The Architecture That Gets You Here Won’t Take You There

4 Practical Ways to Scale Your Application

RAUL JUNCO
MAR 15, 2025

58 13 4 Share

When you're starting out, a simple monolithic app can serve you well. It’s easy, reliable, and quick to build.

But what got you here might not get you to the next level.

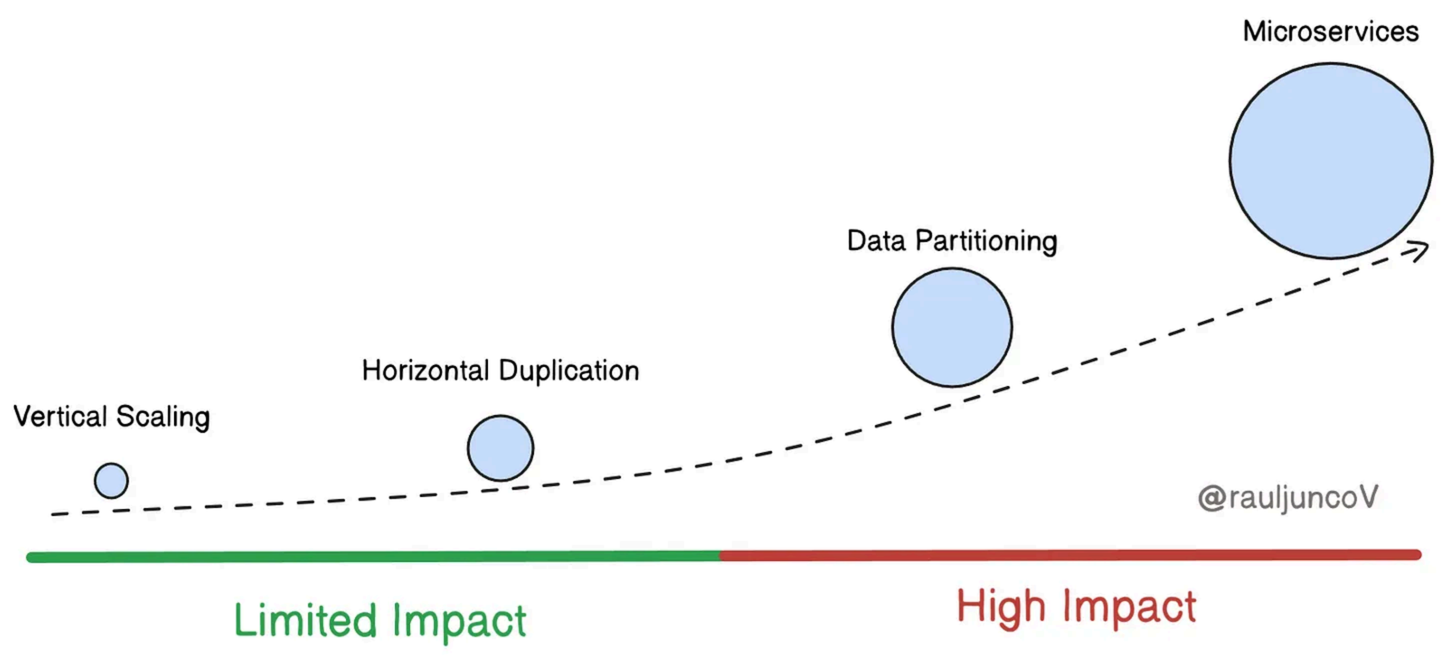
Growth brings more customers, more transactions, and bigger challenges.

Many engineers view architectural changes as a solution to a bad design decision. In reality, such changes are a natural evolution that reflects your success, not your failure.

Systems change over time; changing your architecture doesn’t imply the original design was flawed; it simply means you've reached a new stage and require different trade-offs.

Limited Impact vs. High Impact.

Scaling strategies can be classified into **Limited Impact** and **High Impact**.



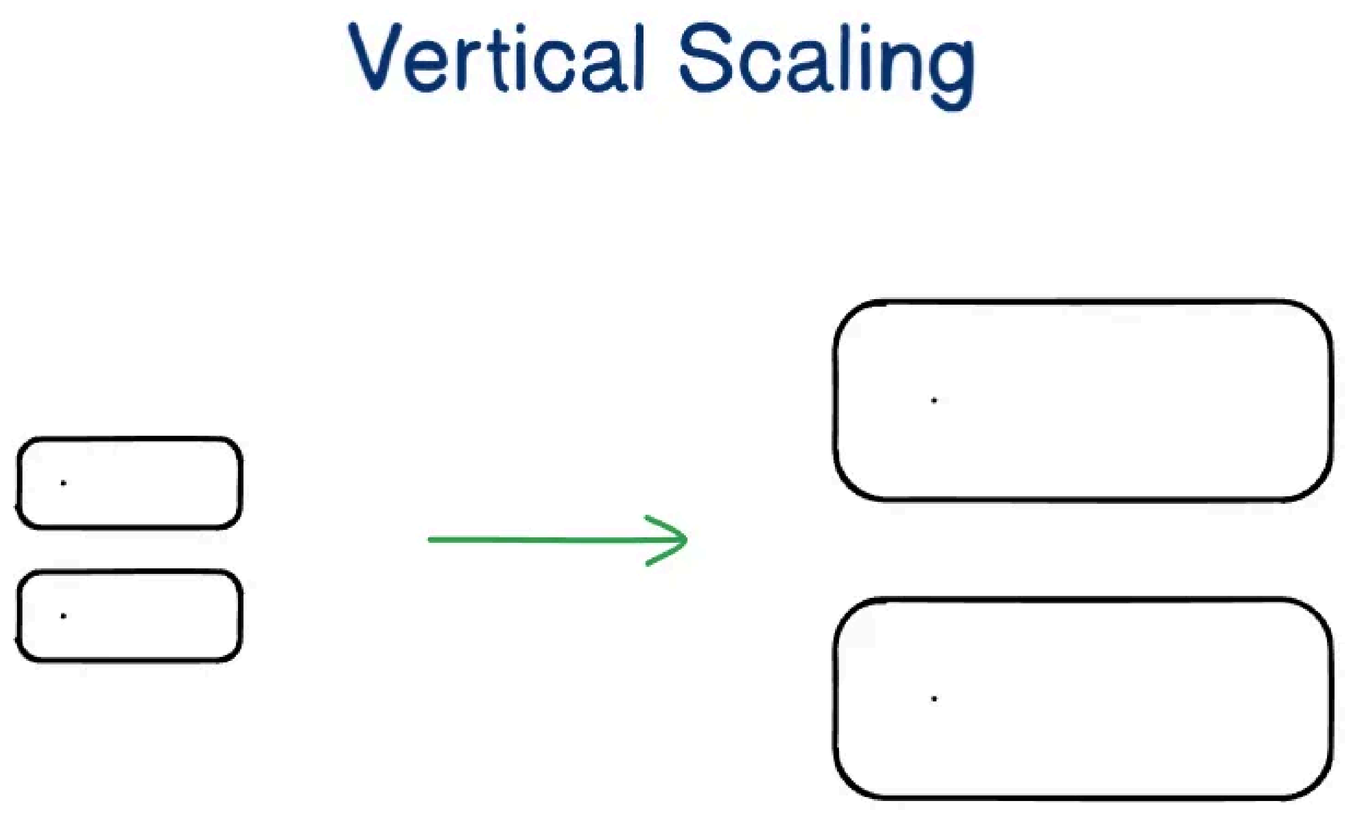
Limited-impact scaling strategies are simpler, quicker, and typically require fewer changes to your current system. They’re ideal for immediate or moderate growth.

High-impact strategies have greater long-term benefits but require significant architectural changes.

Here are four simple examples:

1. Vertical Scaling

Vertical scaling means upgrading your existing server’s resources—more CPU power, additional RAM, or increased storage—to handle extra load.



When to use: Ideal for quickly addressing immediate performance issues without changing your app.

Pros:

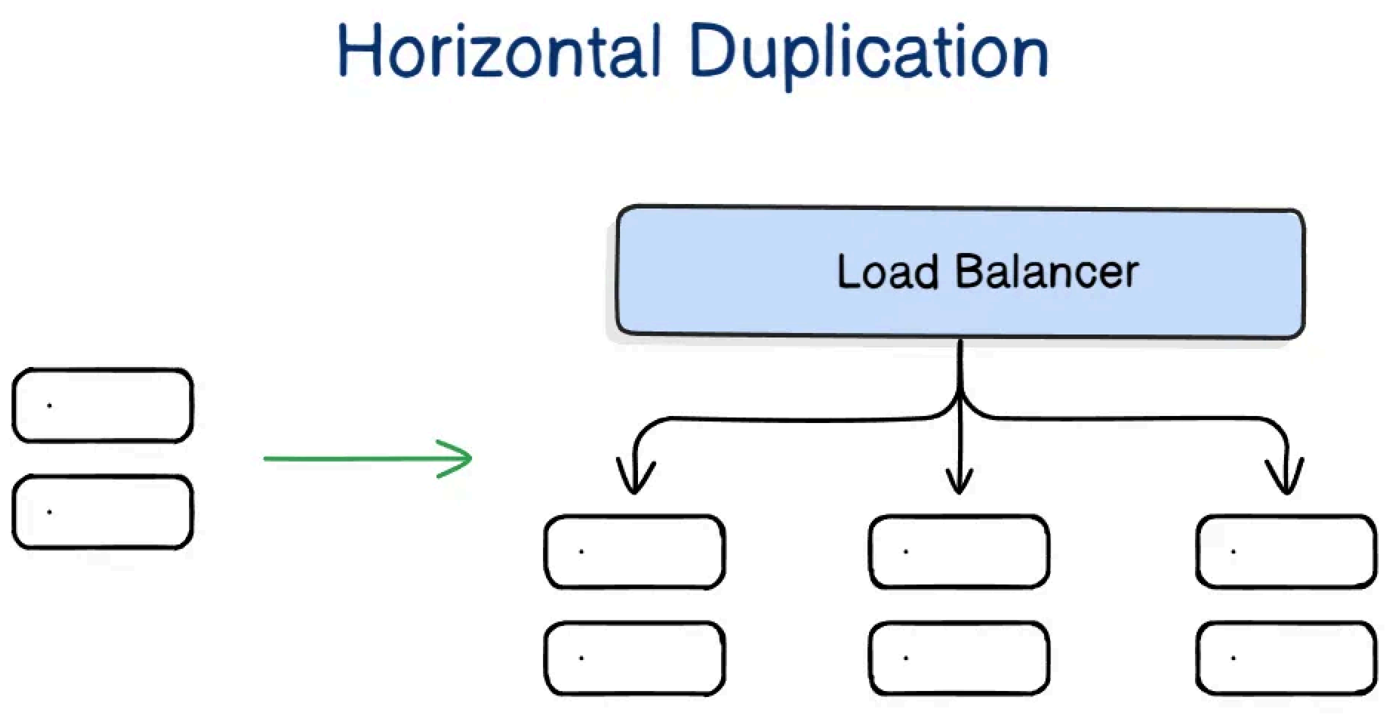
- Quick, easy, and requires no changes to the application.
- Low complexity and minimal operational overhead.

Cons:

- Limited scalability—eventually hits hardware limits.
- Single server means a single point of failure.
- Hardware costs increase significantly at high levels.

2. Horizontal Duplication

Run multiple copies of your app on separate servers, distributing traffic through a load balancer.



When to use: Useful when you need reliability and easy incremental growth.

Pros:

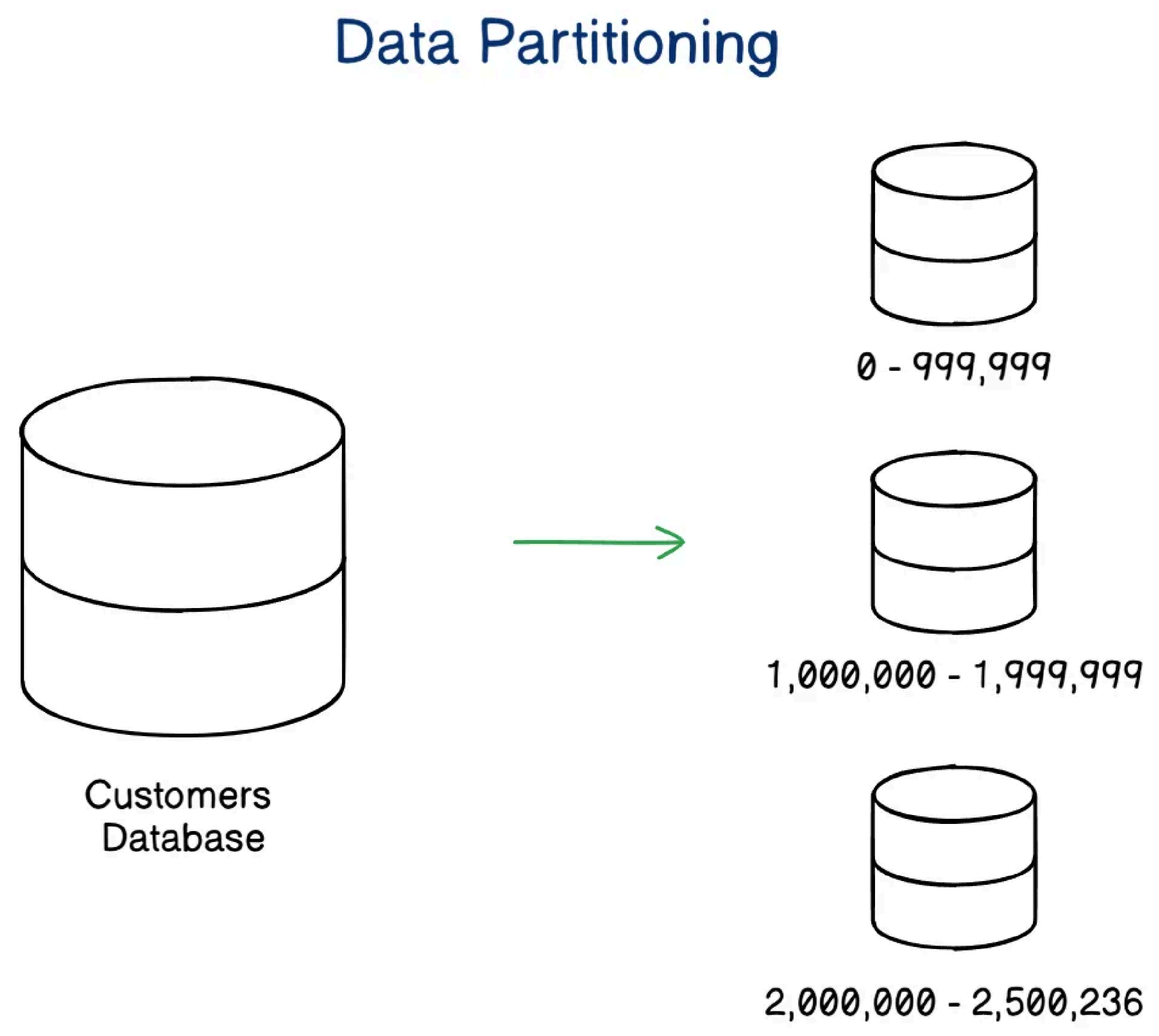
- High availability and fault tolerance.
- Simple scaling by adding new instances.

Cons:

- Management complexity increases.
- Potential issues maintaining consistent data states across instances.

3. Data Partitioning (Sharding)

Data partitioning divides your data into multiple databases based on certain criteria—like user ID ranges, geographic location, or product category—allowing each database to handle a portion of the load.



When to use: When your database hits capacity limits or becomes slow, partitioning helps distribute and handle significant growth.

Pros:

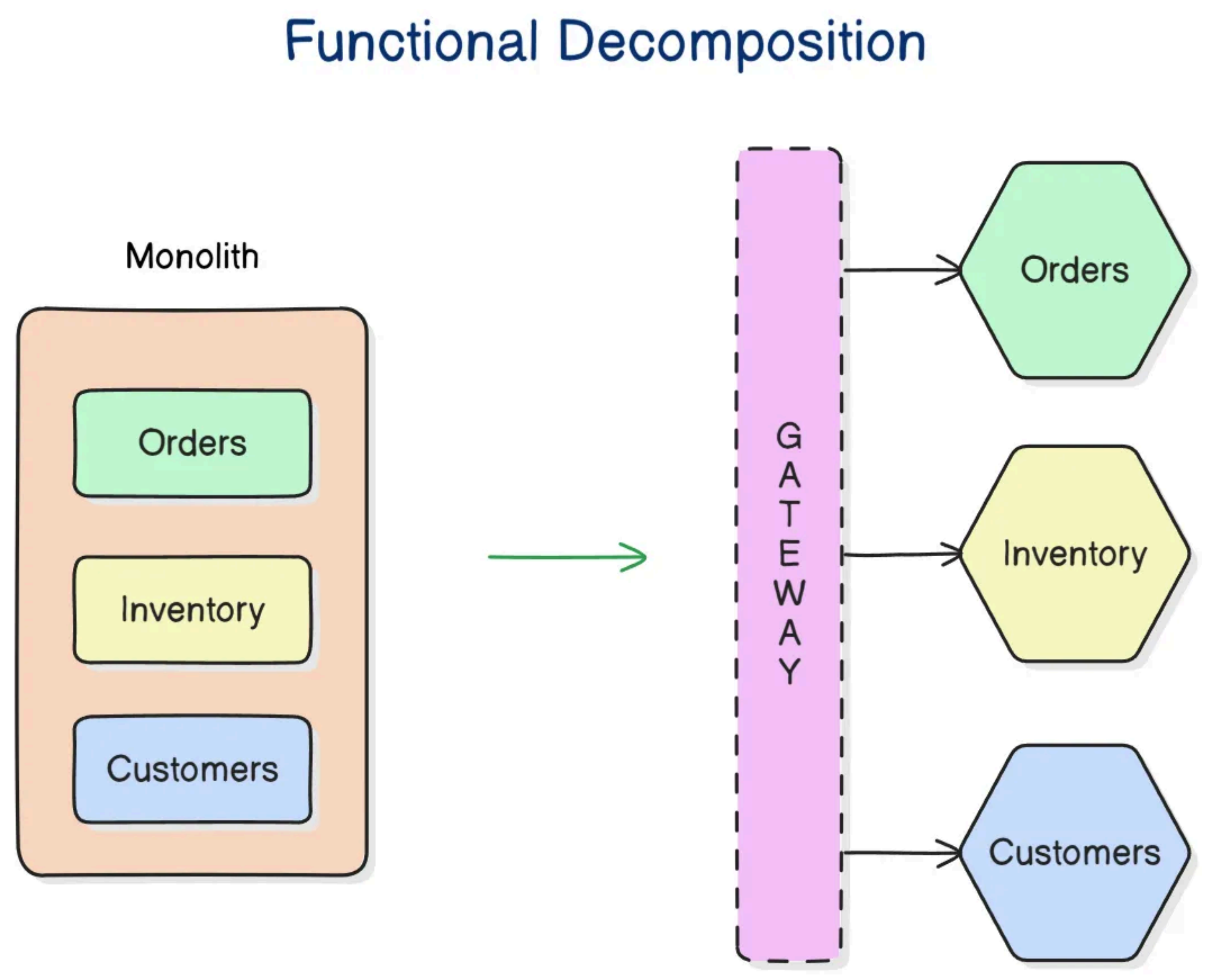
- High scalability by distributing data.
- Efficiently manages resource utilization.

Cons:

- Complex implementation and ongoing maintenance.
- Challenging cross-partition queries.

4. Functional Decomposition (Microservices)

Functional decomposition involves breaking down your monolithic app into smaller, independently deployable microservices, each responsible for a specific business domain (e.g., orders, customers, inventory).



When to use: When your app becomes too complex to maintain or scale, microservices allow independent scaling and easier maintenance.

Pros:

- Maximum scalability and flexibility.
- Independent services can evolve separately.
- Better fault isolation.

Cons:

- Complex architecture that requires careful coordination.
- Data consistency and service communication become harder.
- You will probably need a larger team.

Wrapping up

Start with simpler, low-impact strategies such as vertical scaling and horizontal duplication.

Move to higher-impact strategies like data partitioning and microservices only when simpler solutions no longer meet your needs.

> Architectural evolution isn't negative—it's evidence of your business's growth and maturity. Successful systems naturally evolve.

TL;DR

Remember: If your system never needs scaling, you're probably not growing.

- Begin with simpler solutions.
- Adopt complex methods only when necessary.
- Keep complexity proportional to actual business needs.

Have you already applied these scaling strategies in your own projects?

Articles I enjoyed this week

- How Do Websockets Work by Neo Kim
- REST vs GraphQL by Ashish Pratap Singh
- Use Booleans Like a Pro by Daniel Moka
- Transform Your Codebase to Intuitive Diagrams in a Few Seconds by Saurabh Dashora
- The Shopify Checkout Architecture by Franco Fernando
- How Keeping a Work Log (aka Brag List) by Petar Ivanov

Thank you for reading System Design Classroom. If you like this post, share it with your friends!

Petar Ivanov Mar 18

Liked by Raul Junco

It's always a trade-off between what we can sacrifice and what we can't. In the real world, usually, we can't build the perfect solution.

The only thing that matters is the level of agility and flexibility of the solutions we create.

My go-to approach is to start as simply as possible without sacrificing the flexibility of the architecture to evolve.

Great article, Raul!

LIKE (3) REPLY SHARE

1 reply by Raul Junco

Inayatullah Shinwari Mar 18

Liked by Raul Junco

Very Good Article!

I've got a question: Is horizontal duplication the same as horizontal scaling? If not, why have you mentioned vertical scaling but not horizontal scaling?

LIKE (1) REPLY SHARE

1 reply by Raul Junco

11 more comments...