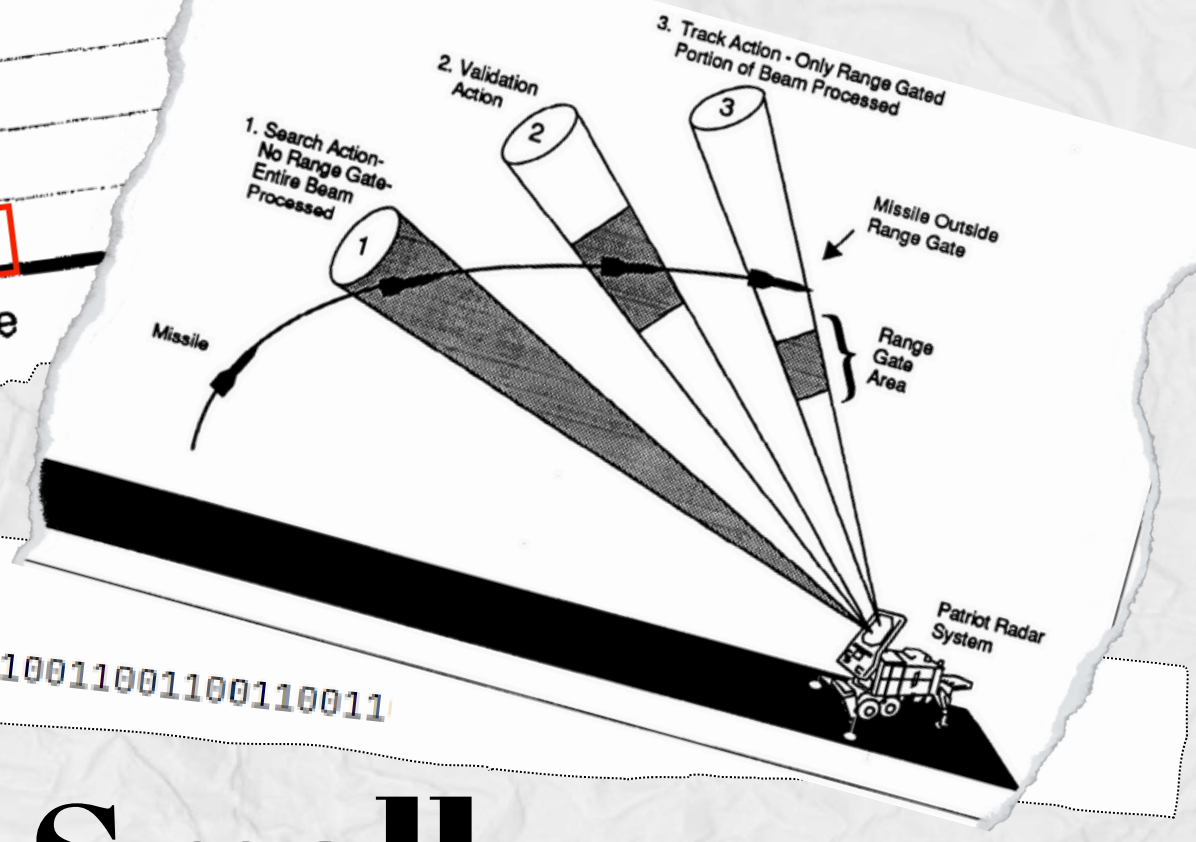
Seconds	Calculated Time (Seconds)	Inaccuracy (Seconds)	Approximate Range Gate (Meters)
0	0	0	0
3600	3599.9966	.0034	7
28800	28799.9725	.0275	55
72000	71999.9313	.0687	
172800	172799.8352	.1648	
259200	259199.7528	.2472	
360000	359999.6667	.3433	

bus operation exceeding about 20 hours—target outside range gate

attery ran continuc

20 hours

```
Float value: 0.1
Binary representation: 00111101110011001100110011
```



How a Small Software Bug Led to a Big Disaster

Why we should use **BigDecimal** instead of **float** and **double** types in **Java** for precise calculations

United States General Accounting Office

Report to the Chairman, Subcommittee on Investigations and Oversight, Committee on Science, Space, and Technology, House of Representatives

PATRIOT MISSILE DEFENSE

Software Problem Led to System Failure at

Post by **Tushig Tsogtbaatar**

01. Introduction

Figure 1. Raytheons GaN-based AESA Radar Prototype

In 1991, **a software bug** in the Patriot missile defense system led to the loss of **28 lives**.

A small **floating-point error** caused this **tragic incident**.

Let's break down how such **tiny issues can have massive consequences.**

Image source: https://commons.wikimedia.org/wiki/File:Raytheons_GaN-based_AESA_Radar_Prototype.jpg

02. What is Floating-Point Arithmetic?

In our decimal system
1 / 3 equals to

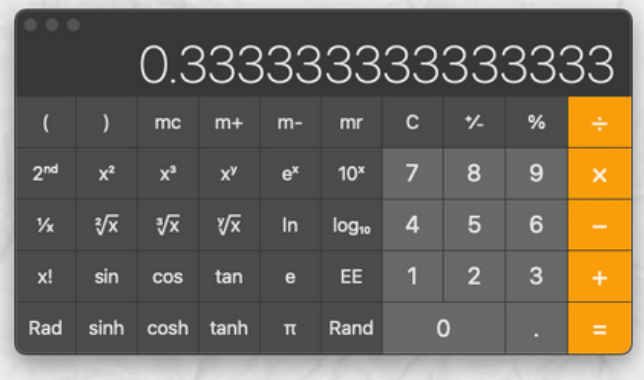


Figure 2. Apple MacOS calculator

In binary system
1 / 10 equals to

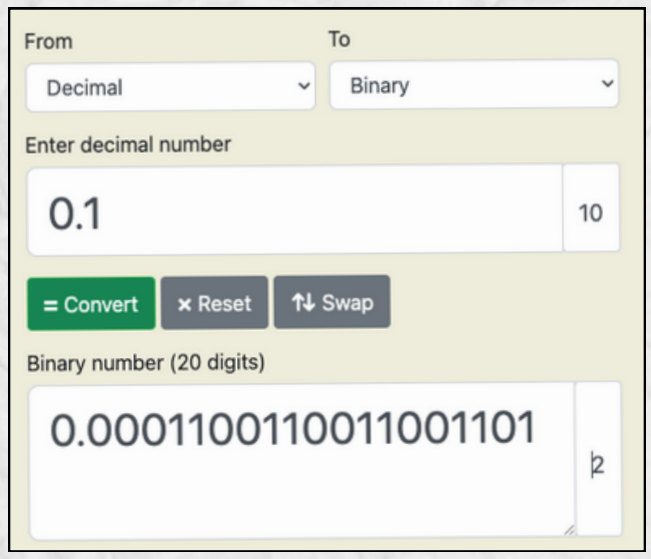


Figure 3. Decimal to Binary calculator (rapidtables.com)

where the digit "3" repeats indefinitely.

where the digits "0011" repeat indefinitely.

A computer uses '**Floating-point Arithmetic'** to handle real numbers **(for this example 0.1).** However, due to **limitations in binary representation (e.g., 32-bit or 64-bit)**, computers can't always store these numbers exactly.

```
System.out.println(new BigDecimal(0.1f));

---------------------------------------------------------------

0.100000001490116119384765625
```

Figure 4. The binary representation of 0.1

Small rounding errors can accumulate over time !

Image source: https://www.rapidtables.com/convert/number/decimal-to-binary.html?x=0.1

03. The Patriot Missile System's Role

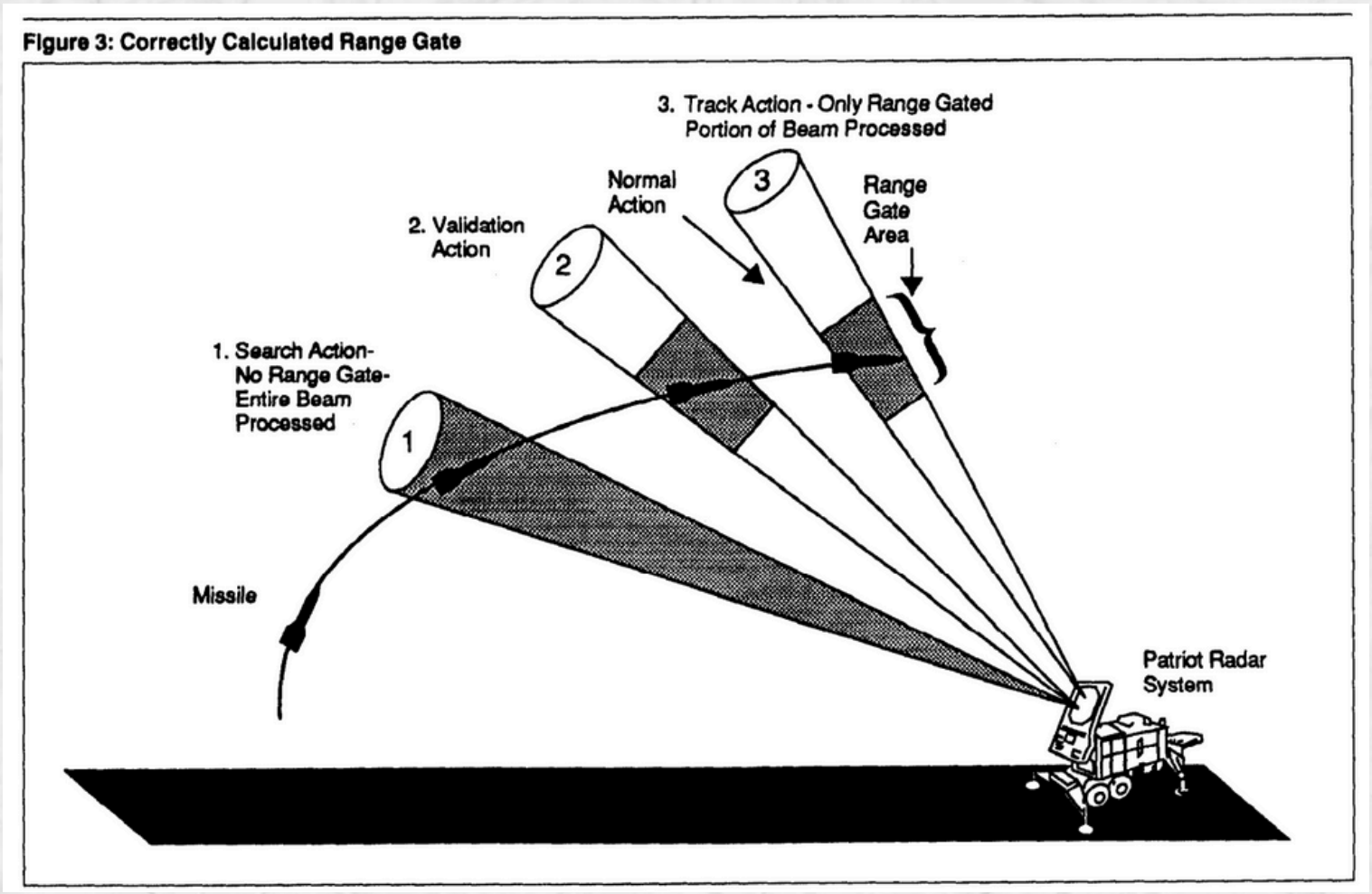


Figure 5. Correctly Calculated Range Gate

The Patriot missile defense system was designed to track and intercept incoming missiles.

It relied on a **tracking algorithm** that continuously calculated the position of incoming targets **based on elapsed time** since tracking began.

This time was tracked using an **internal system clock**, and the calculations depended heavily on **accurate timing**.

Image source: https://www.gao.gov/products/imtec-92-26

04. The Bug: Floating-Point Precision Error

The core of the issue was **a rounding error** caused by the way the **system's clock tracked time**.

The system used a **24-bit** register to keep track of time and the internal clock used **floating-point arithmetic** to calculate time based on increments of **0.1 seconds**.

The error was extremely small per calculation but accumulated over time. After the system had been running for **100 hours**, the **accumulated error amounted to 0.34 seconds**.

Hours	Seconds	Calculated Time (Seconds)	Inaccuracy (Seconds)	Approximate Shift in Range Gate (Meters)
0	0	0	0	0
1	3600	3599.9966	.0034	7
8	28800	28799.9725	.0275	55
20[a]	72000	71999.9313	.0687	137
48	172800	172799.8352	.1648	330
72	259200	259199.7528	.2472	494
100[b]	360000	359999.6667	.3433	687

[a]Continuous operation exceeding about 20 hours—target outside range gate

[b]Alpha Battery ran continuously for about 100 hours

Figure 6. Effect of Extended Run Time on Patriot Operation

Image source: https://www.gao.gov/products/imtec-92-26

05. Why 0.34 Seconds Was Critical

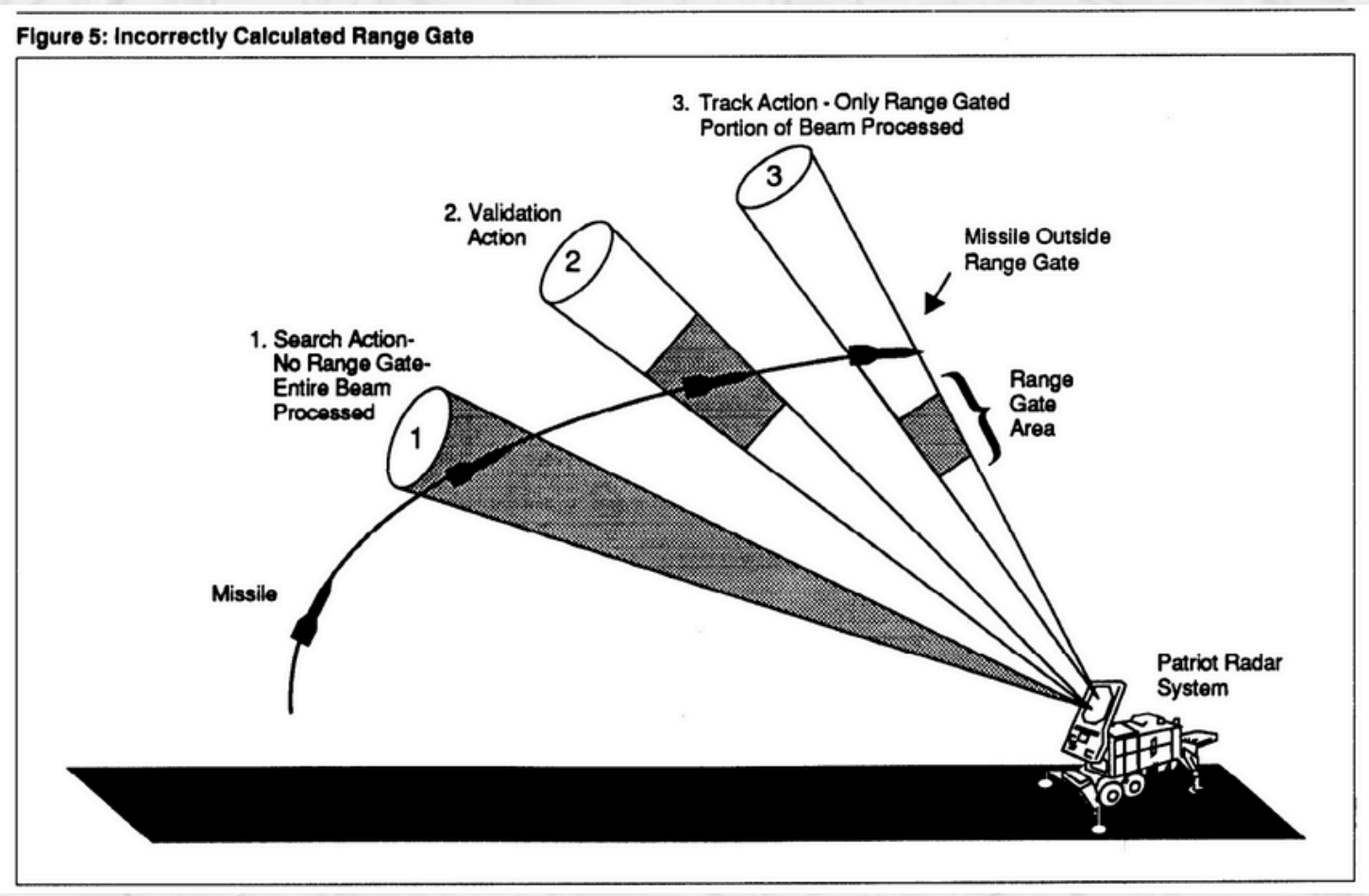


Figure 7. Incorrectly Calculated Range Gate

In a missile defense system, precision is crucial. A **0.34-second** error might not seem significant in many systems, but in the case of the Patriot system, **this was disastrous.**

Missiles like the Scud travel at **very high speeds** (several kilometers per second).

Due to this error, the Patriot system **miscalculated the position** of the incoming Scud missile.

As a result, it failed to engage the missile in time to intercept it.

Image source: https://www.gao.gov/products/imtec-92-26

06. Floating-Point Precision Error In Java

I've tried to simulate the problem in Java using the 'double' type on my 64-bit machine.

Main.java > ...

```java
0 references
public class Main {
    Run | Debug | 0 references
    public static void main(String[] args) {
        double elapsedTime = 0;

        for (int i = 1; i <= 60 * 60 * 100; i++) {
            elapsedTime += 0.1;

            if (i % 10 == 0) {
                System.out.printf(format:"Calculated time after %d seconds: ", i / 10);
                System.out.println(elapsedTime);
            }
        }
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS POSTMAN CONSOLE COMMENT

```
Calculated time after 35971 seconds: 35970.99999975705
Calculated time after 35972 seconds: 35971.999999757034
Calculated time after 35973 seconds: 35972.99999975702
Calculated time after 35974 seconds: 35973.999999757005
Calculated time after 35975 seconds: 35974.99999975699
Calculated time after 35976 seconds: 35975.999999756976
Calculated time after 35977 seconds: 35976.99999975696
Calculated time after 35978 seconds: 35977.99999975695
Calculated time after 35979 seconds: 35978.99999975693
Calculated time after 35980 seconds: 35979.99999975692
Calculated time after 35981 seconds: 35980.9999997569
Calculated time after 35982 seconds: 35981.99999975689
Calculated time after 35983 seconds: 35982.999999756874
Calculated time after 35984 seconds: 35983.99999975686
Calculated time after 35985 seconds: 35984.999999756845
Calculated time after 35986 seconds: 35985.99999975683
Calculated time after 35987 seconds: 35986.999999756816
Calculated time after 35988 seconds: 35987.9999997568
Calculated time after 35989 seconds: 35988.99999975679
Calculated time after 35990 seconds: 35989.99999975677
Calculated time after 35991 seconds: 35990.99999975676
Calculated time after 35992 seconds: 35991.99999975674
Calculated time after 35993 seconds: 35992.99999975673
Calculated time after 35994 seconds: 35993.999999756714
Calculated time after 35995 seconds: 35994.9999997567
Calculated time after 35996 seconds: 35995.999999756685
Calculated time after 35997 seconds: 35996.99999975667
Calculated time after 35998 seconds: 35997.999999756656
Calculated time after 35999 seconds: 35998.99999975664
Calculated time after 36000 seconds: 35999.99999975663
```

Figure 8. A simulation of 0.1 increment in Java

07. Enter BigDecimal for Exact Precision

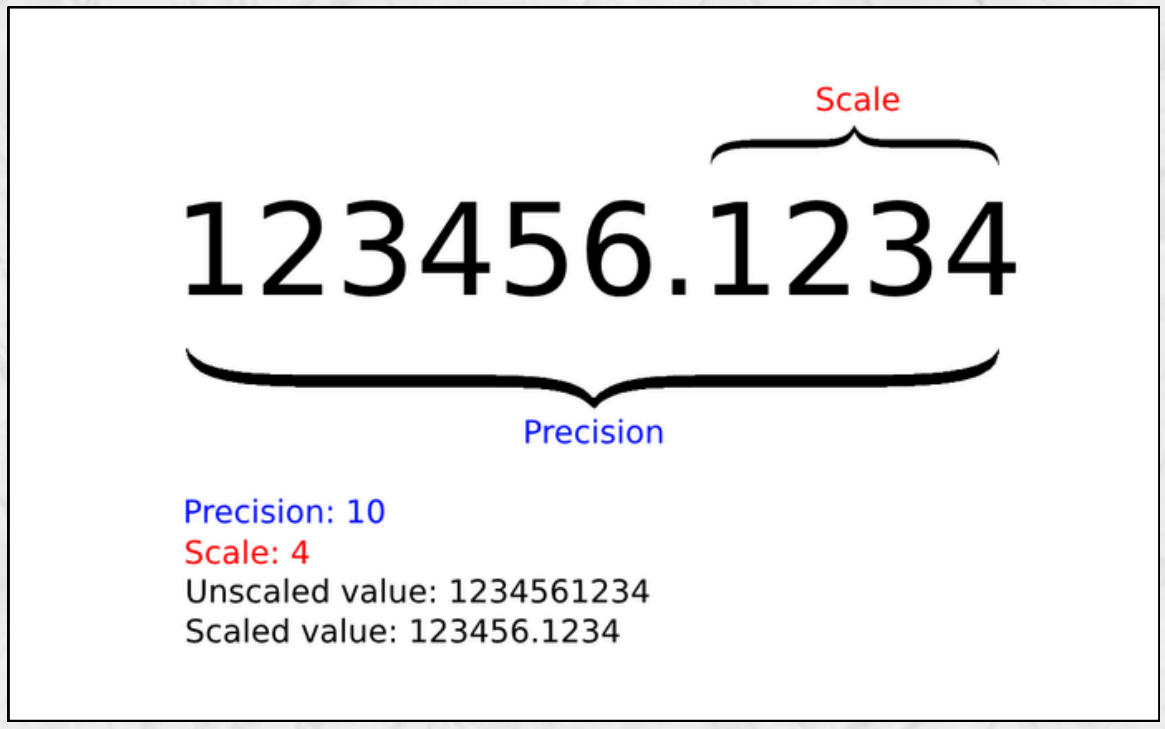


Figure 9. A representation of the BigDecimal saving a value

BigDecimal in Java saves values with **arbitrary precision** by representing numbers in a structured way that **separates the value** (or unscaled value) and the **scale** (the decimal point position).

This approach allows it to handle very large or very small numbers accurately **without being limited to a fixed number of bits (like double or float).**

BigDecimal internally uses **BigInteger** for its **unscaled value**, and **BigInteger** is an **arbitrary-precision** integer, which means **it can represent values as large as the available memory allows.**

08. Other advantages of BigDecimal

```
BigDecimal value1 = new BigDecimal(val:"1");
BigDecimal value2 = new BigDecimal(val:"3");
BigDecimal result = value1.divide(value2, scale:2, RoundingMode.HALF_UP);
System.out.println(result); // Output: 0.33
```

Figure 10. A representation of the BigDecimal rounding a value

In scenarios where **rounding** is necessary (e.g. when dividing numbers with long decimal expansions), **BigDecimal** gives you **full control over how the rounding is done.**

You can specify the rounding mode to ensure that **rounding behaves exactly as required** (e.g., rounding up, rounding down, or rounding half-up).

Also, **BigDecimal** is **immutable**, meaning that once an instance is created, it **cannot be modified**. Every arithmetic operation (addition, subtraction, etc.) **creates a new BigDecimal instance**.

This **immutability** makes the **BigDecimal thread safe** because there's **no risk of modifying the value unexpectedly** across **different threads.**

08. Usage of BigDecimal

Figure 11. Turkish sport shooter Yusuf Dikeç
competing Summer Olympic Games Paris 2024

- **Financial Calculations:** In financial systems, where **exact amounts** of currency must be tracked and manipulated **without rounding errors**.

- **Scientific Calculations:** For cases where **precision** is critical, such as dealing with very small or very large numbers in **simulations or models.**

- **Any Case Requiring Exact Decimal Precision:** Whenever **accuracy** is paramount and **floating-point rounding errors cannot be tolerated**, such as in GPS calculations, measurements, or accounting.

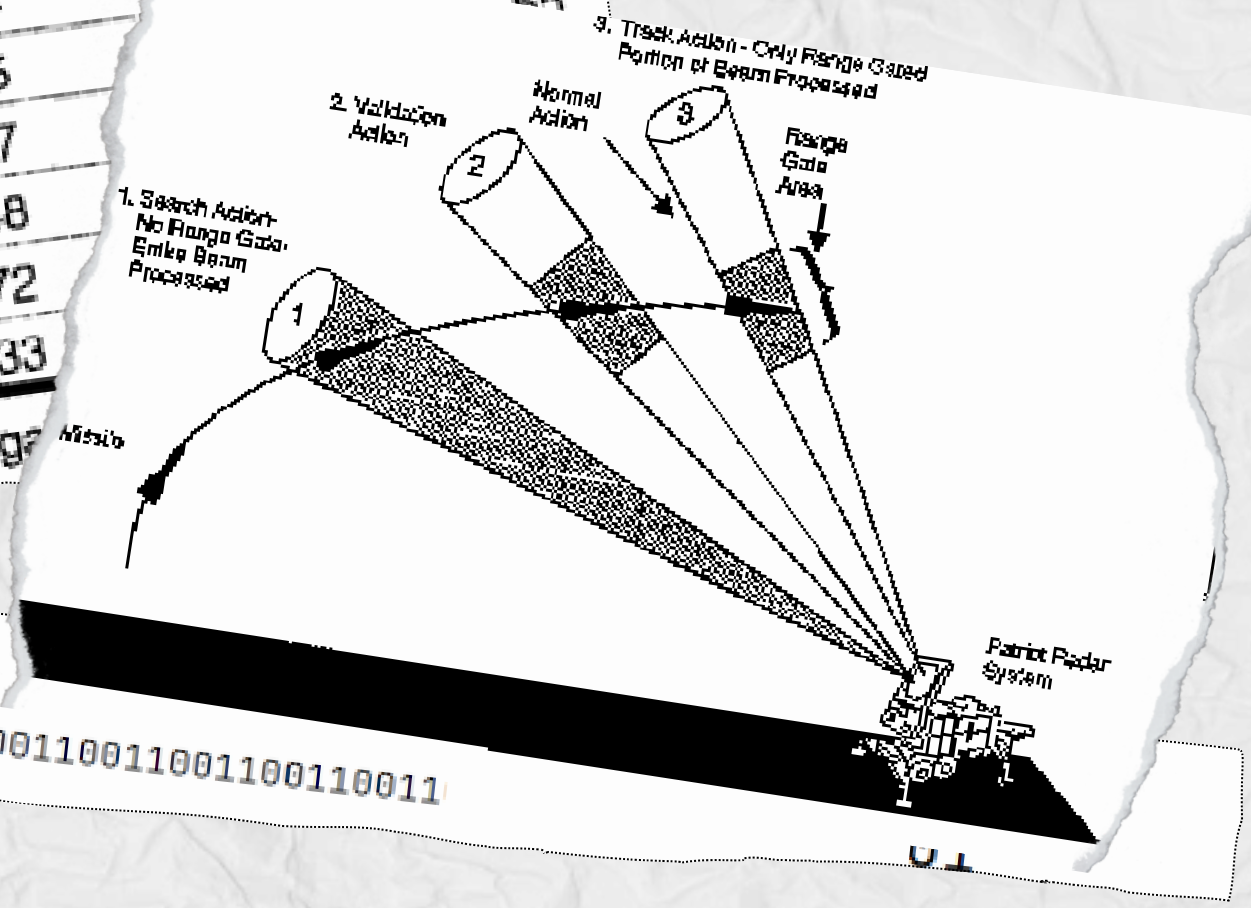
Image source: https://www.bbc.com/sport/olympics/articles/c17g0n15vqwo

Seconds	Calculated Time (Seconds)	Inaccuracy (Seconds)	Approximate Range Gate (Meters)
0	0	0	0
3600	3599.9966	.0034	7
28800	28799.9725	.0275	55
72000	71999.9313	.0687	
172800	172799.8352	.1048	
259200	259199.7528	.2472	
360000	359999.6667	.3433	

us operation exce about 20 hours—target outside range ga

attery ran contin

```
Float value: 0.1
Binary representation: 00111101100110011001100110011
```



Tushig Tsogtbaatar

FOLLOW UP FOR MORE!

Image source: https://www.gao.gov/products/imtec-92-26

Post by **Tushig Tsogtbaatar**