

# Rust-Powered Business Intelligence: Building High-Performance User-Centric Dashboards

The Modern Approach to High-Performance Web Analytics

**Shahzeb Akhtar**

UnitedLex



IP strategy and technology leader with expertise in patent intelligence, data science, and litigation support



**Director at UnitedLex:**

- Lead the IP Data Science Team
- Innovate at the intersection of intellectual property and technology
- Named in the IAM Strategy 300 list of world's top IP strategists

**Creator of Vantage for IP:**

- Patent intelligence platform using natural language processing
- Analyzes 100,000+ patent assets
- Identifies competitive threats, assesses patent strength, and uncovers licensing opportunities

**Technical skills:**

- Develop deep learning models for patent metrics
- Design user-friendly platforms for patent analytics



# The Challenge with Modern Dashboards

## 🚧 Performance Bottlenecks

Heavy data processing and complex calculations are often handled by JavaScript, leading to slow rendering and sluggish user experiences.

## 🚨 Security Risks

JavaScript's dynamic nature can be a source of vulnerabilities. Running untrusted or complex code can expose the application to various attacks.

## 🗑️ Wasted Resources

Inefficient code execution and frequent data transfers put a strain on both the client's device and the server.



# **Perfect Partnership: Rust + Wasm**

## **Unmatched Memory Safety**

Rust's ownership and borrowing model eliminates entire classes of bugs like null pointers and data races, which are common sources of security vulnerabilities.

## **Zero-Cost Abstractions**

Rust provides powerful abstractions without runtime overhead, meaning you get high-level functionality with low-level performance.

## **Built for Concurrency**

Rust's type system makes it easy to write safe, concurrent code, which is ideal for parallel data processing in a dashboard.

## **The "Assembly" of the Web**

A low-level, binary instruction format for a stack-based virtual machine. It's a compilation target for languages like Rust, C++, and Go.

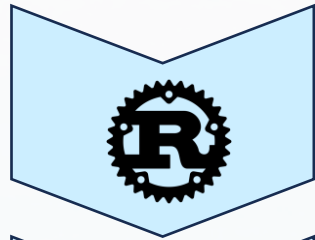
## **Near-Native Performance**

Wasm executes at speeds close to native code, far outperforming traditional JavaScript for computationally intensive tasks.

## **A Secure Sandbox**

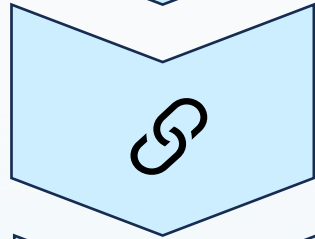
Wasm runs in a secure, sandboxed environment, isolated from the rest of the web page. This prevents malicious code from accessing the host system.

# How It Works: The Rust to Wasm Workflow

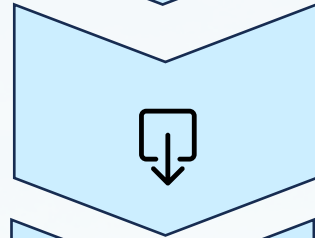


## Write the core logic in Rust

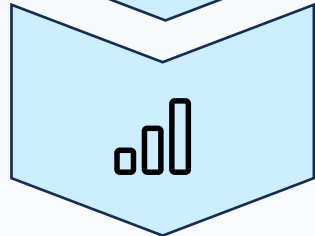
Develop performance-critical functions (e.g., data aggregation, complex filtering) in Rust.



**Compile to Wasm:** Use tools like wasm-pack to compile the Rust code into a .wasm file and a JavaScript "glue" file.

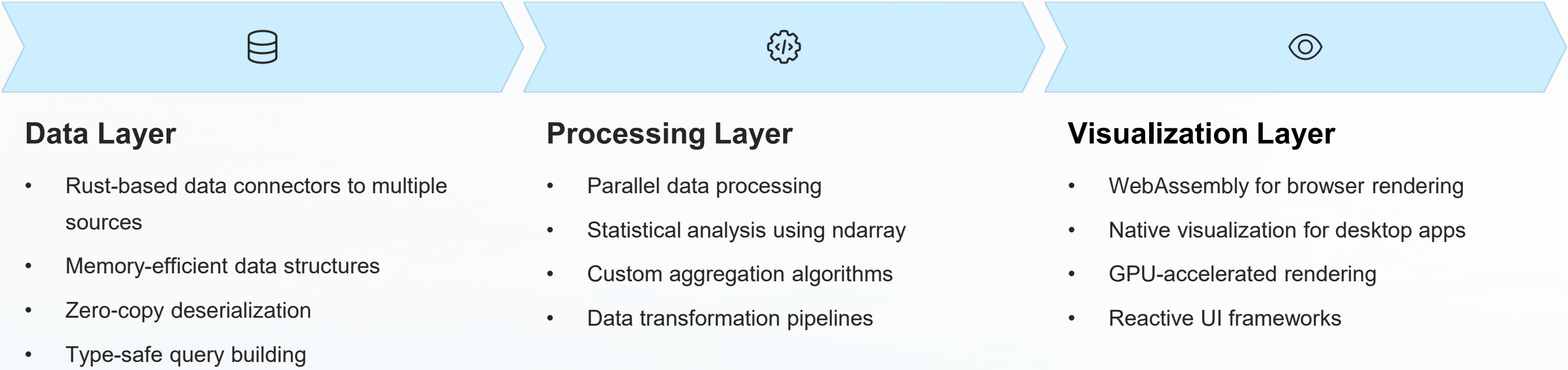


**Import and Use:** The JavaScript front-end imports the Wasm module and calls the high-performance functions as needed, treating the Wasm module like a super-fast library.



**Visualize with JavaScript:** The UI layer (React, Vue, etc.) remains in JavaScript, using the results from the Wasm module to render charts and tables.

# Technical Architecture for Rust-Powered Dashboards





The Rust ecosystem provides excellent libraries for each layer of the dashboard architecture:


Data Handling	Processing	Visualization
<ul style="list-style-type: none"><li>• sqlx for database interaction</li><li>• serde for serialization</li><li>• arrow for columnar data</li><li>• polars for data frames</li></ul>	<ul style="list-style-type: none"><li>• rayon for parallel processing</li><li>• ndarray for numerical computing</li><li>• smartcore for machine learning</li><li>• futures for async processing</li></ul>	<ul style="list-style-type: none"><li>• wasm-bindgen for WebAssembly</li><li>• plotters for chart generation</li><li>• egui for immediate mode GUI</li><li>• tauri for desktop applications</li></ul>



# Case Study: Figma's Multiplayer Engine

 **The Challenge:** Figma needed to handle complex, real-time synchronization between multiple users editing the same design file.

 **The Solution:** They rewrote their core multiplayer engine in Rust

 **The Results:** This approach dramatically improved performance, enabling a seamless collaborative experience with minimal latency.

Metric	Old server		New server	Improvement
Peak average per-worker memory usage	4.2gb	→	1.1gb	3.8x smaller
Peak average per-machine CPU usage	24%	→	4%	6x smaller
Peak average file serve time	2s	→	0.2s	10x faster
Peak worst-case save time	82s	→	5s	16.4x faster



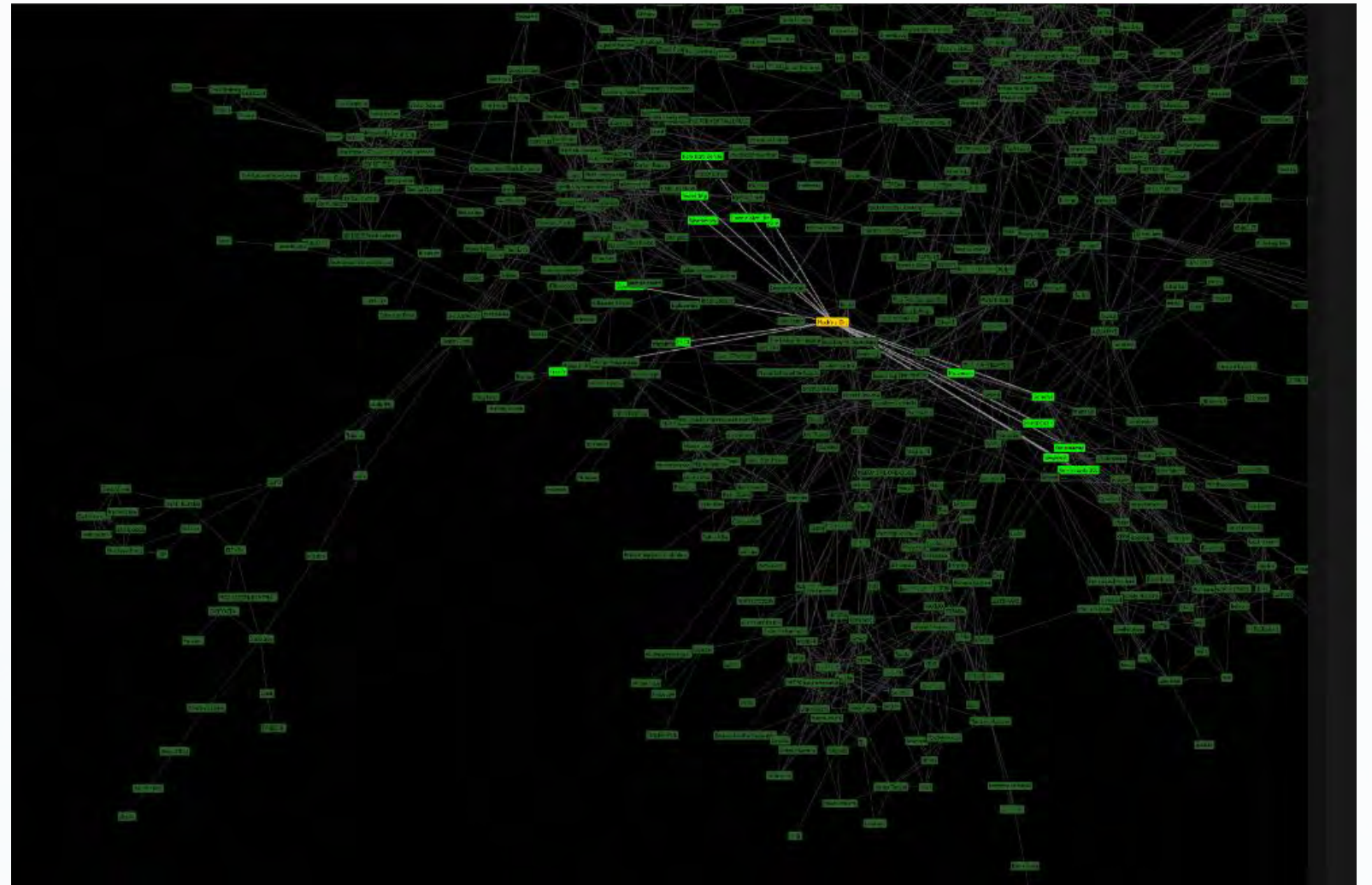
Read more at <https://www.figma.com/blog/rust-in-production-at-figma/>

# Case Study: Speeding Up Graph Viz with Rust + WebAssembly

🏔️ **The Challenge:** Graph visualization running at <10 FPS on powerful desktop, poor user experience

💡 **The Solution:** Employ Rust + Webassembly to optimize the calculations caused by user interaction


🏆 **The Results:** Users can now to interact with the same visualization at well over 60 FPS, even on weaker mobile devices.





Read more at <https://cprimozic.net/blog/speeding-up-webcola-with-webassembly/>

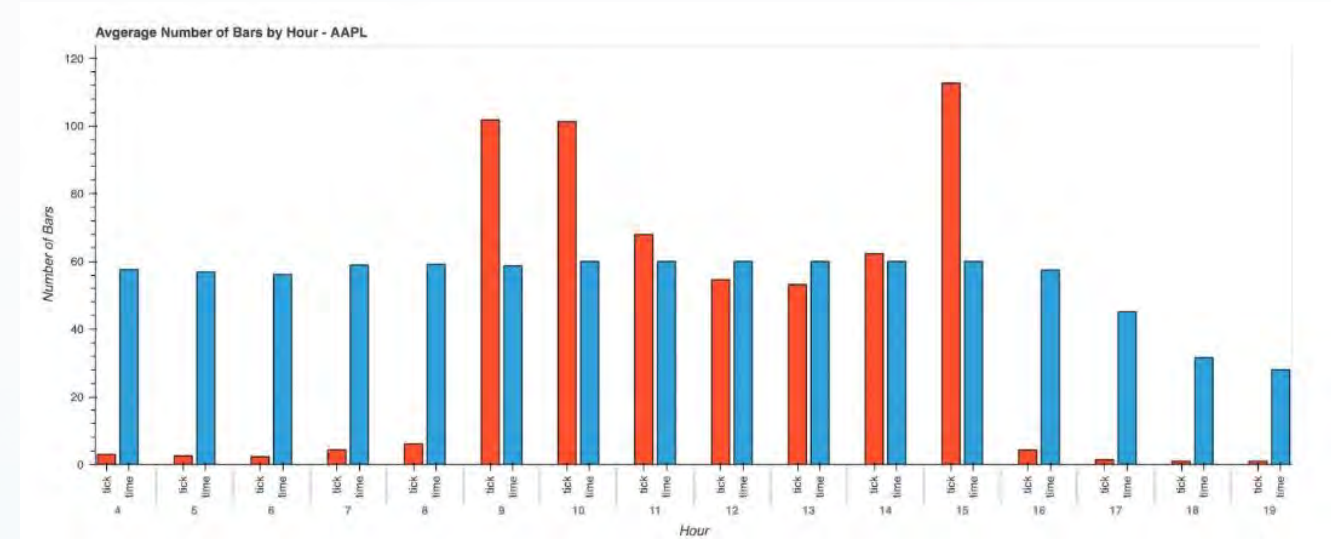


# Case Study: Data wrangling in Financial Applications

 **The Challenge:** Process datasets containing close to a billion rows on a single machine

 **The Solution:** Employ Polars, a blazingly fast DataFrame library for manipulating structured data. The core is written in Rust, and available for Python, R and NodeJS.

 **The Results:** Benchmark tests have shown Polars outperforming Pandas and other competing solutions by 8–15x.



“


By leveraging Polars' capabilities, market data users can process and analyze large datasets with ease, allowing for timely and data-driven decisions in response to market changes.





Read more at:

- <https://databento.com/blog/polars>
- <https://databento.com/blog/downsampling-pricing-data>

# Visualization libraries in Rust

 **Plotters:** This is a powerful and flexible drawing library that focuses on data plotting. It supports various backends, including bitmap, SVG, and HTML5 canvas via WebAssembly.


 **Charming:** This library is a Rust wrapper for Apache ECharts, a popular and powerful JavaScript visualization library. Charming provides a declarative and user-friendly Rust API that allows you to create a wide variety of high-quality charts. It can render to multiple formats, including HTML, SVG, and various image formats.


 **Plotly.rs:** As the name suggests, this is a Rust plotting library powered by Plotly.js. It's a great choice if you need the extensive chart types and interactive features of Plotly, with the safety and performance of Rust. It's especially useful for scientific and statistical visualization.





# User-Centric Design Principles for Dashboards


While Rust excels at performance and security, successful dashboard implementations must balance technical excellence with thoughtful design.


 **Know Your Audience:** Who the users are, what their goals are, and what decisions they need to make.


 **Keep it Simple:** Use a clear, logical layout and a consistent visual language.


 **Provide Context:** A good dashboard provides context by showing data against a benchmark

 **Prioritize Information:** Use visual hierarchy to guide the user's attention to the most important information first.

 **Enable Customization:** Allowing users to filter data, change date ranges to make the dashboard more relevant to their specific workflow.


 **Ensure Interactivity:** Allow users to drill down into specific data points, hover over charts for more details, or click to explore underlying data. This interactivity encourages exploration and deeper understanding.


 **Choose the Right Visuals:** Select the chart type that best represents the data and supports the user's task. For example, use a line chart to show trends over time, a bar chart to compare categories, and a pie chart for parts-to-whole relationships.


 **Accessibility:** Design with accessibility in mind. This includes using color palettes that are accessible to users with color blindness, providing clear labels, and ensuring the dashboard is navigable with a keyboard or screen reader.

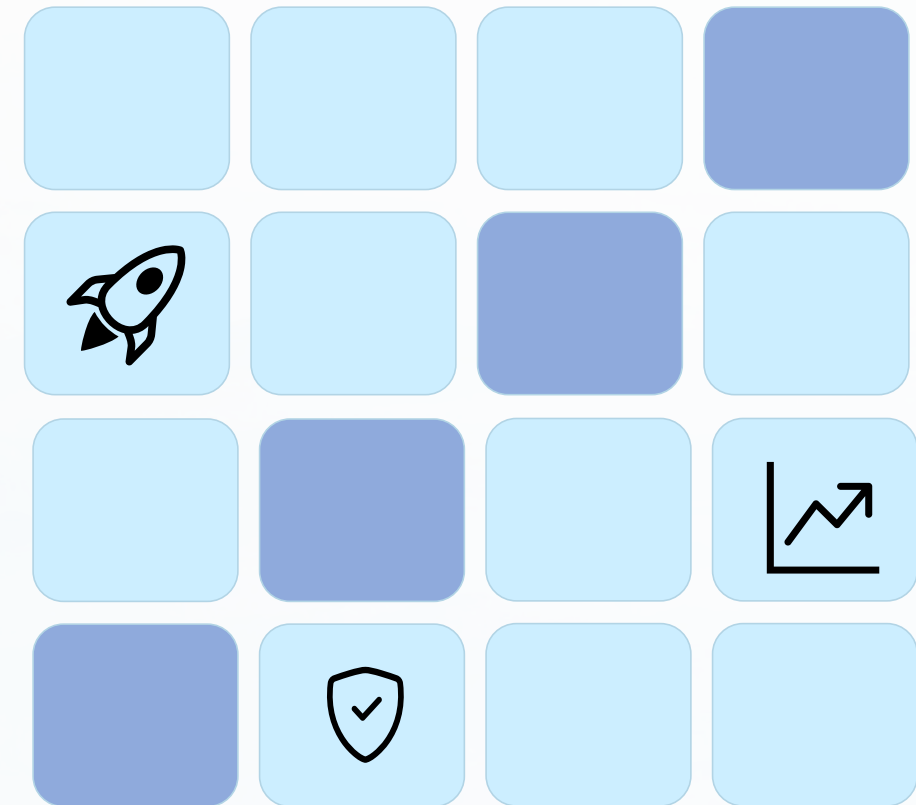


# The Business Impact: Why It Matters

 **Blazing Speed:** Deliver an outstanding user experience with dashboards that load faster and feel more responsive, even with large datasets.

 **Ironclad Security:** Mitigate common security threats with Rust's memory safety and WebAssembly's sandboxed environment.

 **Developer Productivity:** Leverage a type-safe language that helps developers write correct and maintainable code from the start.



# Key Takeaways & Next Steps

## Performance Transforms Experience

Rust's speed enables real-time interaction with complex data, reducing decision time and increasing user engagement.

## Safety Creates Trust

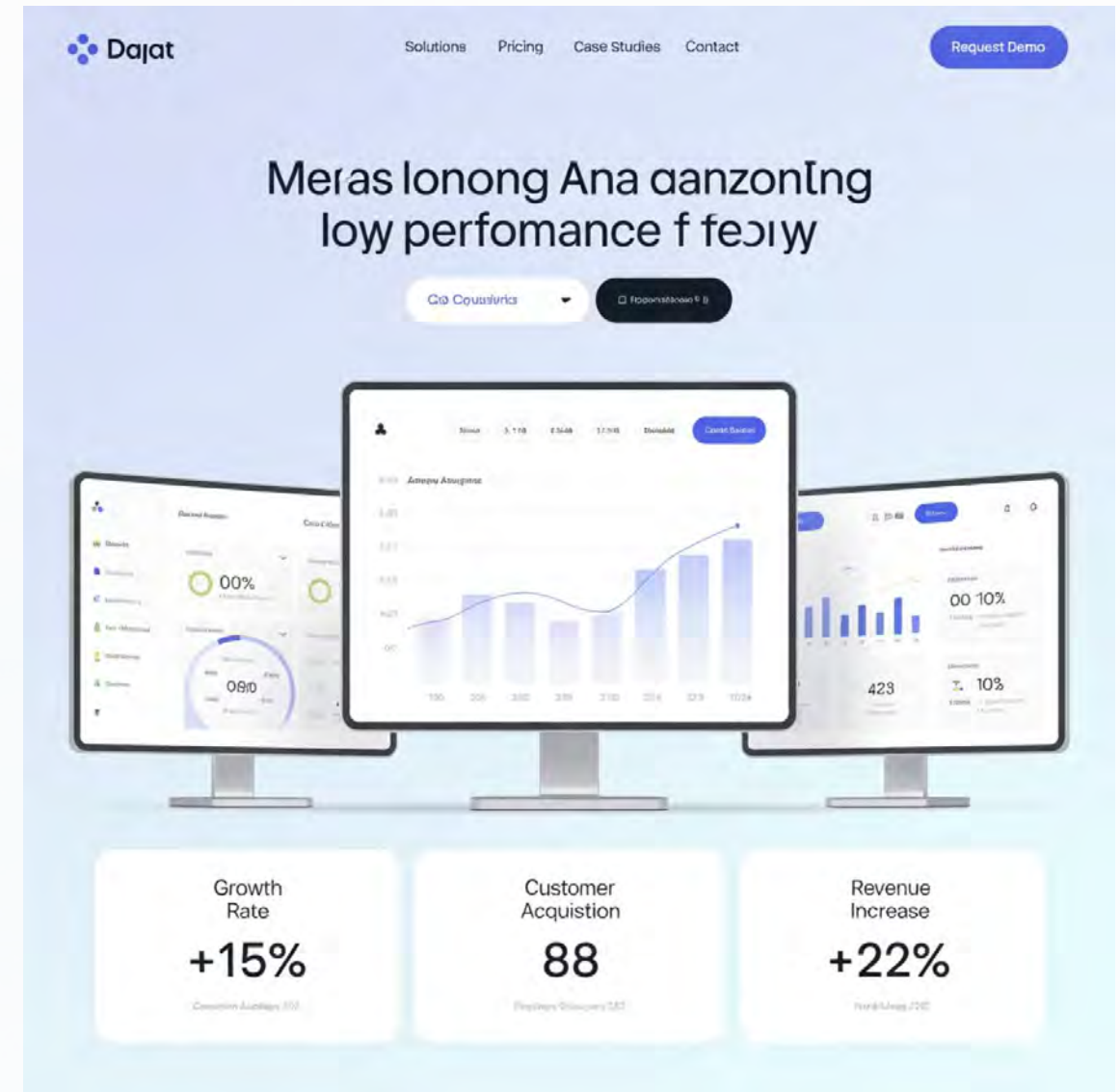
Memory safety eliminates 99.9% of vulnerabilities, critical for sensitive business data and regulatory compliance.

## User-Centric Design Matters

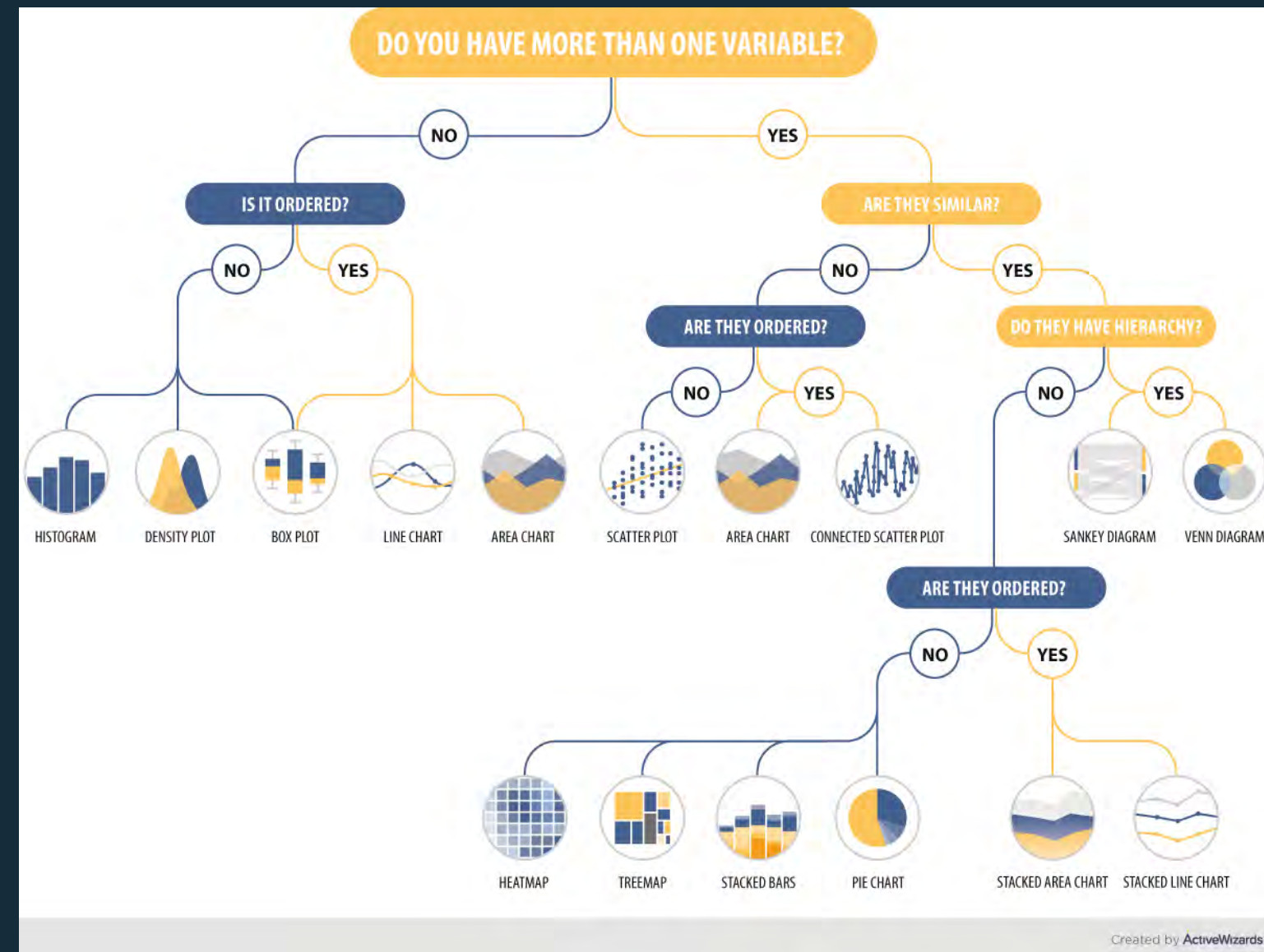
Balancing technical excellence with thoughtful UX design leads to higher satisfaction and reduced training needs.

## Next Steps for Your Organization

1. Assess your current BI performance bottlenecks and security vulnerabilities
2. Identify high-impact use cases where Rust could provide immediate value
3. Start with a targeted proof-of-concept for a critical dashboard component
4. Measure performance improvements and user satisfaction changes
5. Develop a phased migration plan based on demonstrated results



# Thank You



Check out this infographic by Active Wizards to help in choosing the right chart type

<https://activewizards.com/blog/how-to-choose-the-right-chart-type-infographic/>

