universität wien

Conceptual Modeling and Programming in GIScience

Lecture 9: Ripley’s K Worked Example

Yingjing Huang

yingjing.huang@univie.ac.at

2026S – June 15, 2026

What You Will Learn in This Lecture

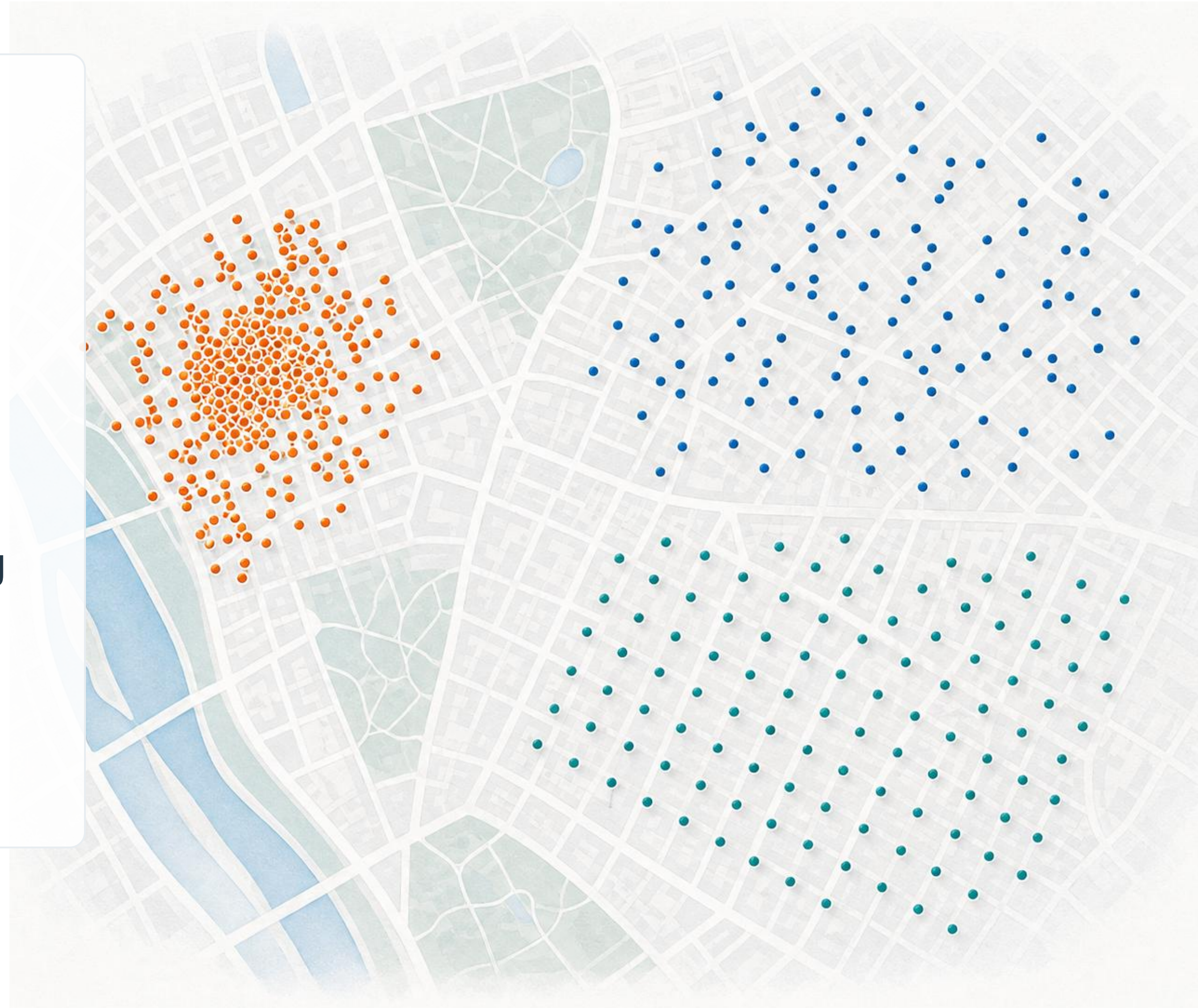
- Recognise clustered, random, and dispersed point patterns
- Explain Ripley's K as neighbour counting across distances
- Compare an observed pattern with CSR simulations
- Read a Monte Carlo envelope and account for edge effects
- Translate the method into UML, Java classes, and algorithms

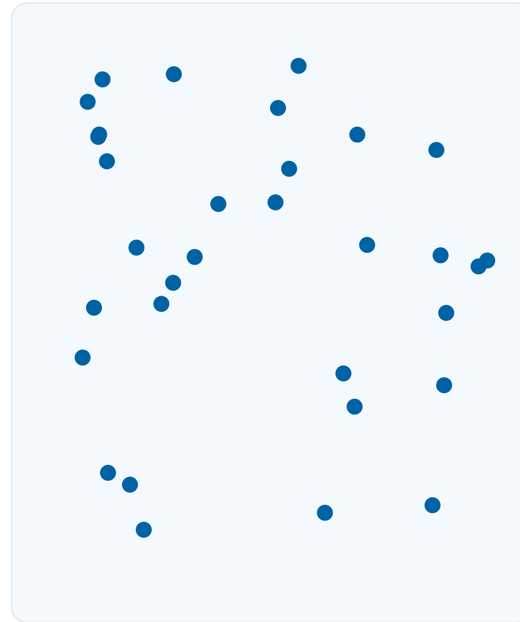
Point Pattern Analysis Starts with One Question

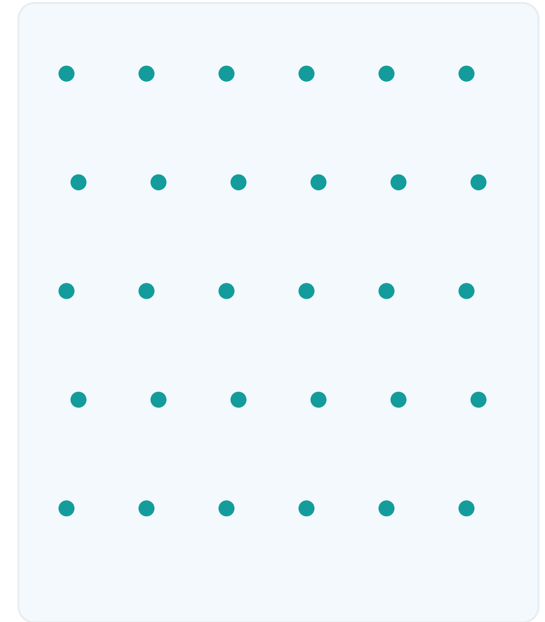
Imagine each point is…

a coffee shop

Are the shops gathering in busy areas, appearing without a clear pattern, or keeping their distance?







CLUSTERED

groups / hotspots

RANDOM

no obvious arrangement

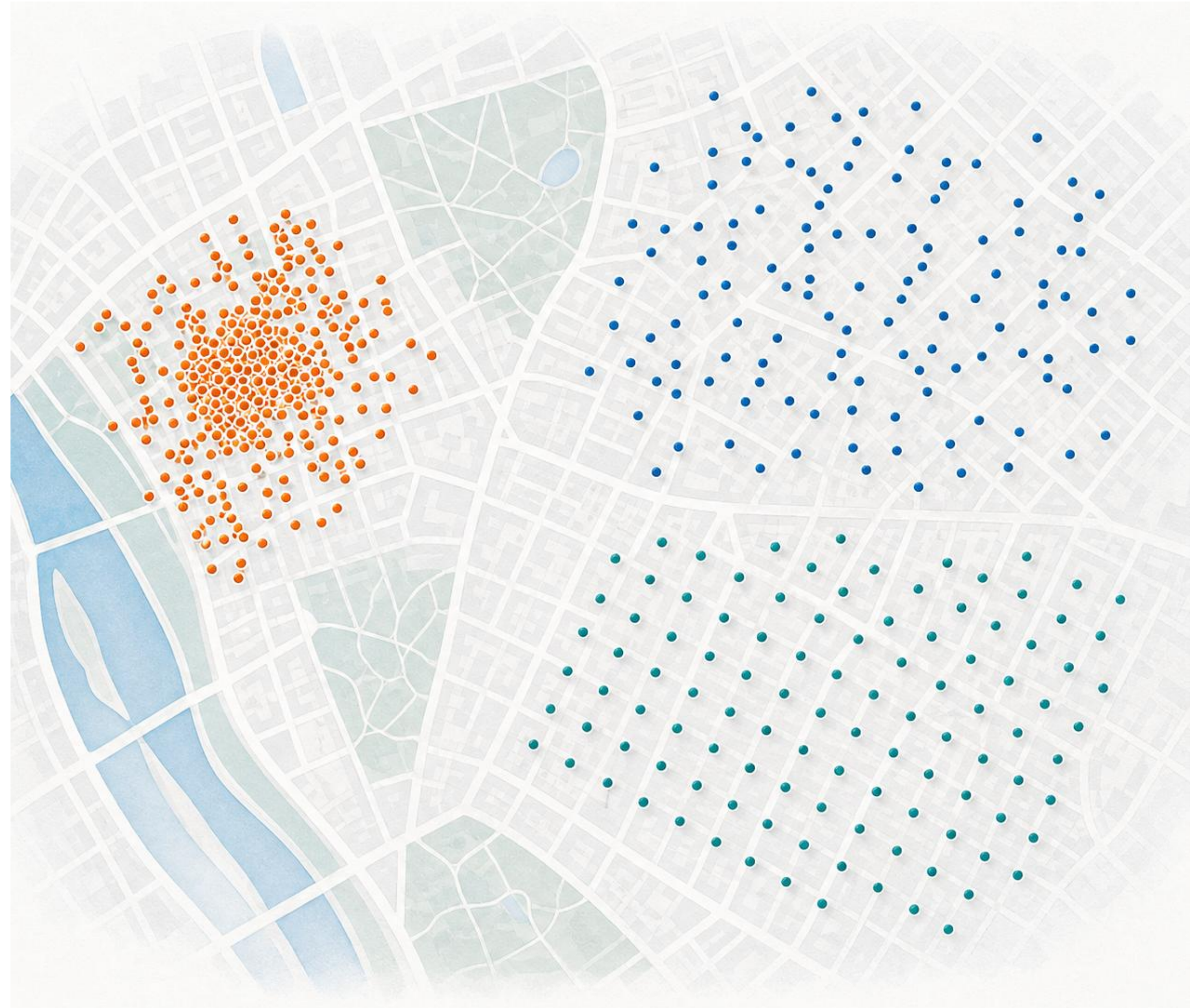
DISPERSED

points keep apart

Using numbers (e.g. Ripley's K) to accurately describe visual patterns

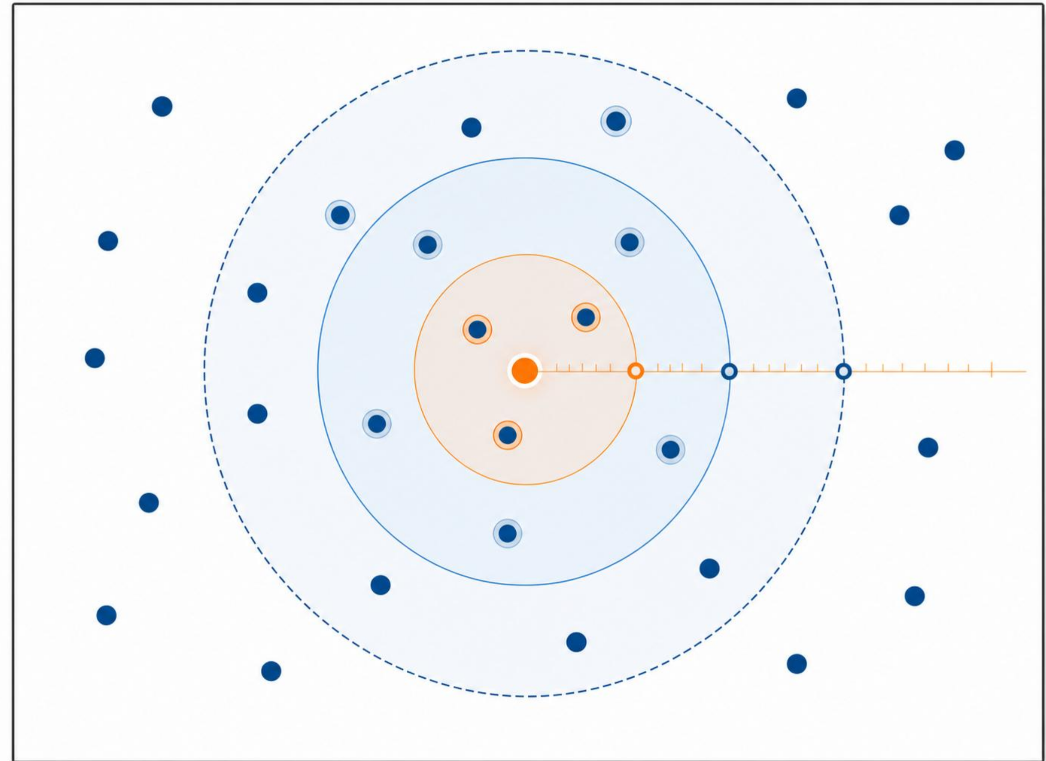
The Same Geometry Appears in Many Fields

- **TREES**
- **DISEASE**
- **SHOPS**
- **CRIME**
- **ACCIDENTS**

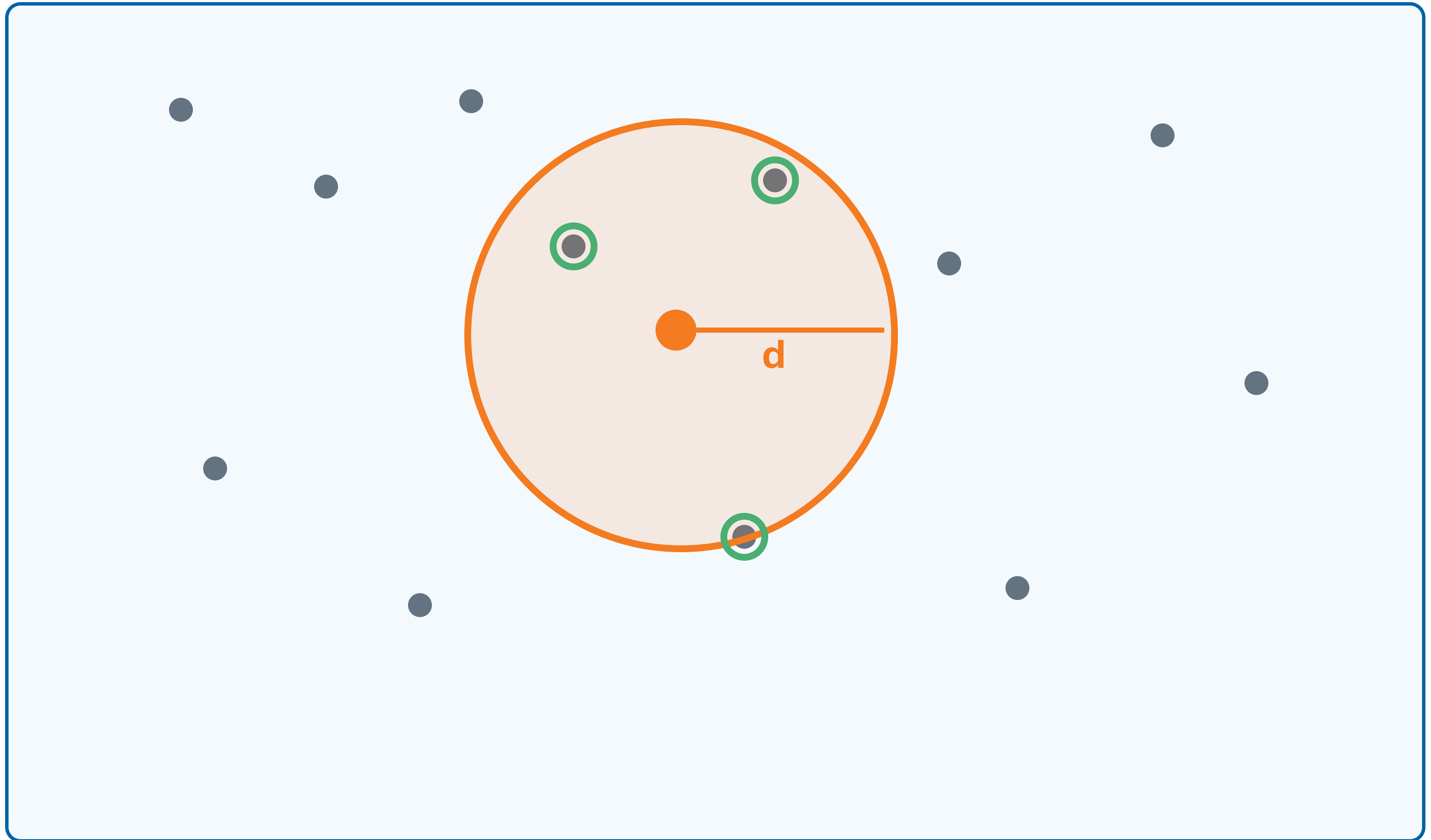


Ripley's K: Draw Circles and Count Neighbours

- Choose one point
- Draw a circle
- Count other points inside
- Repeat for every point

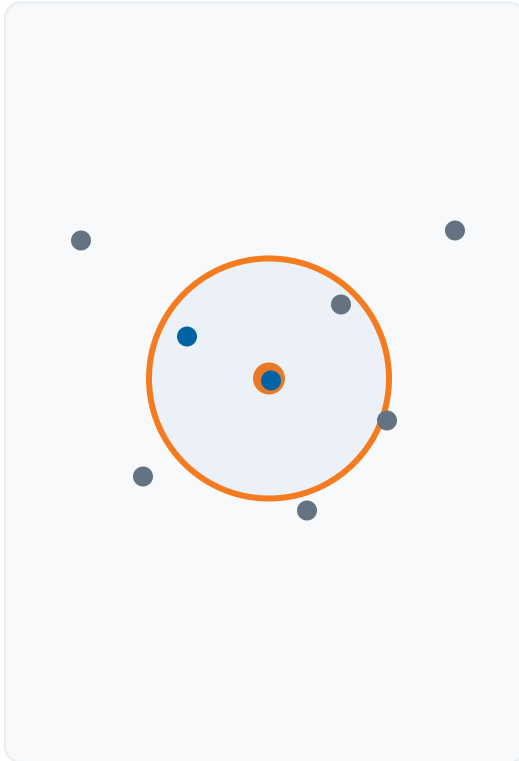


One Circle, One Yes-or-No Question

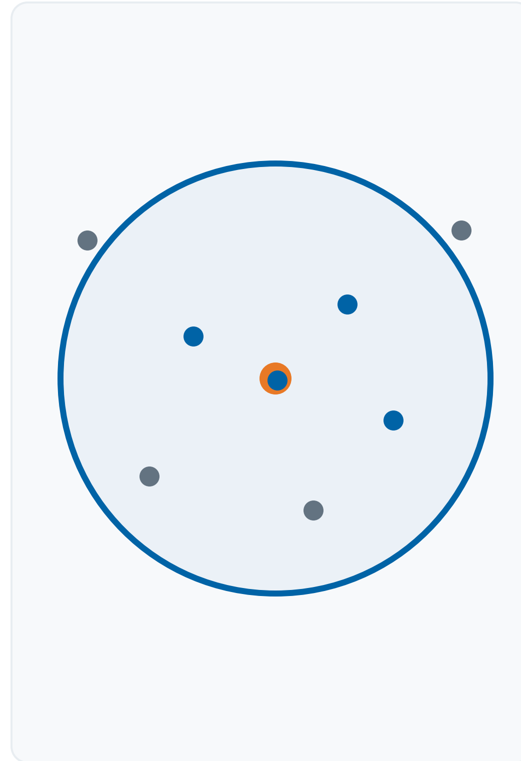


For each other point: distance ≤ d ?
Yes → count it.

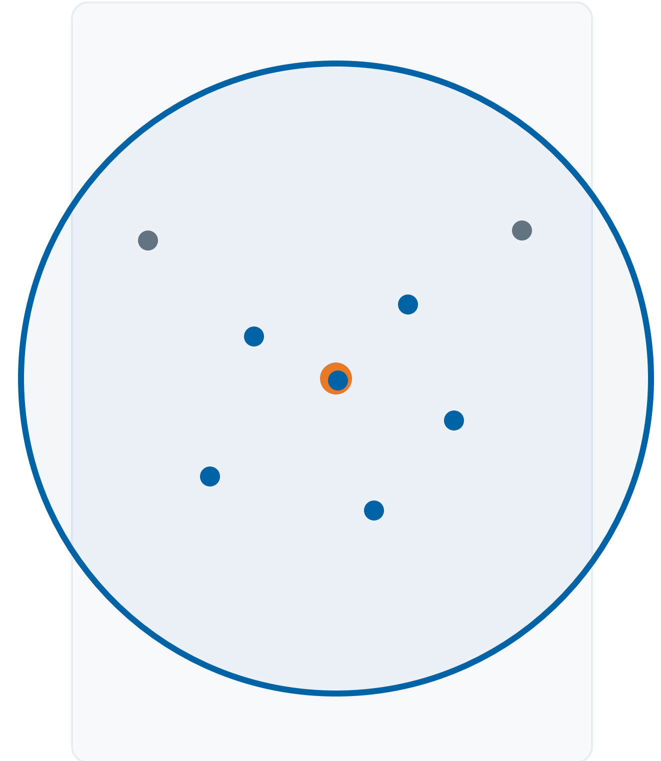
"Nearby" Changes with Scale



25 px
very close



50 px
local group



100 px
broader pattern

Ripley's K: Draw Circles and Count Neighbours

$$K(r) = \frac{A}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} w_{ij} I(d_{ij} \leq r)$$

Variable Breakdown:

- **K(r)**: The K-function value for a given radius or distance, r.
- **A**: The total area of the study region.
- **n**: The total number of points.
- **i** and **j**: Individual points in the dataset.
- d_{ij}: The straight-line distance between point i and point j.
- **I**: An indicator function that equals 1 if the distance d_{ij} is less than or equal to r, and 0 otherwise.
- w_{ij}: A boundary/edge correction weight. This adjusts the calculation so that points near the edge of the study area don't skew results (accounting for the fact that we can't observe points outside the boundary).

K(r) =

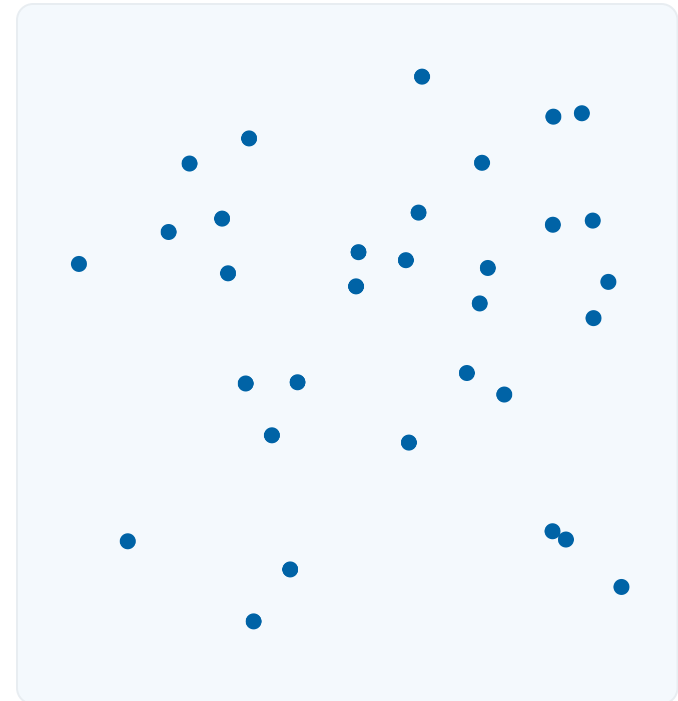
A / n(n−1)	Σ Σ	I(distance ≤ r)	× weight
make datasets comparable	check every ordered pair	count only neighbours	correct the boundary

In Java: loops + if + arithmetic

Always translate the notation into operations

universität wien

compare

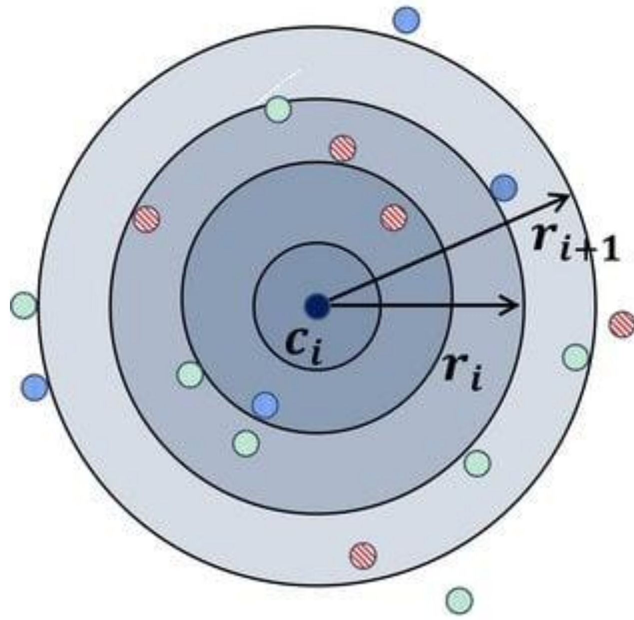


OBSERVED

CSR: SAME n, SAME AREA

complete spatial randomness (CSR)

Under CSR, expected $K(d) = \pi d^2$.



Ripley's K

$K(r)$

An observed K-function

K-function for a random process

Confidence interval for random process

above random

more neighbours

Clustered

Dispersed

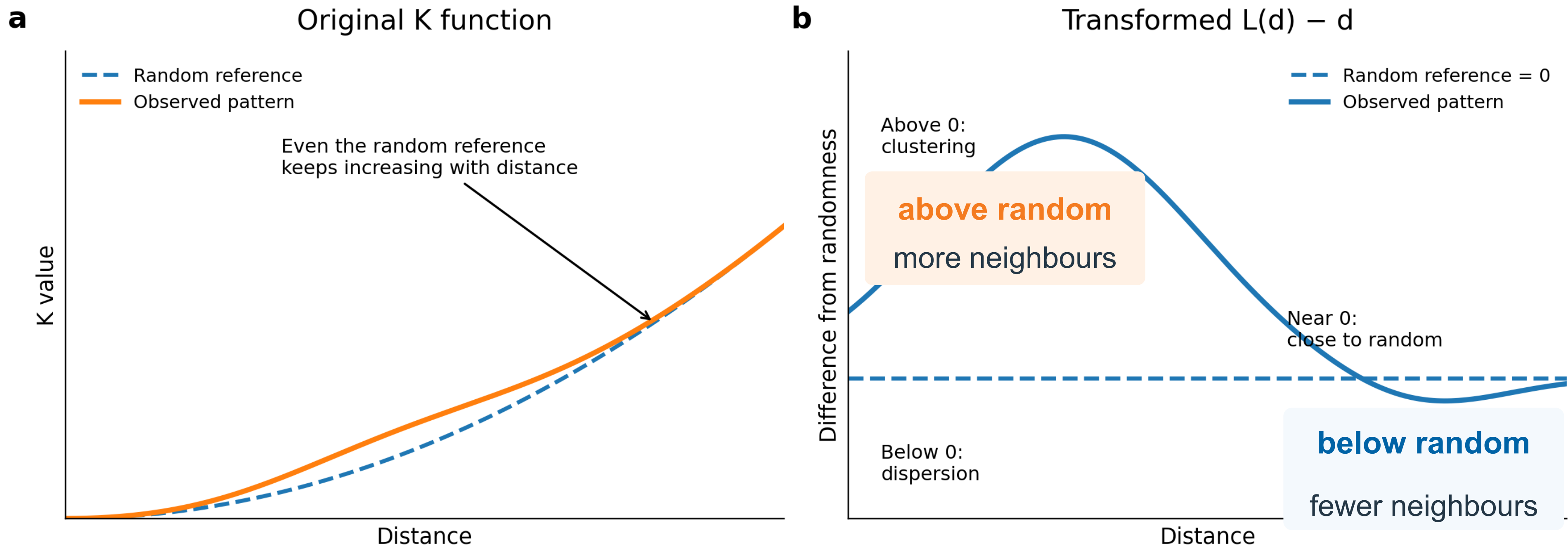
radius r

below random

fewer neighbours

Always add the phrase: at this distance

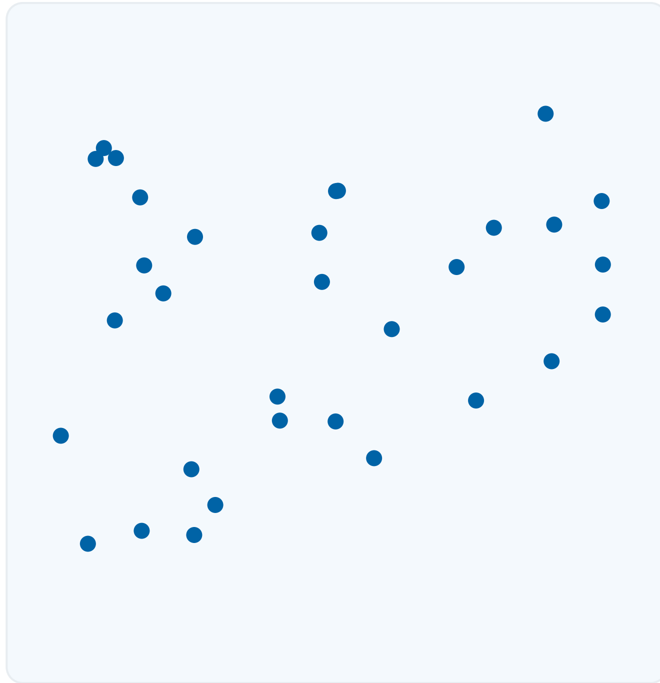
L(d) − d Moves the Random Reference to Zero

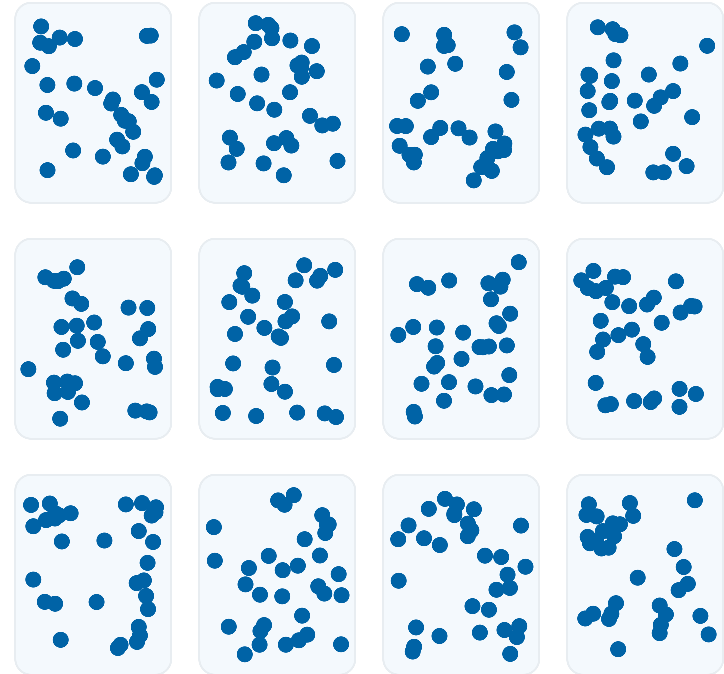


L changes the scale, not the information

L(d) – d is easier to interpret

One Random Pattern Can Be Unusually Lucky



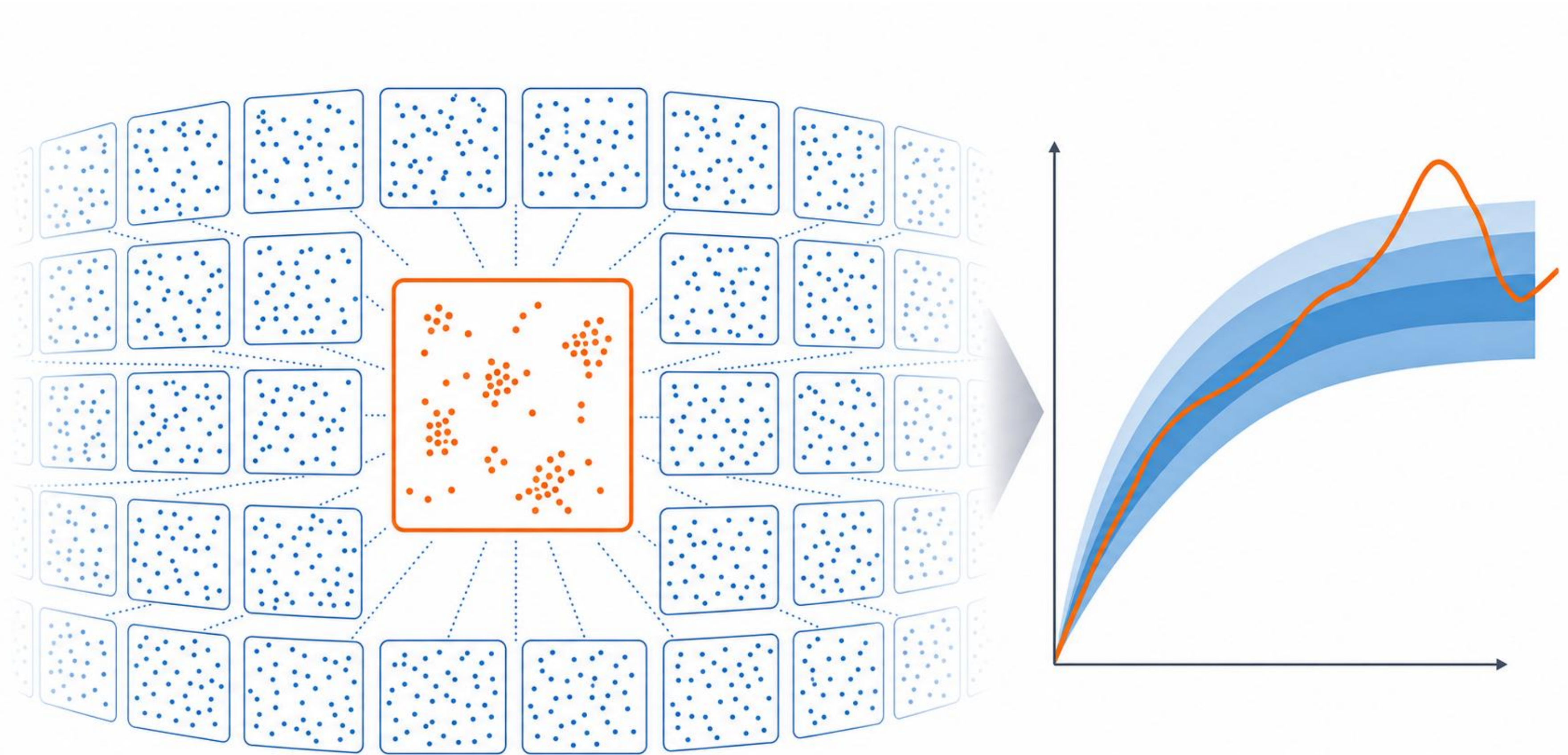


ONE RANDOM PATTERN

REPEATED RANDOM PATTERNS

One example can mislead; a distribution gives context

Monte Carlo Builds the Random Envelope



The middle 95% becomes the simulation envelope

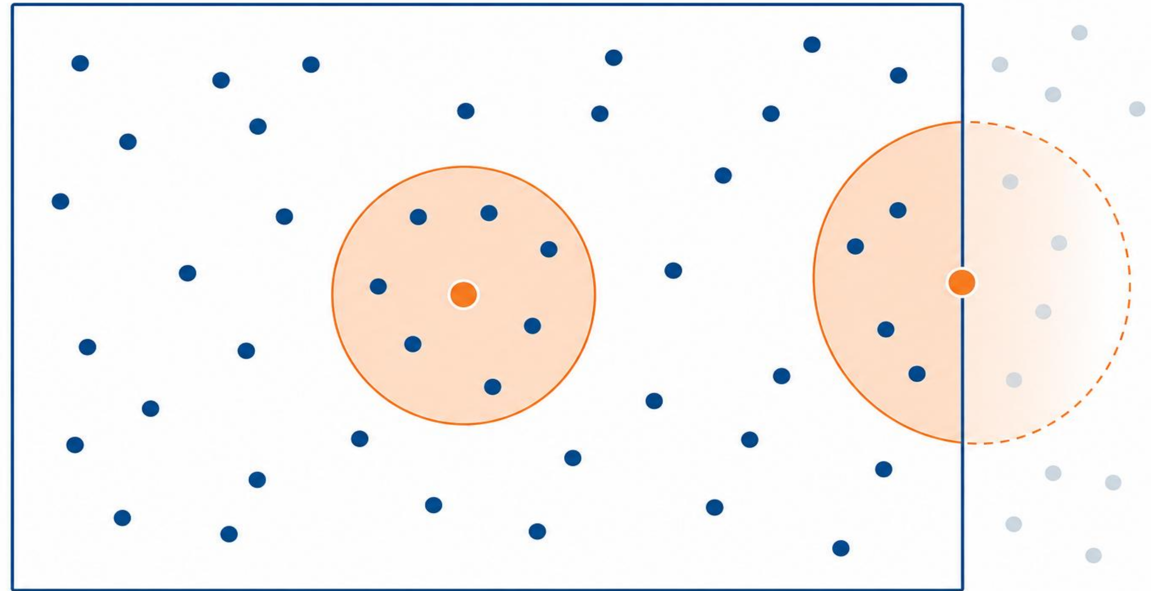
universität wien

centre point

full circle observable

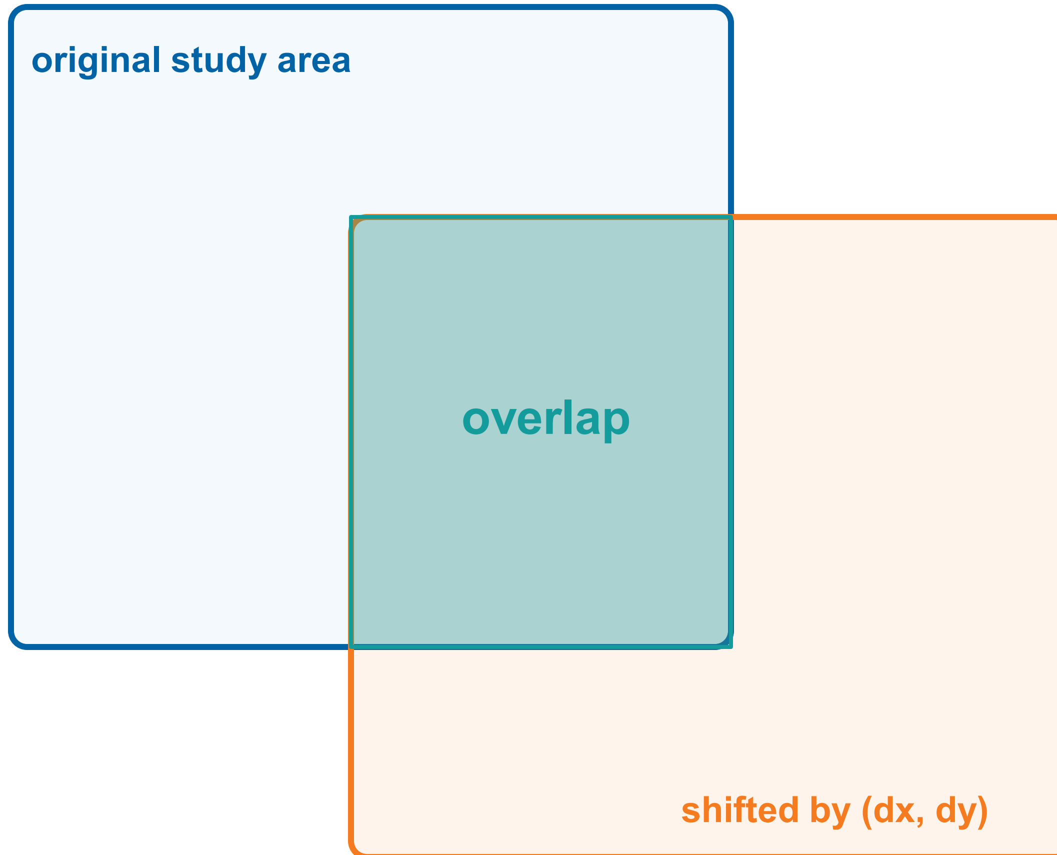
boundary point

part of the circle lies
outside the study area



Without correction,

boundary points appear to have too few neighbours.



overlap =

(width − |dx|)
×
(height − |dy|)

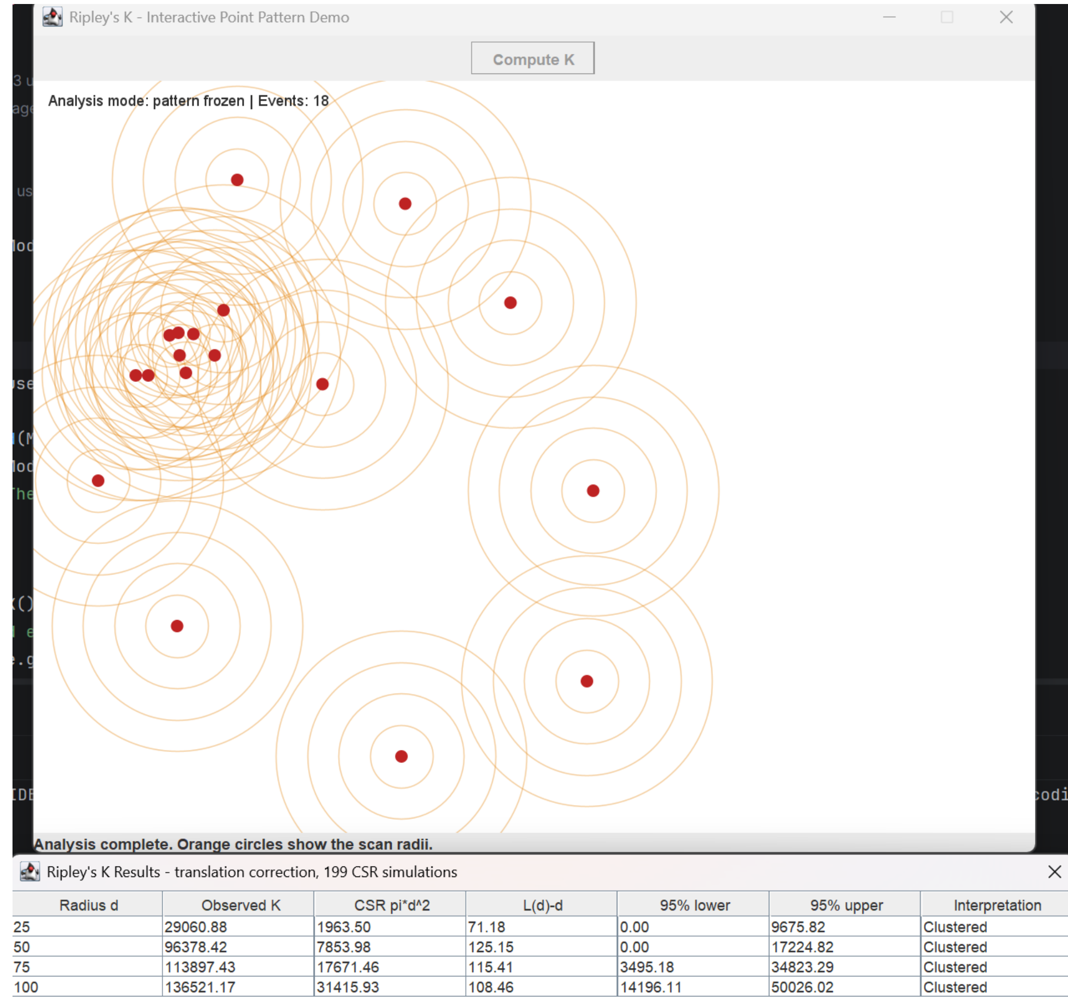
weight = area / overlap

$$K(r) = \frac{A}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} w_{ij} I(d_{ij} \leq r)$$

Smaller observable overlap → larger compensation.

The App Turns the Story into an Experiment

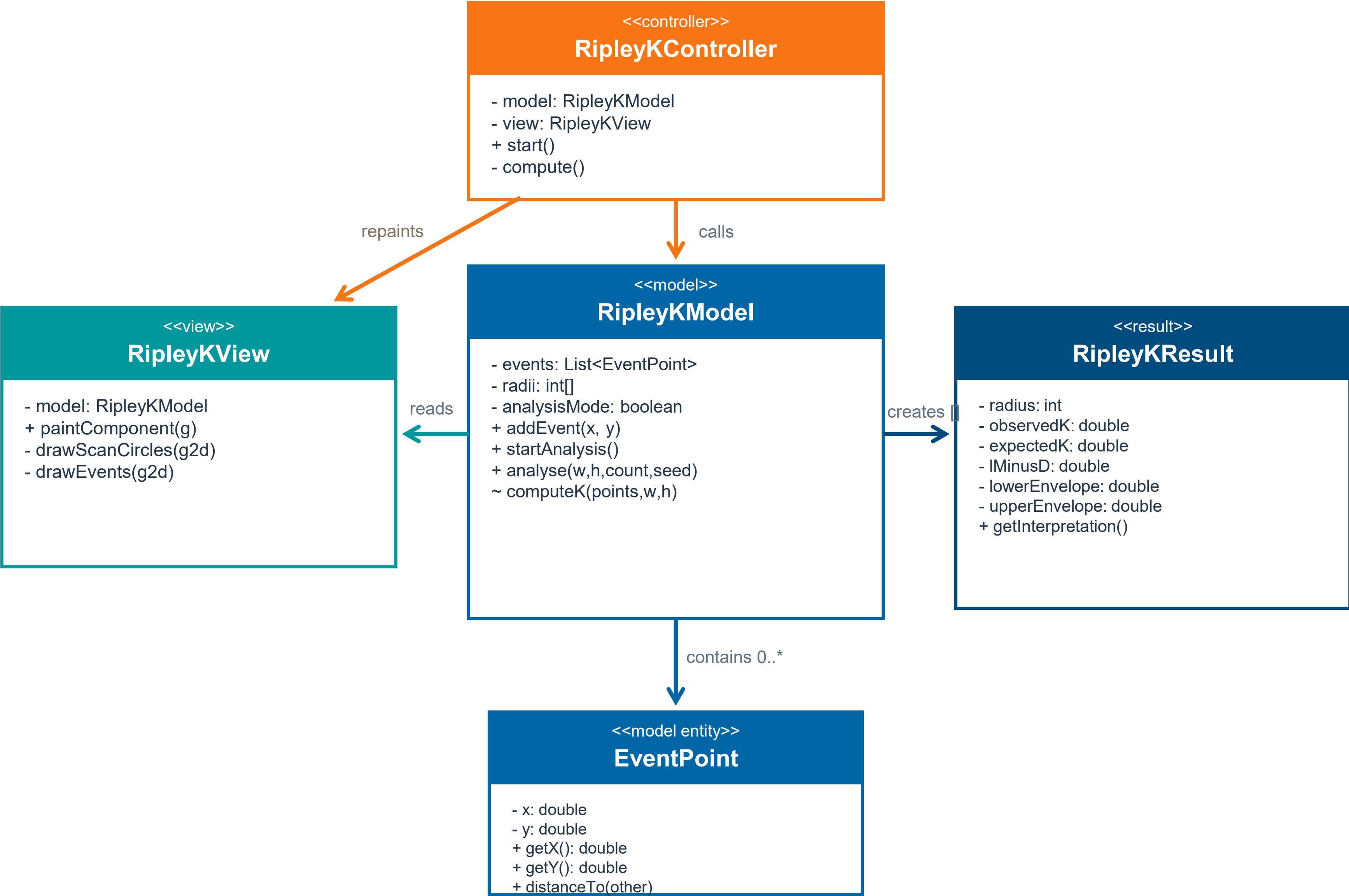
- Click to add points
- Freeze with Compute K
- Calculate observed K
- Run 199 random patterns
- Display the result table



Radius d	Observed K	CSR pi*d^2	L(d)-d	95% lower	95% upper	Interpretation
25	29060.88	1963.50	71.18	0.00	9675.82	Clustered
50	96378.42	7853.98	125.15	0.00	17224.82	Clustered
75	113897.43	17671.46	115.41	3495.18	34823.29	Clustered
100	136521.17	31415.93	108.46	14196.11	50026.02	Clustered

The interface is simple because the model does the difficult work.

UML First: See the Whole Program Before the Code



EventPoint Stores One Coordinate

```
public class EventPoint {
    private final double x;
    private final double y;

    public EventPoint(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() { return x; }
    public double getY() { return y; }

    public double distanceTo(EventPoint other) {
        double dx = x - other.x;
        double dy = y - other.y;
        return Math.sqrt(dx * dx + dy * dy);
    }
}
```

Plain immutable data class

the model remains independent of AWT

This matches EventPoint.java: constructor + getters + distanceTo().

RipleyKModel Owns the State and Interaction Mode

```
public class RipleyKModel {

    private final List<EventPoint> events =
        new ArrayList<>();
    private final int[] radii;
    private boolean analysisMode;

    public RipleyKModel(int[] radii) {
        this.radii = radii.clone();
    }

    public void addEvent(double x, double y) {
        if (!analysisMode) {
            events.add(new EventPoint(x, y));
        }
    }

    public void startAnalysis() {
        analysisMode = true;
    }
}
```

events
point data

radii
analysis
scales

analysisMode
controls
whether
points can
change

The model, not the controller, enforces the frozen-pattern rule.

The Controller Handles a Mouse Release

```
view.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        if (model.isAnalysisMode()) {
            status.setText(
                "The pattern is frozen in analysis mode.");
            return;
        }

        model.addEvent(e.getX(), e.getY());
        status.setText(
            "Added event " + model.size()
            + " at (" + e.getX()
            + ", " + e.getY() + ").");
        view.repaint();
    }
});
```

1. guard the mode

2. update the model

3. update the status

4. repaint the view

paintComponent() Reads and Draws the Model State

universität wien

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    Graphics2D g2d = (Graphics2D) g.create();
    g2d.setRenderingHint(
        RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);

    if (model.isAnalysisMode()) {
        drawScanCircles(g2d);
    }
    drawEvents(g2d);
    drawStatus(g2d);

    g2d.dispose();
}
```

analysis mode: draw circles

always: draw events

always: draw status

The view asks the model what to draw; it does not change model state.

The View Draws Events and Scan Circles

```java
private void drawEvents(Graphics2D g2d) {
    g2d.setColor(new Color(190, 35, 35));

    for (EventPoint event : model.getEvents()) {
        int x = (int) Math.round(event.getX());
        int y = (int) Math.round(event.getY());

        g2d.fillOval(
            x - EVENT_RADIUS,
            y - EVENT_RADIUS,
            EVENT_RADIUS * 2,
            EVENT_RADIUS * 2);
    }
}

private void drawScanCircles(Graphics2D g2d) {
    g2d.setColor(SCAN_CIRCLE_COLOR);

    for (EventPoint event : model.getEvents()) {
        int x = (int) Math.round(event.getX());
        int y = (int) Math.round(event.getY());

        for (int radius : model.getRadii()) {
            g2d.drawOval(
                x - radius, y - radius,
                radius * 2, radius * 2);
        }
    }
}
```

Both methods convert model coordinates into Java2D bounding boxes

Compute K Freezes the Pattern and Starts Analysis

```
private void compute() {
    if (model.size() < 2) {
        JOptionPane.showMessageDialog(
            frame,
            "Add at least two event points "
                + "before computing K.",
            "Not enough events",
            JOptionPane.WARNING_MESSAGE);
        return;
    }

    model.startAnalysis();
    computeButton.setEnabled(false);
    status.setText(
        "Analysis complete. Orange circles "
            + "show the scan radii.");
    view.repaint();

    RipleyKResult[] results = model.analyse(
        RipleyKView.WIDTH,
        RipleyKView.HEIGHT,
        MONTE_CARLO_SIMULATIONS,
        RANDOM_SEED);
    showResults(results);
}
```

validate

freeze

analyse

display

computeK() Validates and Prepares Its Accumulators

```
double[] computeK(List<EventPoint> points,
                  double width, double height) {
    validateWindow(width, height);

    int n = points.size();
    if (n < 2) {
        throw new IllegalArgumentException(
            "At least two events are required.");
    }

    double area = width * height;
    double[] weightedCounts =
        new double[radii.length];

    // pair loops follow...
}
```

radii is a model field

one accumulator per radius

invalid input fails early

Setup happens once before any pair or radius loop.

For Each Ordered Pair, Compute Its Edge Weight

universität wien

```
for (int i = 0; i < n; i++) {
    EventPoint first = points.get(i);

    for (int j = 0; j < n; j++) {
        if (i == j) {
            continue;
        }

        EventPoint second = points.get(j);
        double dx = Math.abs(
            first.getX() - second.getX());
        double dy = Math.abs(
            first.getY() - second.getY());
        double overlapArea =
            (width - dx) * (height - dy);

        if (overlapArea <= 0) {
            continue;
        }

        double edgeWeight = area / overlapArea;
        // distance and radius checks follow...
    }
}
```

skip self-pairs

measure window overlap

invert overlap to get weight

One Pair Distance Is Checked Against Every Radius

```
double distance = first.distanceTo(second);

for (int r = 0; r < radii.length; r++) {
    if (distance <= radii[r]) {
        weightedCounts[r] += edgeWeight;
    }
}
```

distance computed once

r indexes the radii array

add weight when inside

After Counting, Normalise the Weighted Sum

```
double[] values = new double[radii.length];
double denominator = n * (double) (n - 1);

for (int r = 0; r < radii.length; r++) {
    values[r] =
        area * weightedCounts[r] / denominator;
}

return values;
```

weightedCounts[r]
translation-corrected pairs

x

area / n(n-1)
dataset scale

=

values[r]
K at radius r

The returned array follows the same order as the model's radii array.

analyse() Computes the Observed K Before Simulating

```
public RipleyKResult[] analyse(
        double width, double height,
        int simulationCount, long randomSeed) {

    if (events.size() < 2) {
        throw new IllegalStateException(
            "At least two events are required.");
    }
    validateWindow(width, height);
    if (simulationCount < 19) {
        throw new IllegalArgumentException(
            "Use at least 19 simulations "
                + "for a useful envelope.");
    }

    double[] observed =
        computeK(events, width, height);
    double[][] simulatedByRadius =
        new double[radii.length][simulationCount];
    Random random = new Random(randomSeed);
```

1. validate analysis input

2. compute observed K

3. prepare simulation storage

Observed and simulated patterns use the same computeK() estimator

Monte Carlo Is a Loop Around the Same Estimator

universität wien

```
double[][] simulatedByRadius =
    new double[radii.length][simulationCount];

for (int simulation = 0;
     simulation < simulationCount;
     simulation++) {

    List<EventPoint> randomEvents =
        new ArrayList<>();
    for (int i = 0; i < events.size(); i++) {
        randomEvents.add(new EventPoint(
            random.nextDouble() * width,
            random.nextDouble() * height));
    }

    double[] simulatedK =
        computeK(randomEvents, width, height);
    for (int r = 0; r < radii.length; r++) {
        simulatedByRadius[r][simulation] =
            simulatedK[r];
    }
}
```

outer loop one simulation

inner loop copy every radius

[radius][simulation]

The code stores all simulated K values together for each radius

Build the Envelope Radius by Radius

```
for (int r = 0; r < radii.length; r++) {
    Arrays.sort(simulatedByRadius[r]);

    double lower = percentile(
        simulatedByRadius[r], 0.025);
    double upper = percentile(
        simulatedByRadius[r], 0.975);

    double expected =
        Math.PI * radii[r] * radii[r];
    double lMinusD =
        Math.sqrt(observed[r] / Math.PI)
        - radii[r];

    results[r] = new RipleyKResult(
        radii[r], observed[r], expected,
        lMinusD, lower, upper);
}
```

sort one radius's simulations

take 2.5% and 97.5% percentiles

create one RipleyKResult per radius

analyse() returns RipleyKResult[], which the controller displays as table rows

RipleyKResult Encapsulates the Interpretation Rule

```
public String getInterpretation() {
    if (observedK > upperEnvelope) {
        return "Clustered";
    }
    if (observedK < lowerEnvelope) {
        return "Dispersed";
    }
    return "Consistent with CSR";
}
```

above upper → Clustered

below lower → Dispersed

inside envelope → CSR

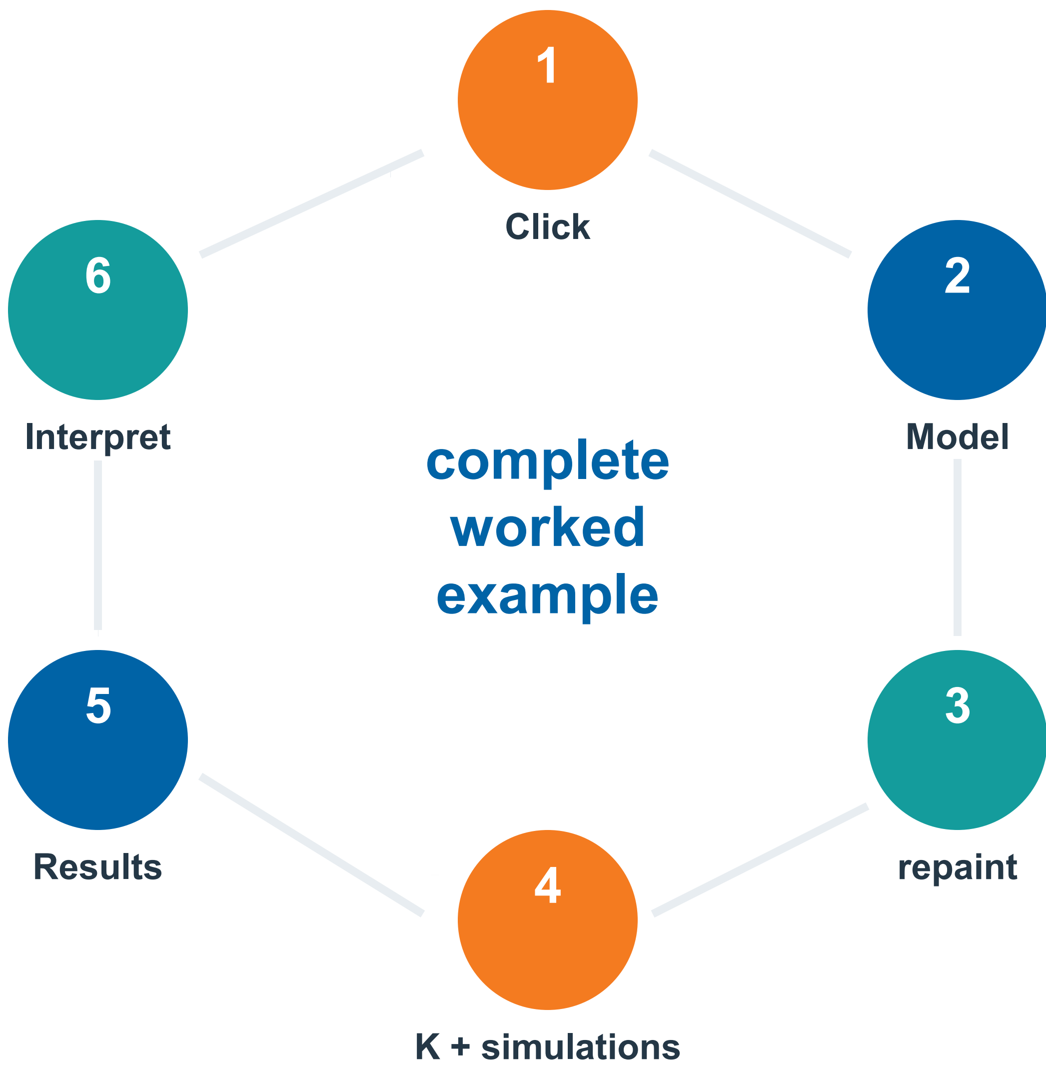
The controller asks the result object for the final label.

The results table keeps every radius together

d	Observed K	CSR K	L(d)−d	95% envelope	Meaning
25	2,200	1,963	+3.2	1,500–2,500	**CSR**
50	9,200	7,854	+4.1	6,100–8,500	**Clustered**
75	15,400	17,671	−4.8	15,900–19,200	**Dispersed**

One row = one distance scale.

universität wien



universität wien

External libraries

&

Final exam scopes