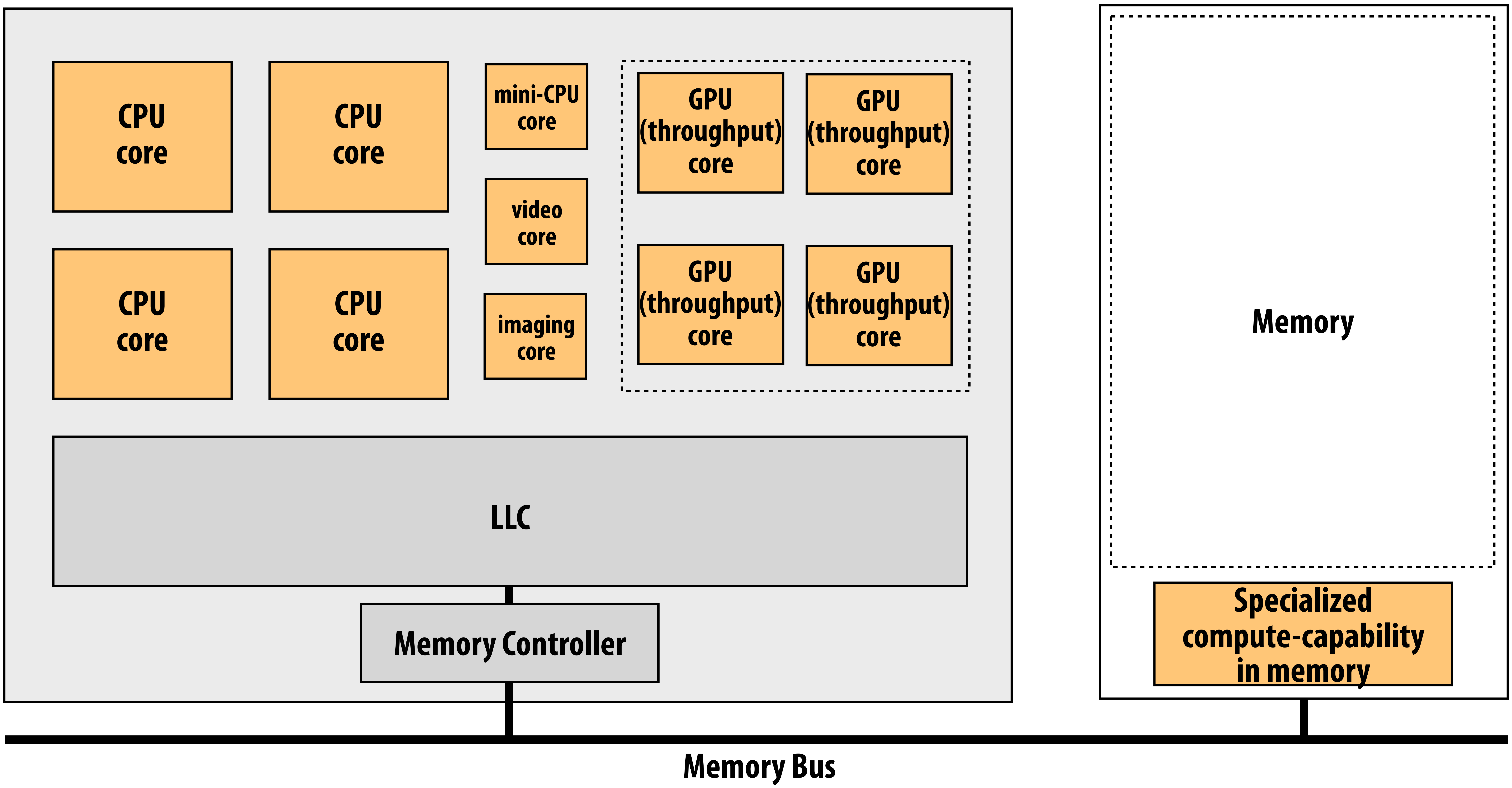
Idea: offload simple bandwidth-heavy operations (bulk data copy and bulk data initialize) from CPU to DRAM. (The operations do not require computation.)

- **Accelerates bulk copy by 11.5x**
- **Eliminates memory bus traffic: reduces energy cost by 1.5 to 74.4x**

- **Next step: move from copy (no computation) to simple computations (e.g., bit-wise operations)**
  - XOR + count bits is a Hamming distance
  - XOR seems possible, count bits much tricker
  - Memory requests: load, store, bulk copy, bulk initialize, filter by predicate

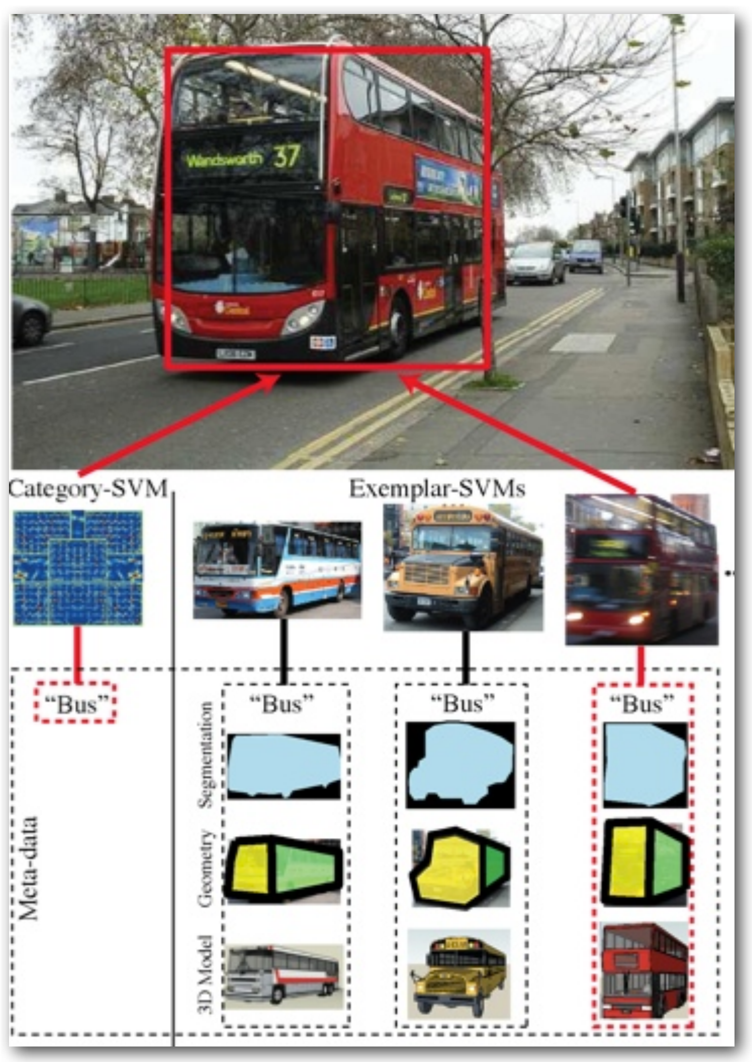


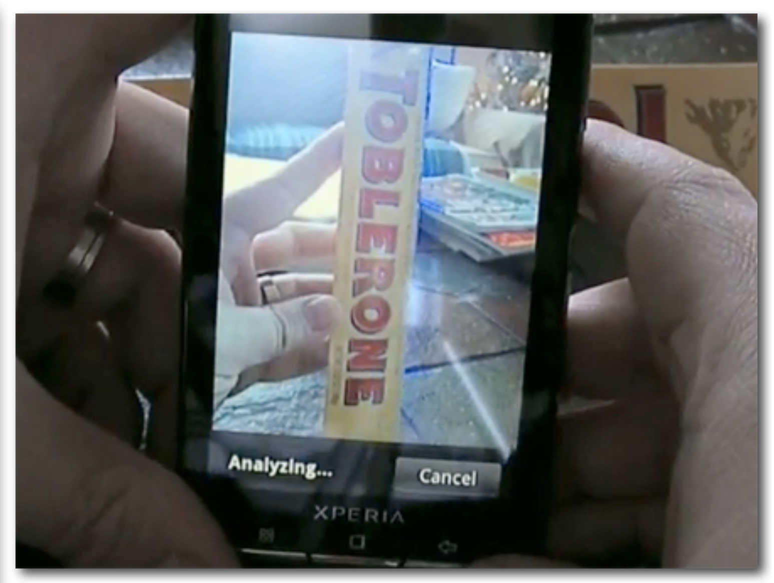
Heterogeneous parallel architecture view



Summary

- Image retrieval as a core building block of compelling visual computing applications
- We will need very efficient implementations to enable advanced new applications





Object Detection
[Malisiewicz 11]

Movement prediction [Yuen 11]

Novelty Detection

Image Matching [Shrivastava 11]