

Clustering with *k*-means: faster, smarter, cheaper

Charles Elkan

University of California, San Diego

April 24, 2004

Clustering is difficult!

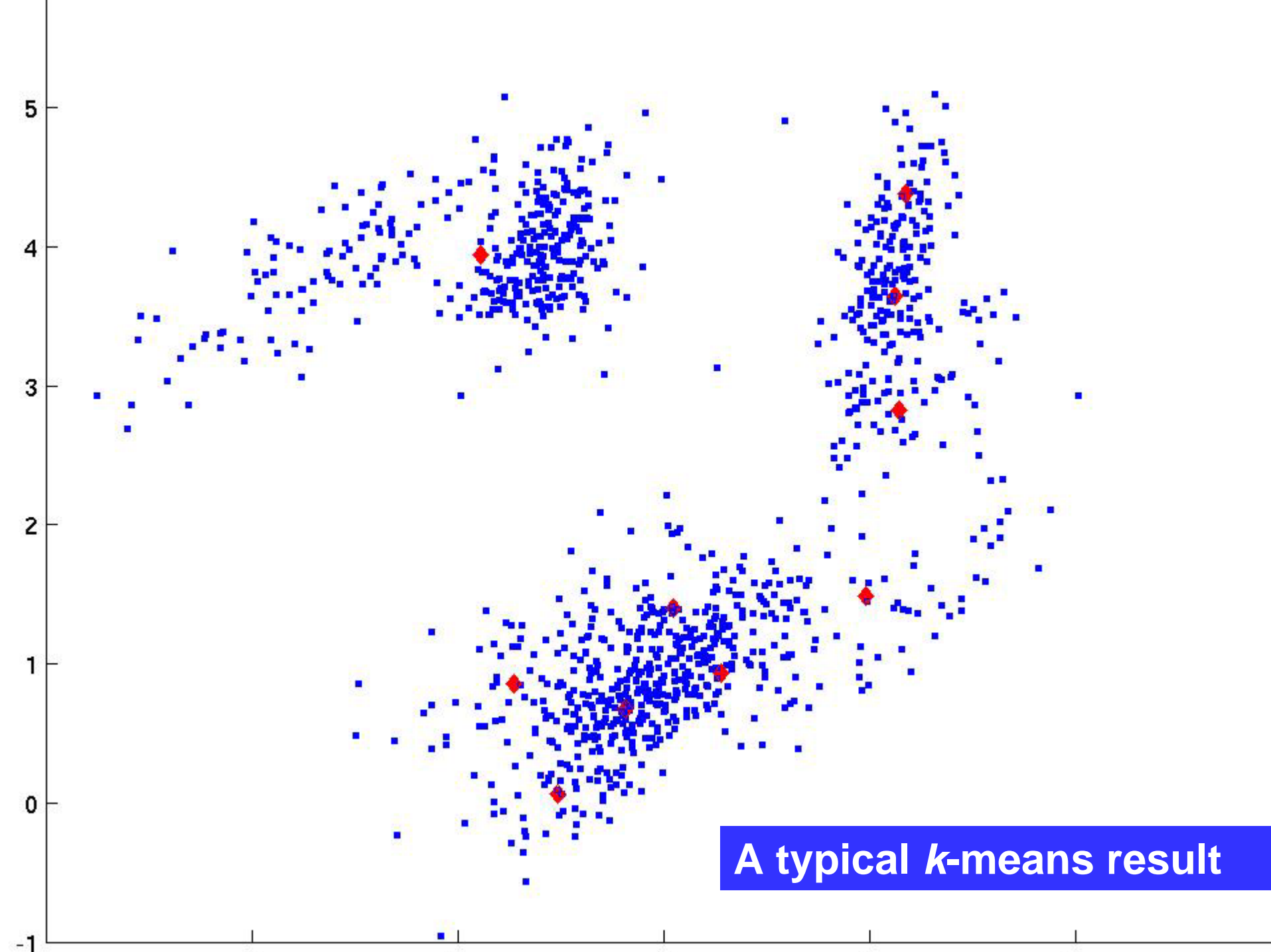


Source: Patrick de Smet, University of Ghent

The standard k -means algorithm

Input: n points, distance function $d()$,
number k of clusters to find.

- | | STEP NAME |
|--|------------|
| 1. Start with k centers | |
| 2. Compute d (each point x , each center c) | |
| 3. For each x , find closest center $c(x)$ | "ALLOCATE" |
| 4. If no point has changed "owner" $c(x)$, stop | |
| 5. Each $c \leftarrow$ mean of points owned by it | "LOCATE" |
| 6. Repeat from 2 | |



A typical k -means result

Observations

Theorem: If $d()$ is Euclidean, then k -means converges monotonically to a local minimum of within-class squared distortion:

$$\sum_x d(c(x), x)^2$$

1. Many variants, complex history since 1956, over 100 papers per year currently
2. Iterative, related to expectation-maximization (EM)
3. # of iterations to converge grows slowly with n , k , d
4. No accepted method exists to discover k .

We want to ...

1. ... make the algorithm faster.
2. ... find lower-cost local minima.
(Finding the global optimum is NP-hard.)
3. ... choose the correct k intelligently.

With success at (1), we can try more alternatives for (2).
With success at (2), comparisons for different k are less likely to be misleading.

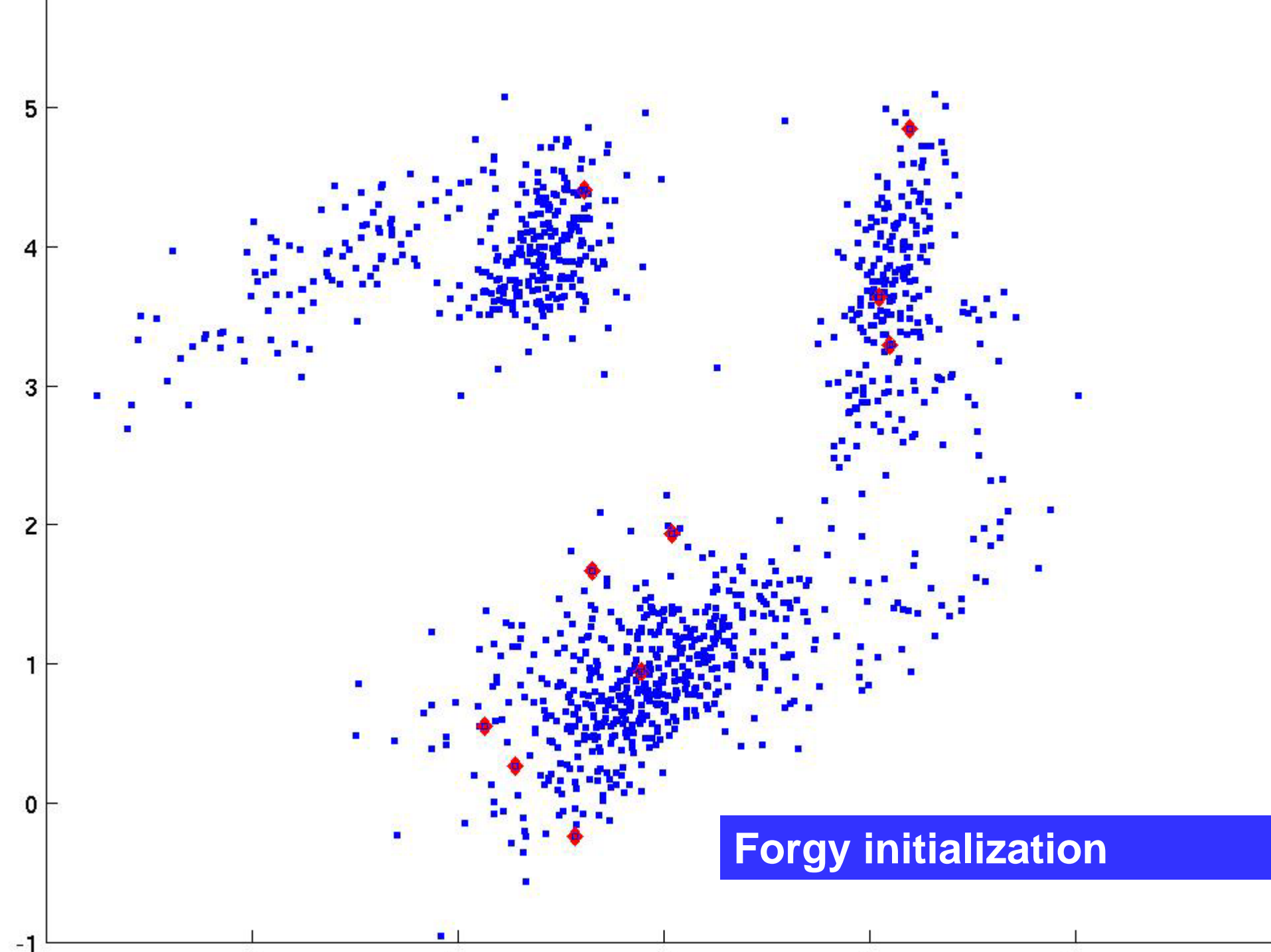
Standard initialization methods

Forgy initialization: choose k points at random as starting center locations.

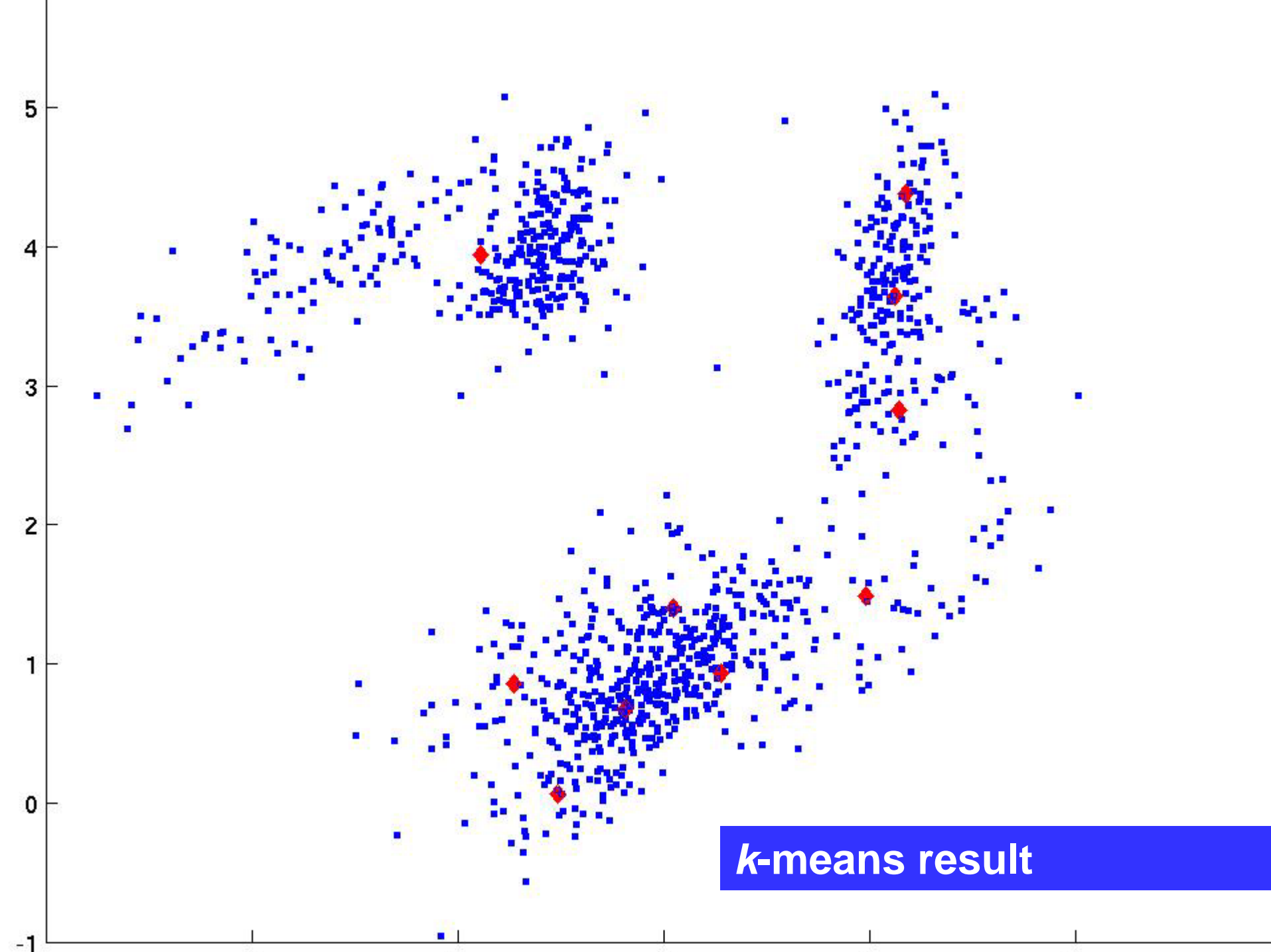
Random partitions: divide the data points randomly into k subsets.

Both these methods are bad.

E. Forgy. *Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications*. Biometrics, 21(3):768, 1965.



Forgy initialization

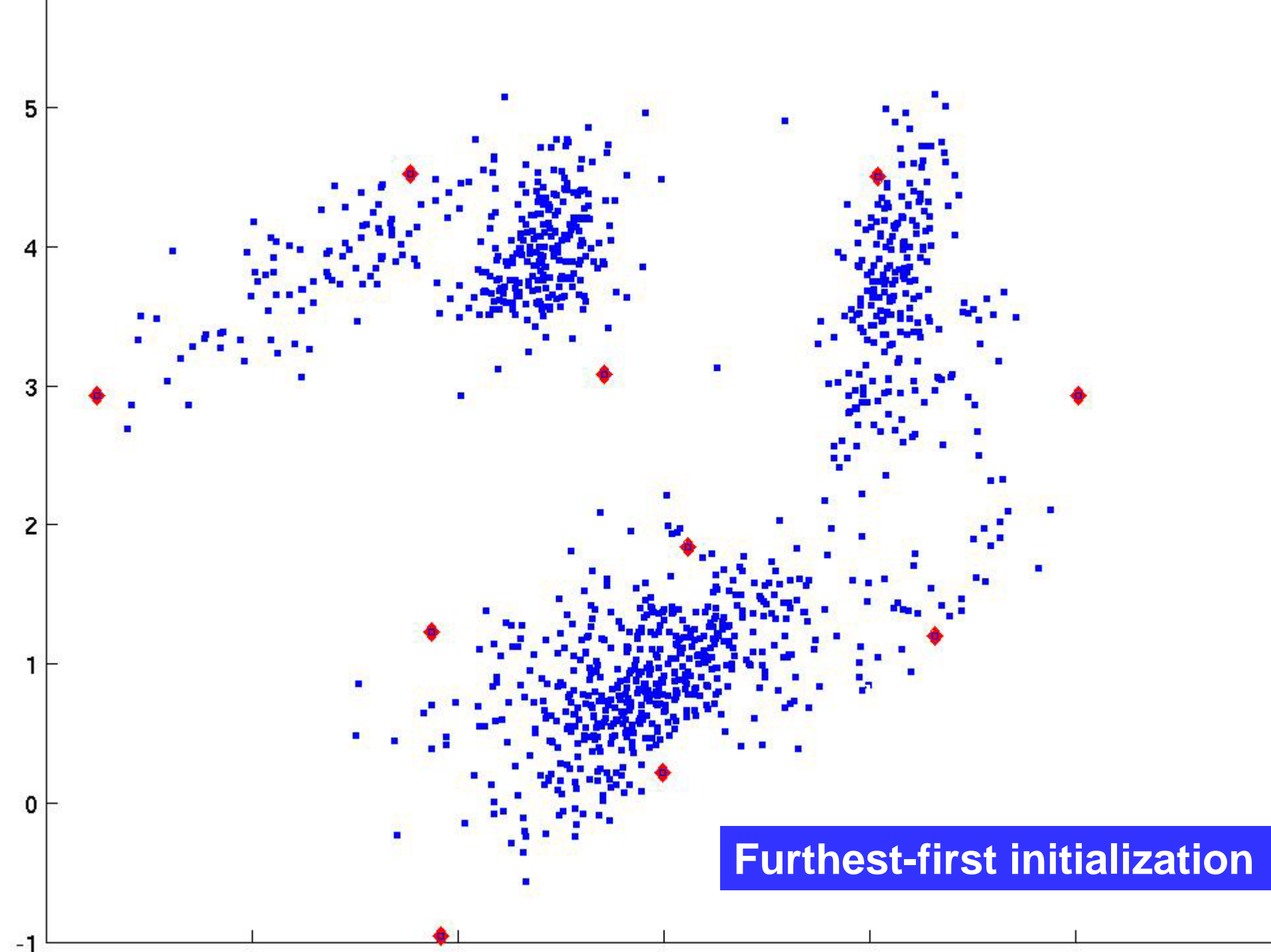


Smarter initialization

The “furthest first” algorithm (FF):

1. Pick first center randomly.
2. Next is the point furthest from the first center.
3. Third is the point furthest from both previous centers.
4. In general: next center is $\operatorname{argmax}_x \min_c d(x,c)$

D. Hochbaum, D. Shmoys. *A best possible heuristic for the k-center problem*, Mathematics of Operations Research, 10(2):180-184, 1985.



Subset furthest-first (SFF)

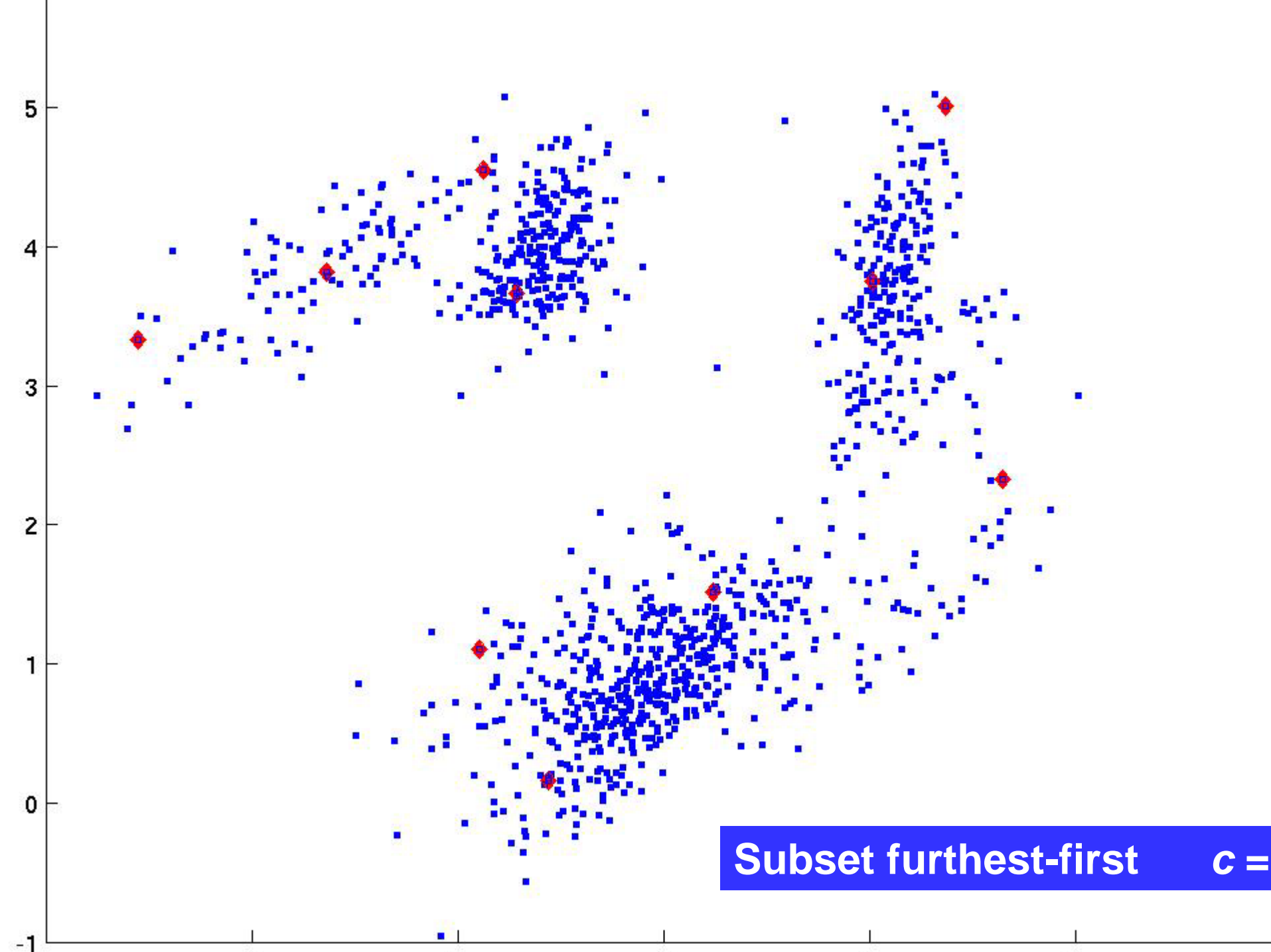
FF finds outliers, by definition not good cluster centers!

Can we choose points far apart and typical of the dataset?

Idea: A random sample includes many representative points, but few outliers.

But: How big should the random sample be?

Lemma: Given k equal-size sets and $c > 1$, with high probability $ck \log k$ random points intersect each set.



Comparing initialization methods

<i>method</i>	<i>mean</i>	<i>std. dev.</i>	<i>best</i>	<i>worst</i>
Forgy	218	193	29	2201
Furthest-first	247	59	139	426
Subset furthest-first	83	30	20	214

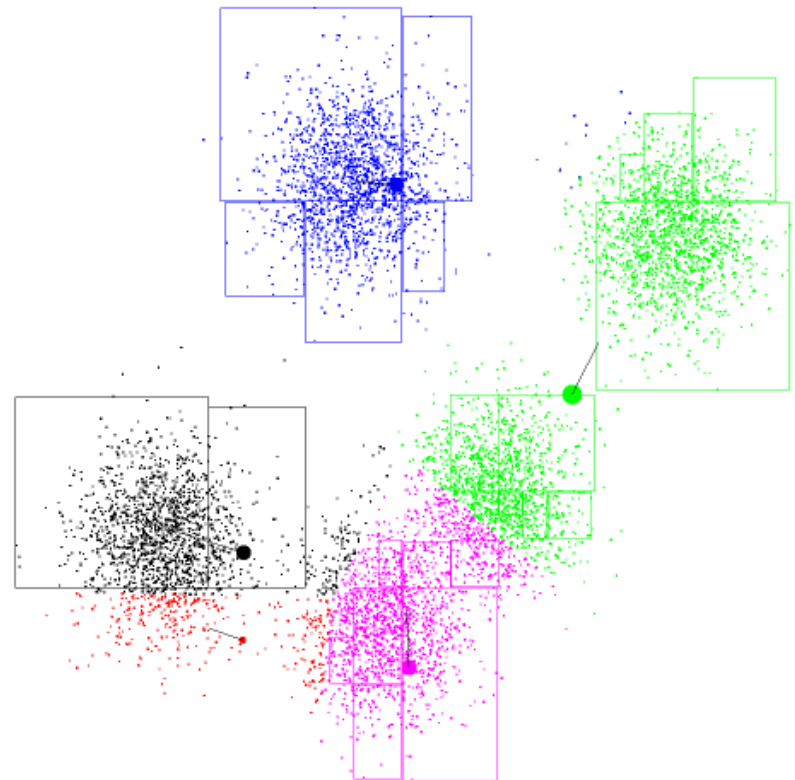
218 means 218% worse than the best clustering known.

Lower is better.

Goal: Make k -means faster, but with same answer

Allow any black-box $d()$,
any initialization method.

1. In later iterations, little movement of centers.
2. Distance calculations use the most time.
3. Geometrically, these are mostly redundant.



Lemma 1: Let x be a point, and let b and c be centers.

If $d(b,c) \geq 2d(x,b)$ then $d(x,c) \geq d(x,b)$.

Proof: We know $d(b,c) \leq d(b,x) + d(x,c)$.

So $d(b,c) - d(x,b) \leq d(x,c)$.

Now $d(b,c) - d(x,b) \geq 2d(x,b) - d(x,b) = d(x,b)$.

So $d(x,b) \leq d(x,c)$.

• c



<i>name</i>	<i>cardinality</i>	<i>dimensionality</i>	<i>description</i>
covtype	150000	54	Remote soil cover measurements
random	10000	1000	Uniform random data
kdd98	95413	56	KDD Cup 1998 data, normalized

		<i>k=3</i>	<i>k=20</i>	<i>k=100</i>
covtype	iterations	18	256	152
	standard	8,100,000	768,000,000	2,280,000,000
	fast	941,560	7,147,200	7,353,400
	speedup	8.60	107	310
	Moore speedup	24.8	11.3	19.0
random	iterations	52	3	18
	standard	1,560,000	6,600,000	18,000,000
	fast	1,039,600	3,019,700	5,341,000
	speedup	1.50	2.19	3.37
kdd98	iterations	186	89	125
	standard	53,240,000	169,800,000	1,192,700,000
	fast	2,395,000	8,951,000	29,162,000
	speedup	22.2	19.0	40.9

Beyond the triangle inequality

Geometrically, the triangle inequality is a coarse screen

Consider the 2-d case with all points on a line:

data point at $(-4,0)$

center1 at $x=(0,0)$

center2 at $x=(4,0)$

Triangle inequality ineffective, even though our safety margin is huge

Remembering the margins

Idea: when the triangle inequality fails, cache the margin and refer to it in subsequent iterations

Benefit: a further 1.5X to 30X reduction over Elkan's already excellent result

Conclusion: if distance calculations weren't much of a problem before, they really aren't a problem now

So what is the new bottleneck?

Memory bandwidth is the current bottleneck

Main loop reduces to:

- 1) fetch previous margin
- 2) update per most recent centroid movement
- 3) compare against current best and swap if needed

Most compares are favorable (no change results)

Memory bandwidth requirement is one read and one write per cell in an $N \times K$ margin array

Reducible to one read if we store margins as deltas

Going parallel – data partitioning

Use a PRNG distribution of the points

Avoids hotspots “statically”

Besides N more compare loops running, we may also get super-scalar benefit if our problem moves from main memory to L2 or from L2 to L1

Misdirections – trying to exploit the sparsity



Full sorting

Waterfall priority queues

