A Nearest Neighbor Data Structure for Graphics Hardware

Lawrence Cayton
Max Planck Institute, Tübingen

Problem setting

Database $X = \{x_1, x_2, \ldots, x_n\}$

Query q (or many queries Q)

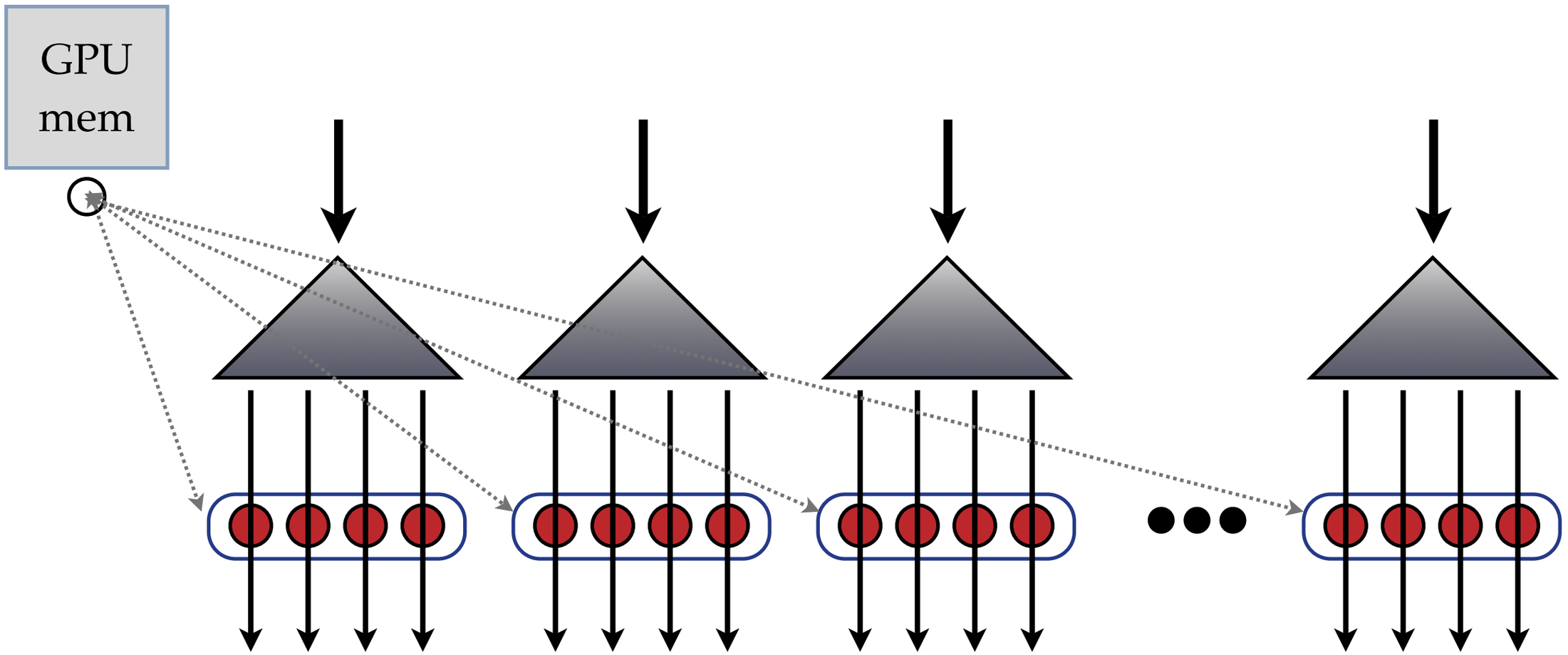
Metric $d(\cdot, \cdot)$

Goal: return x_i minimizing $d(q, x_i)$

$(\forall q \in Q)$

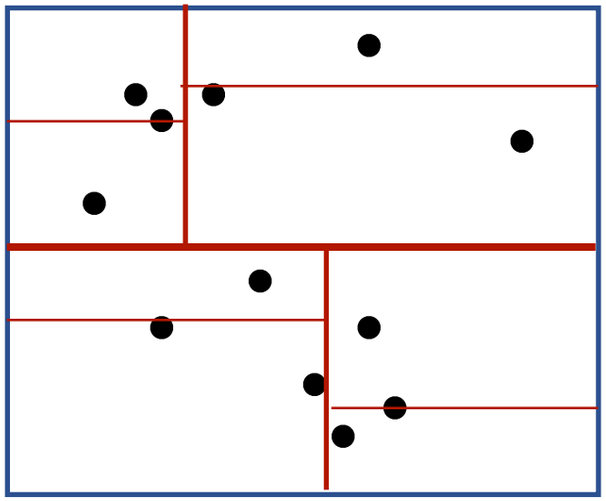
Hardware setting

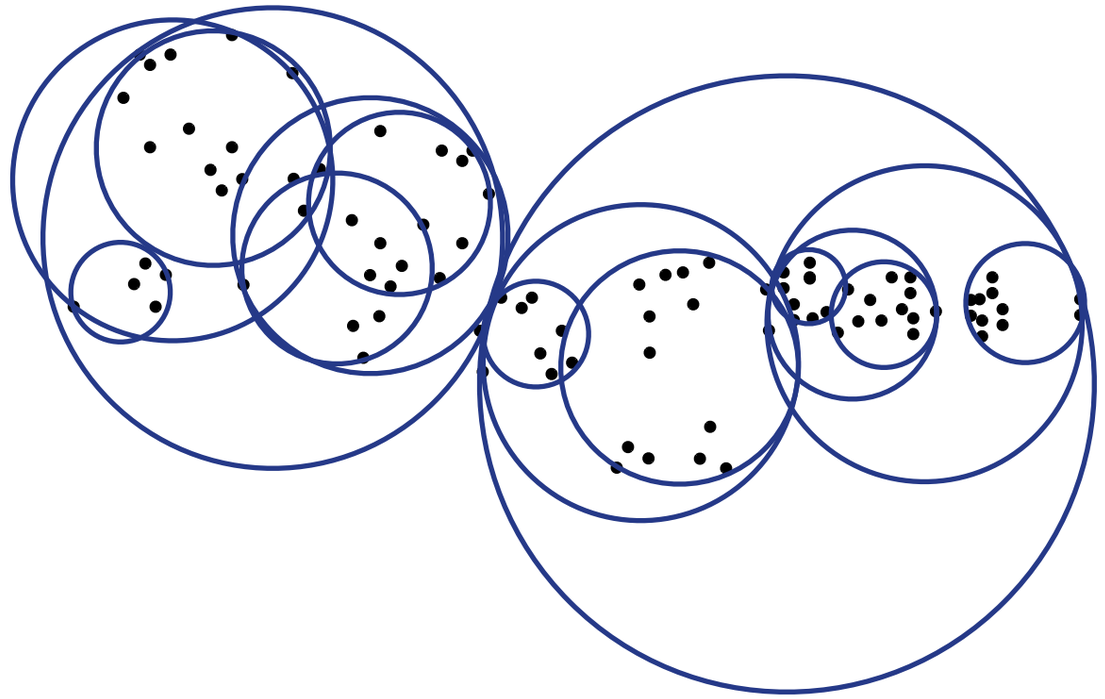
Massive parallelism; limited memory; limited communication



Efficient NN search: classic approach

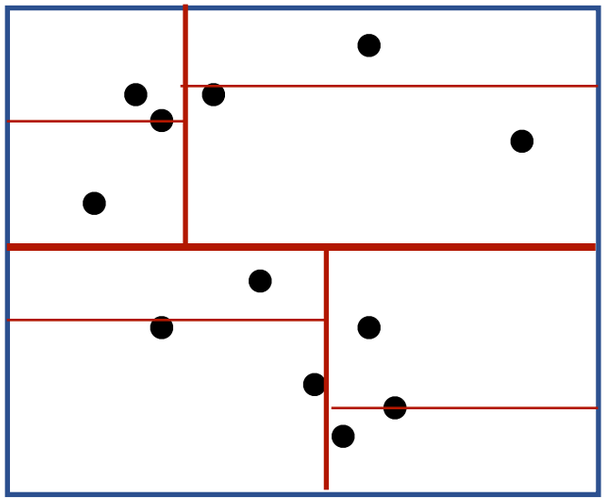
Decompose space; hopefully will only
have to look at a small part

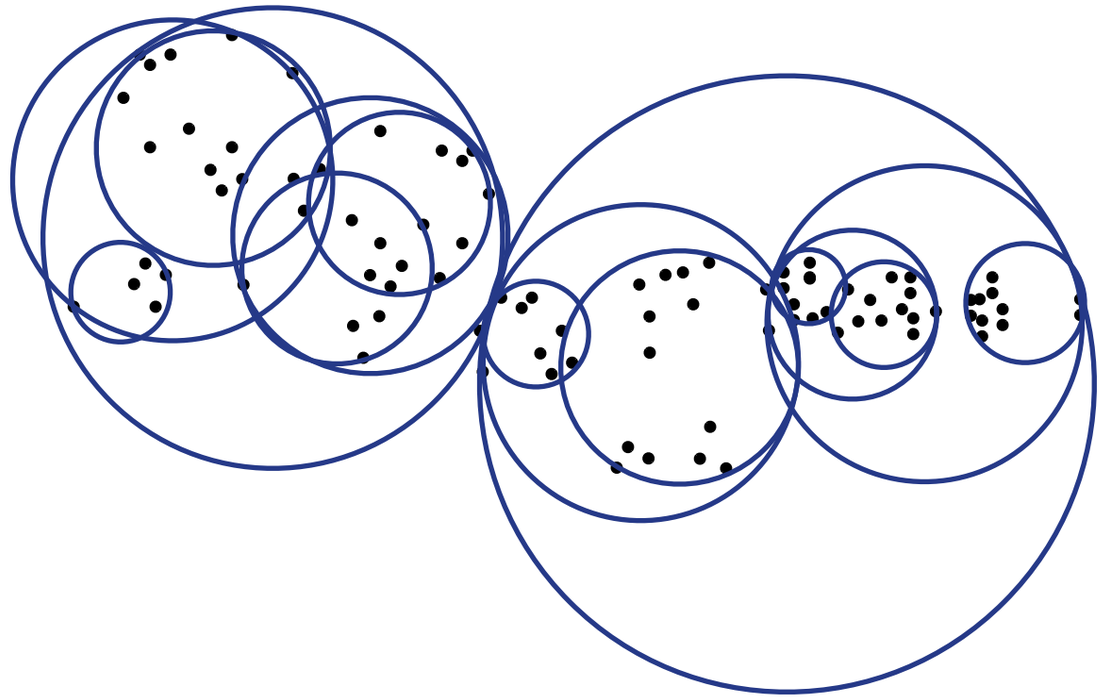




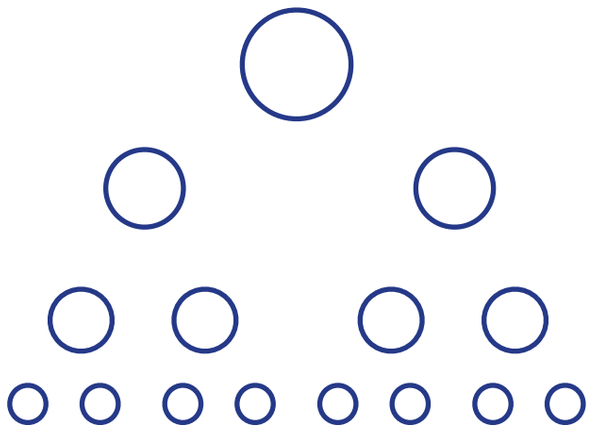
Efficient NN search: classic approach

Decompose space; hopefully will only have to look at a small part



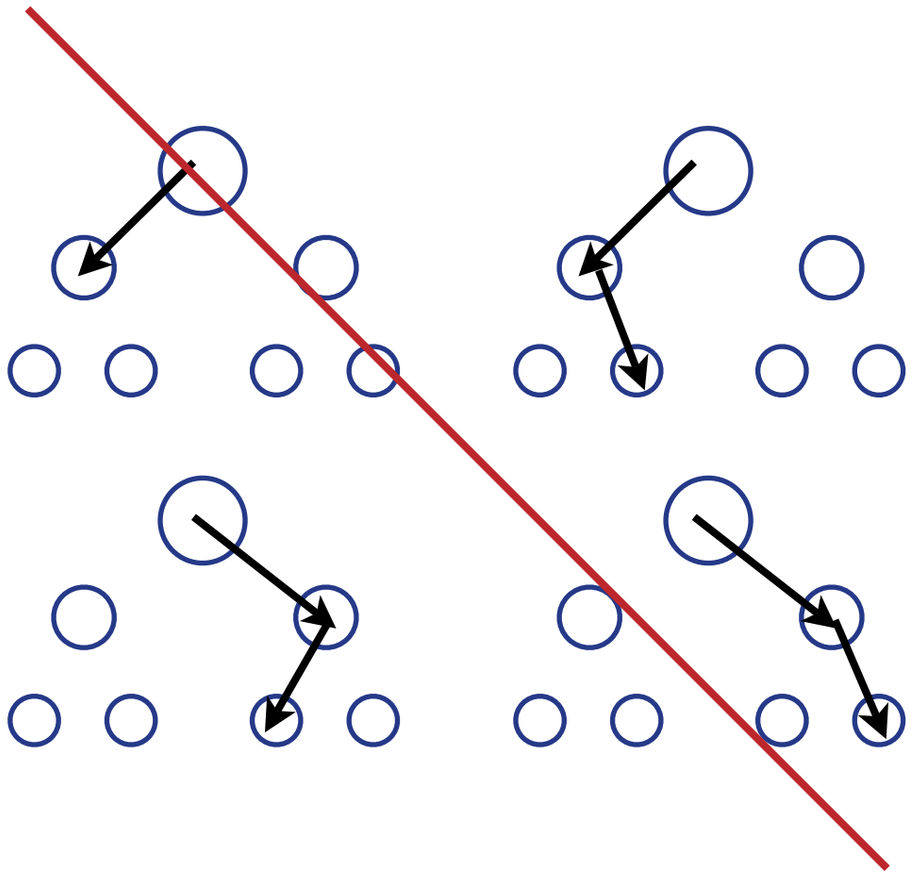


Organize cells into a tree:



Explore using branch-and-bound approach

Challenges for parallelism

1. 

Complex conditional computation seems difficult to distribute

2. Memory issues, practical and theoretical.

3. Needs to run in data-independent way.

What does work in parallel?

Matrix-matrix multiply

huge amount of work to do,
mostly independent.

What does work in parallel?

Matrix-matrix multiply

huge amount of work to do,
mostly independent.

Brute-force NN search

basically a matrix-matrix
multiply.

Brute force NN search

dataset	dim	CPU (s)	GPU (s)	Speedup
Bio	74	926.78	9.98	93
Physics	78	486.68	4.99	97

State-of-the-art data struct: 5-20x / 30-100x

[Beygelzimer *et al.*, 2006, Ram *et al.*, 2009]

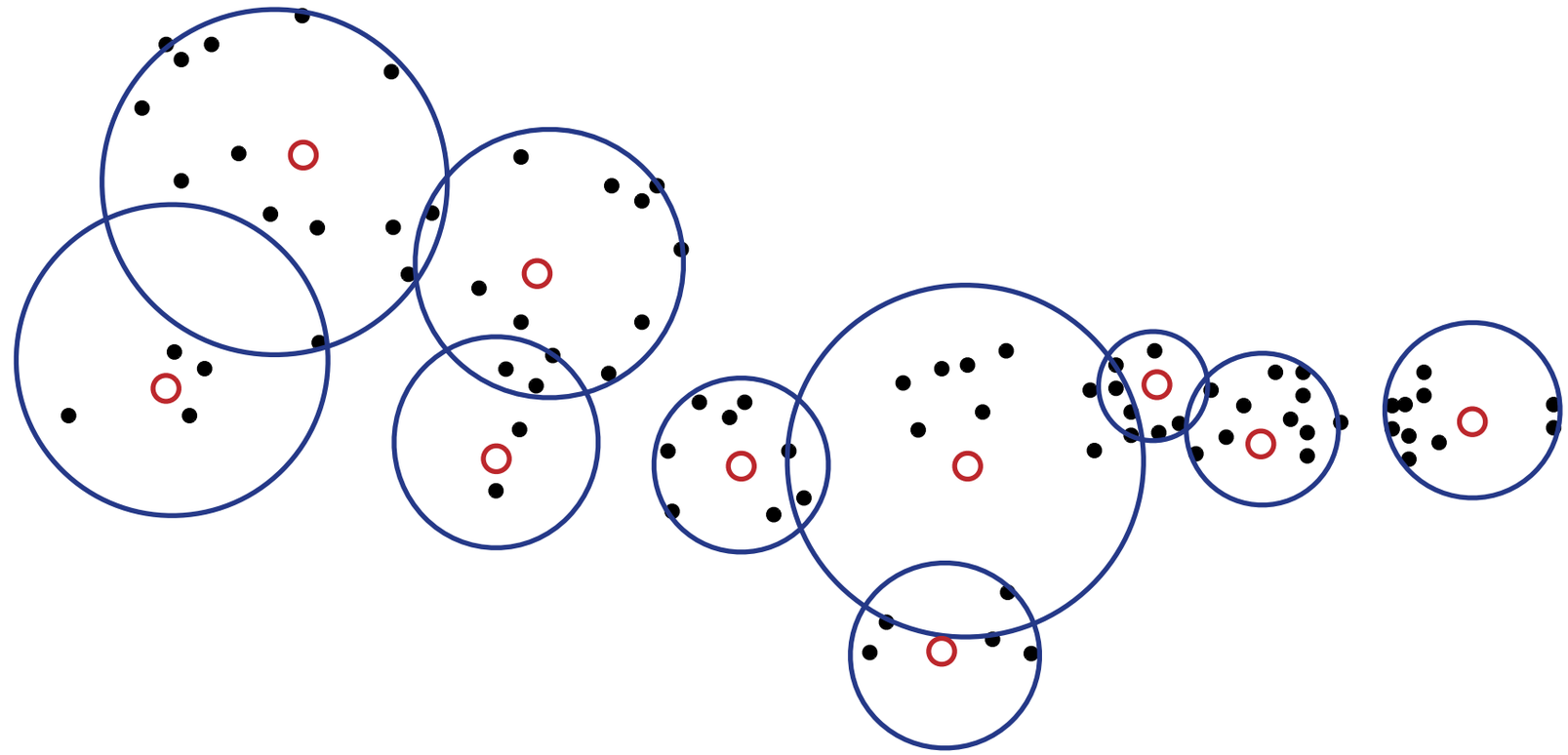
see also [Garcia *et al.*, 2008]

Goal

Build a data structure that provides a speedup
over GPU brute force

similar to the speedup given by metric trees
over CPU brute force.

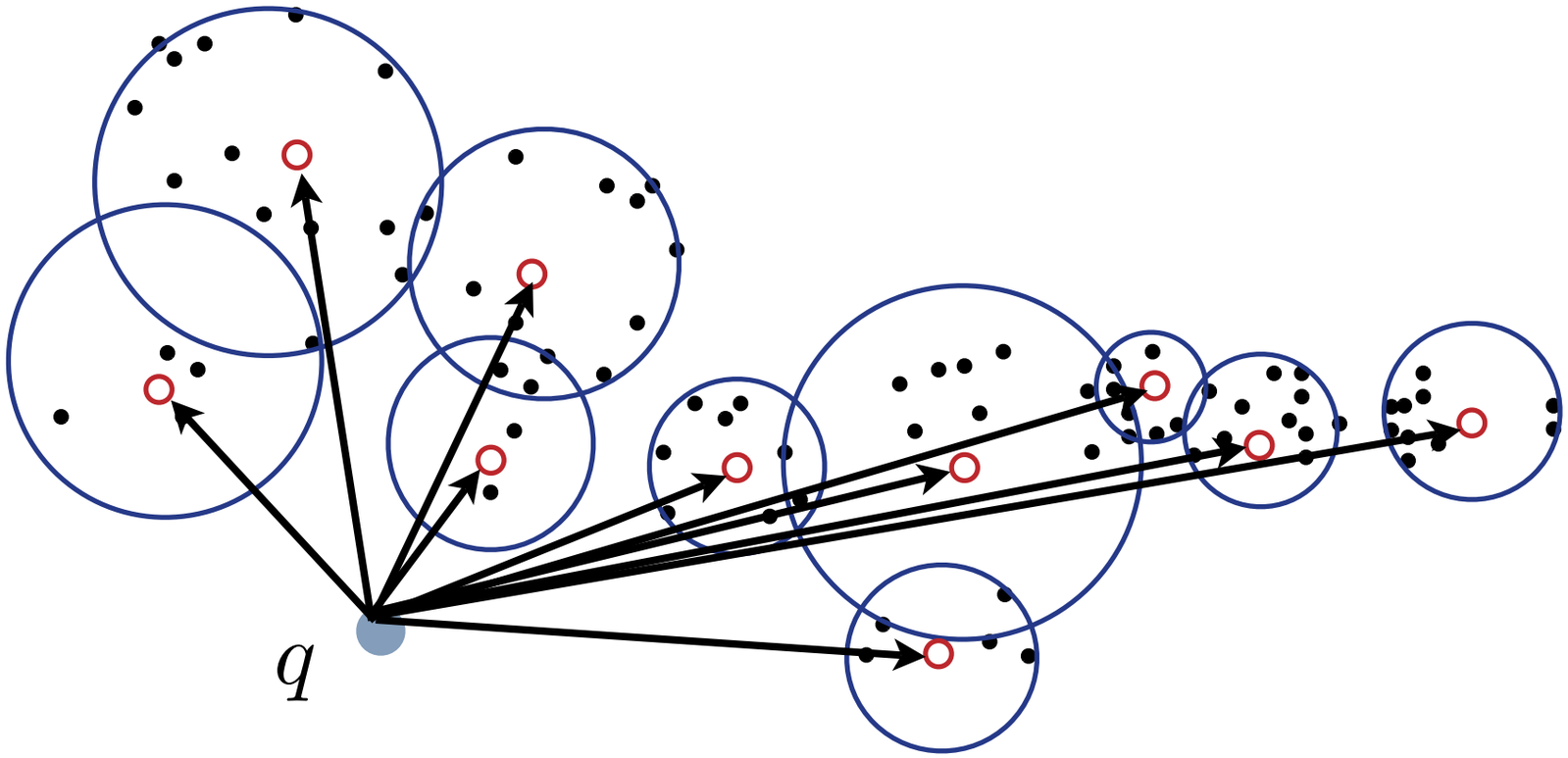
Random ball cover



r random representatives

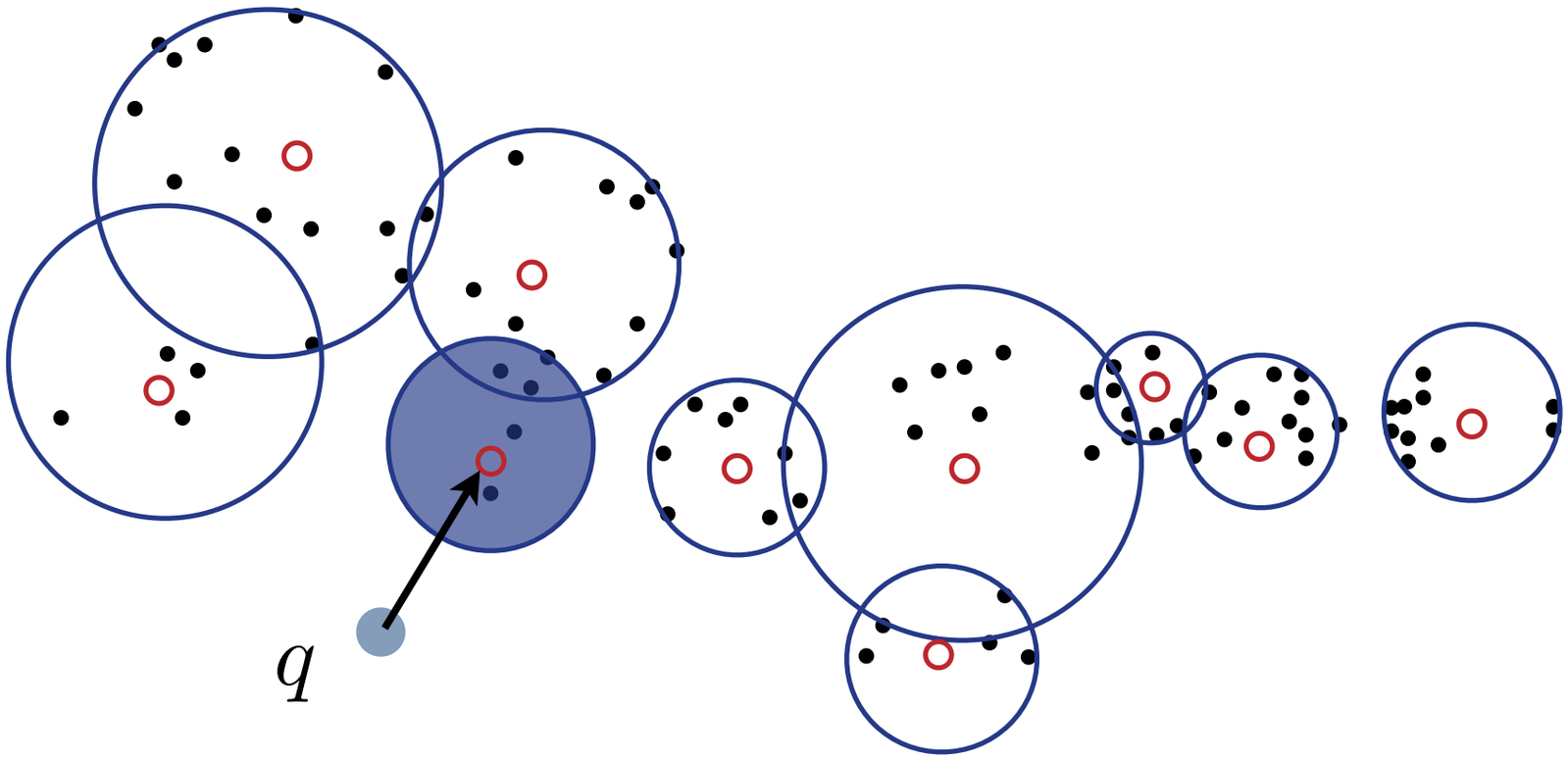
ball around representatives containing s points

RBC search algorithm



1. compute nearest representative

RBC search algorithm cont



2. find nearest point within set covered by nearest representative

Algorithm summary

For m queries, algorithm is two brute-force searches:

1. One for the representatives of size $m \cdot r$.
2. Another for the covered points of size $m \cdot s$.

Still fully utilizes parallel architecture, but requires far less work than brute force.

Parameters & Theory

$r =$ # of reps

$s =$ # of points assigned to each rep

Pick $s = r = O(\sqrt{n \log n})$

Yields $O(\sqrt{n \log n})$ query time (work)

(vs $O(n)$ for brute force)

Major work reduction; still parallelizable.

Parameters & Theory

Yields $O(\sqrt{n}\log n)$ query time (work)
(vs $O(n)$ for brute force)

Can prove low probability of error under standard notion of intrinsic dimensionality.

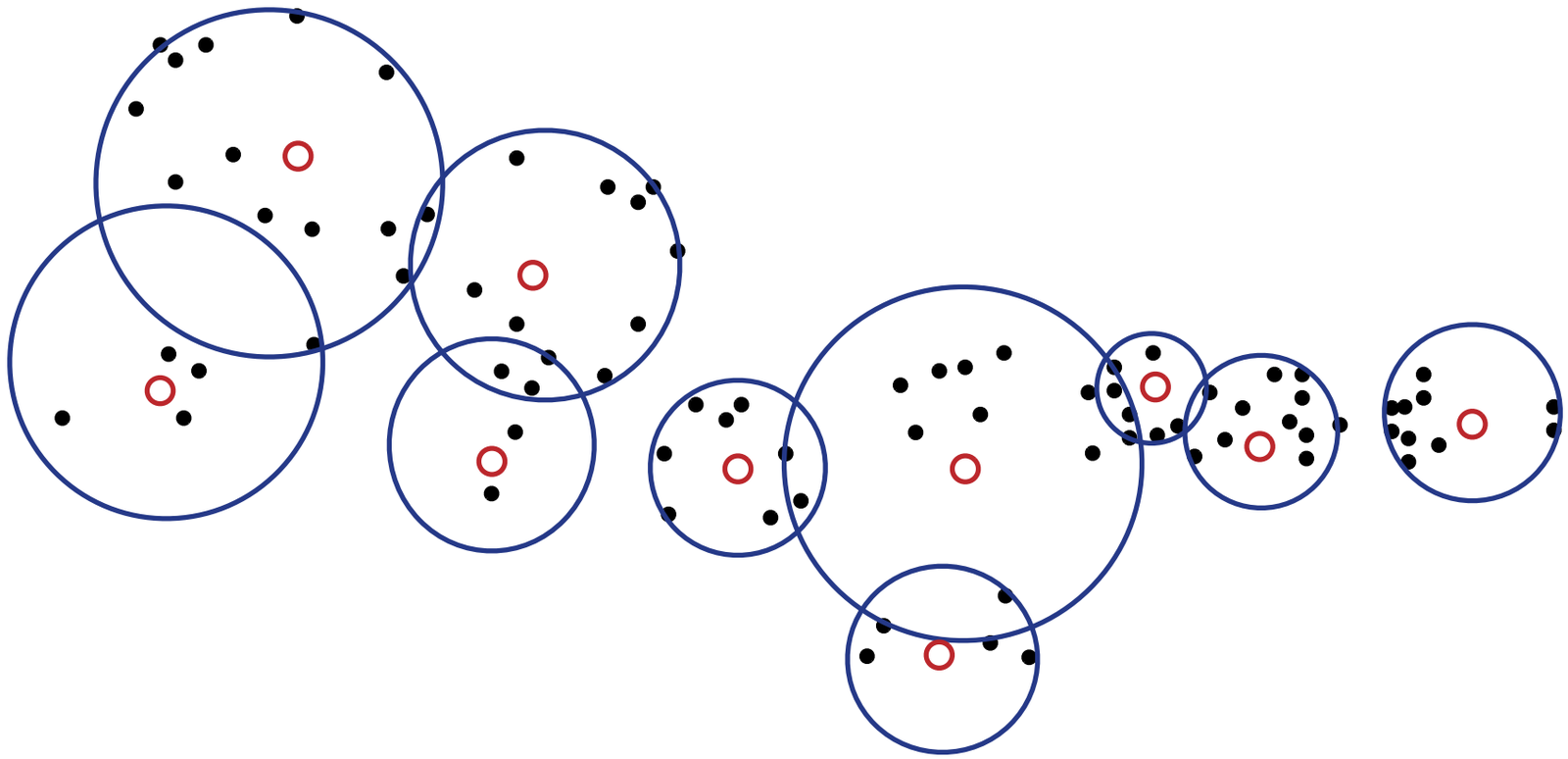
Intuition:

Each point belongs to $\log n$ reps on avg.

Overlap boosts probability of success.

Building the RBC

1. Select representatives at random



2. For each representative, generate list of sites owned L_r

Building on the GPU

For each rep, could compute all distances, then sort the list to get the top s...

..but the sorting time quickly dominates the computation time as s grows; here s is quite large ($\sqrt{n} \log n$)

Why?

- Irregular memory accesses (or work-inefficiency)
- GPU sorting is still an ongoing focus of research

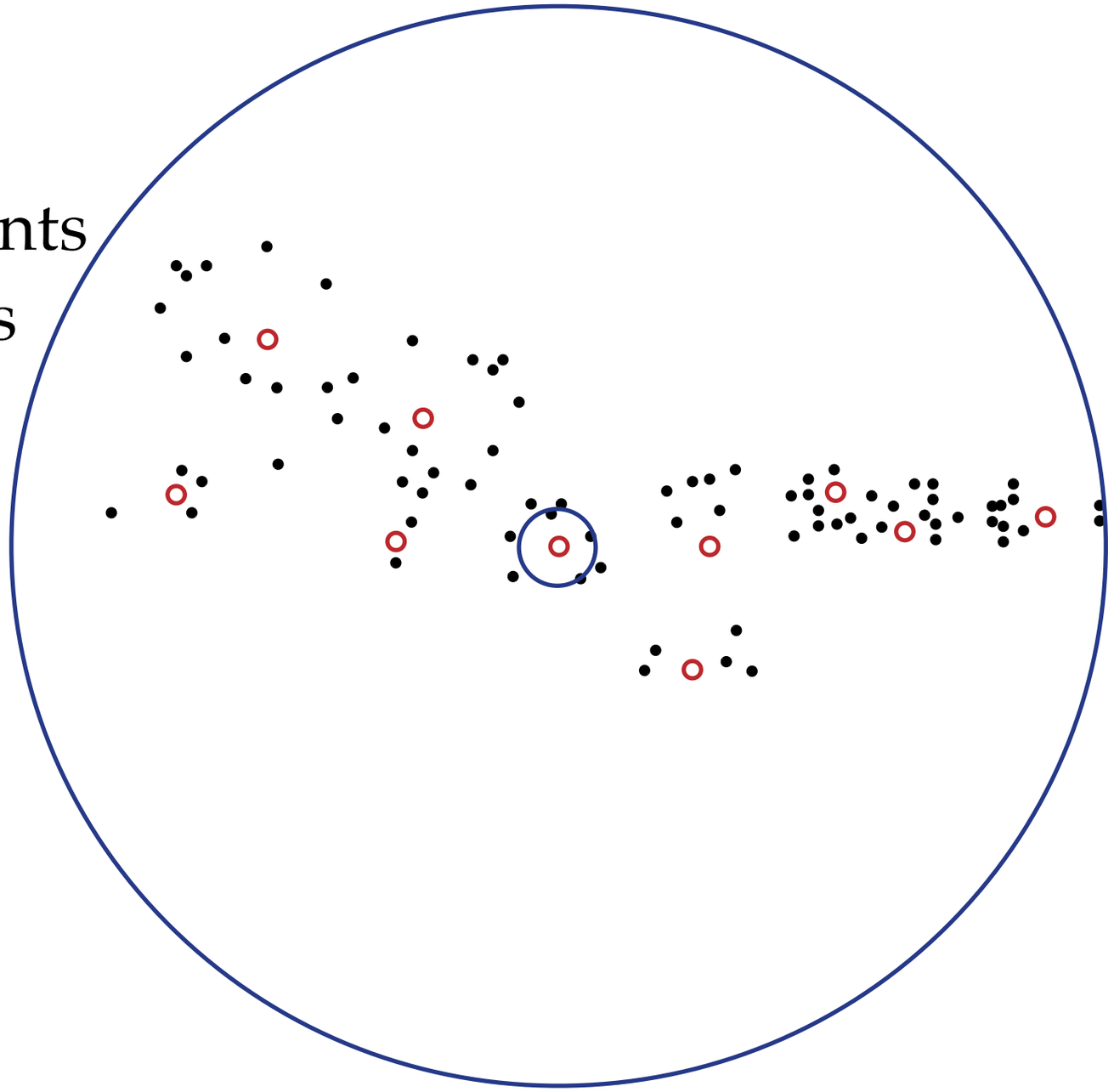
Building on the GPU

Want: build algorithm composed of simple, naturally parallel operations.

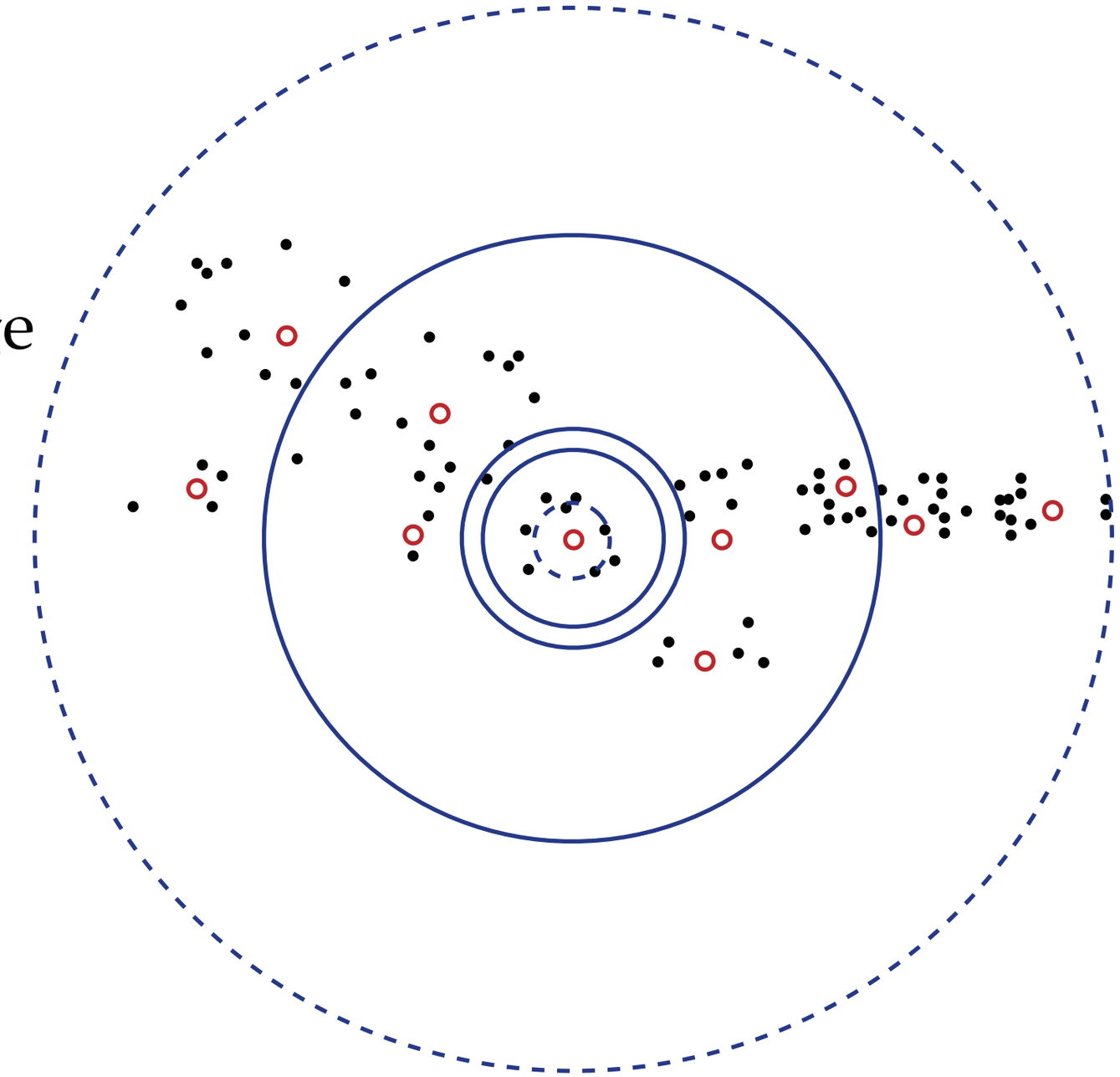
Idea: If we knew the range γ such that s points are within distance γ of the rep, we wouldn't need to sort.

.. so perform a sequence of brute force searches to find γ.

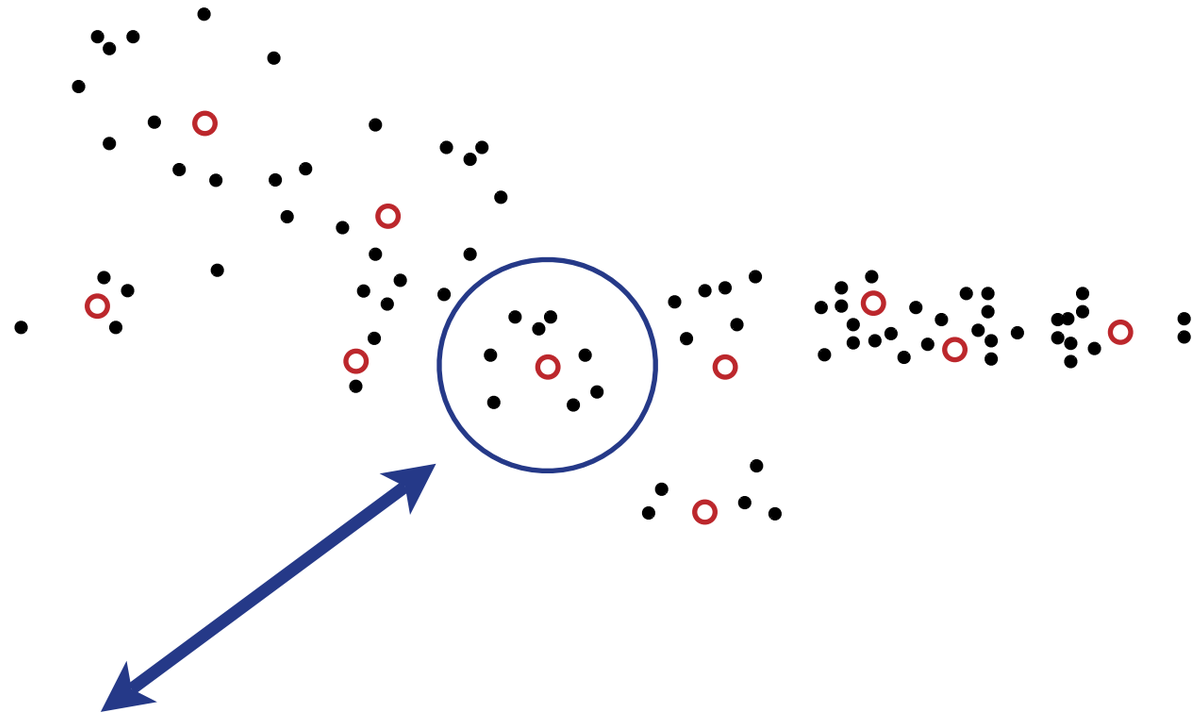
Find closest and farthest points via brute force; gives bounds on correct radius

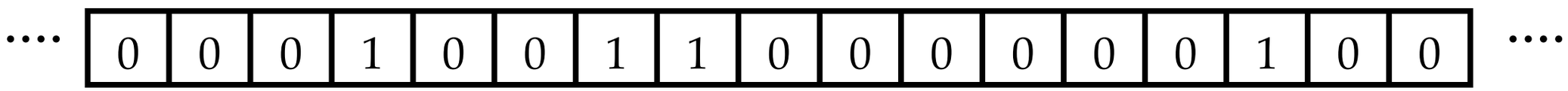


Perform succession of range counts to find correct radius



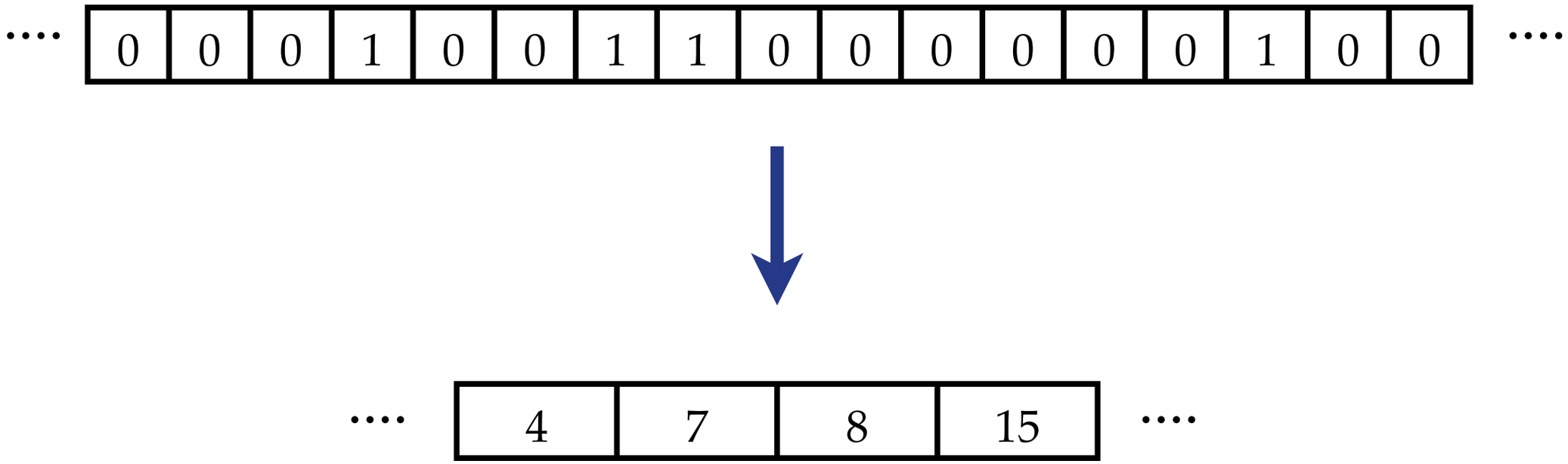
With correct radius found, peform range *search* to set binary indicator matrix





Finally

Perform parallel scan on bit arrays to produce mapping



Why bother?

All operations are naturally parallel and highly efficient on GPU:

- Brute force searches (essentially matrix-matrix)
- Parallel scan

Experiments: data

dataset	dim	size	# queries
Bio	74	200k	50k
Robot	21	1M	1M
Phy	78	100k	50k

Experiments: search time

dataset	Brute (s)	RBC (s)	Speedup	Rank
Bio	9.97	0.20	49	0.74
Robot	408.23	3.35	122	0.71
Phy	4.99	0.14	35	1.34

Experiments: total time

dataset	Brute (s)	RBC (s)
Bio	9.97	1.28
Robot	408.23	11.92
Phy	4.99	0.65

Code available for download.