UPI PAYMENT SYSTEM DESIGN

How one QR scan moves money between two banks, and why a crowd does not lose it.

> **Simplified example, not NPCI's actual architecture. It is a teaching design built from general system design ideas. Only the items marked "NPCI" in the facts table come from NPCI.**

1. THE SITUATION

Festival week. A very large number of people scan a QR code at the same moment: the mithai shop, the petrol pump, an online sale. Every payment touches at least two banks and has to finish in a few seconds. Not one rupee may be lost, and not one rupee may be debited twice.

The question: **what happens behind one scan, and why does the money not get lost in the rush?**

2. FACTS USED

Fact	Value	Source
UPI transactions, October 2026	25,466.70 million	NPCI, UPI Product Statistics (monthly)
UPI value, October 2026	₹31,52,929.46 crore	NPCI, UPI Product Statistics (monthly)
Banks live on UPI, October 2026	758	NPCI, UPI Product Statistics (monthly)
Participants	UPI App, Payer PSP, Remitter Bank, Payee PSP, Beneficiary Bank, Users and Merchants	NPCI, About UPI
Two-factor authentication	Mobile number linked to the bank, and the UPI PIN	NPCI, UPI product page
Availability	"24*7 and 365 days"	NPCI, About UPI
Direct bank-to-bank connections for 758 banks	2,86,903	Calculated: 758 × 757 ÷ 2
10,000 payments per second	Assumed	"Maan lo" number for the example, not a real figure

NPCI pages read on 12 November 2026: npci.org.in/product/upi/product-statistics, npci.org.in/product/upi/about-upi, npci.org.in/product/upi. Amounts, UPI IDs and transaction IDs in this document (Rs 500, ramesh@bank, TXN 4821) are examples.

3. REQUIREMENTS

1. Move money from any bank account to any other bank account using only a UPI ID or a QR code.
2. Finish in a few seconds, at any time of day.
3. Never lose money and never debit twice, even when a network call fails halfway.
4. Only the account owner can approve a payment.
5. Keep working on the busiest day of the year.

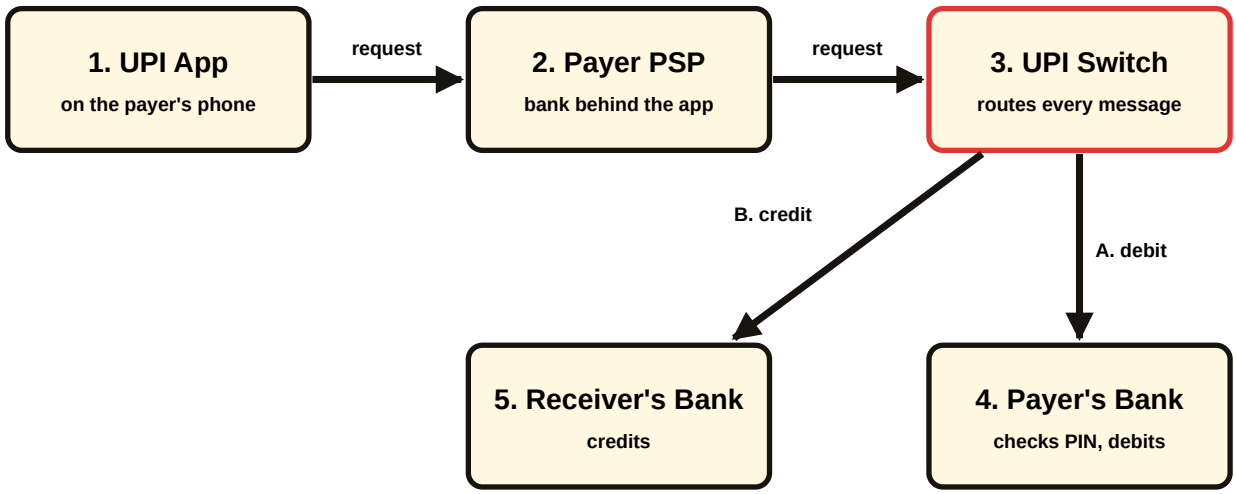
4. THE SIMPLE APPROACH, AND WHERE IT BREAKS

Every bank connects directly to every other bank, and they send each other money.

- 758 banks would need 758 × 757 ÷ 2 = **2,86,903** direct connections to build, test and keep alive.
- Every app would have to know every bank's formats and rules.
- If the receiving bank is slow, the paying bank does not know whether the money arrived. It tries again, and the customer is debited twice.

So the design needs one common middle, one common address format, and a rule for payments that stop halfway.

5. THE FULL DIAGRAM



The request travels left to right. The switch first asks the payer's bank to debit (A), and only then asks the receiver's bank to credit (B). NPCI's participant list also has a Payee PSP, which resolves the receiver's UPI ID; here it is folded into the address lookup in idea 1 to keep the picture to five boxes.

6. DESIGN IDEAS, IN THE ORDER A PAYMENT TRAVELS

Idea 1: A UPI ID instead of an account number

Problem: nobody wants to type an account number and IFSC at a shop, and sharing them is risky.

How: the UPI ID (for example `ramesh@bank`) is a nickname. The part after `@` says which PSP knows the real account. A PSP (payment service provider) is the bank that stands behind an app and connects it to UPI. That PSP looks up the nickname and returns the bank and account behind it. It is like tapping a contact name on your phone: the phone finds the number.

> **Trade-off:** one extra lookup before every payment, and the lookup service must always be up.

Idea 2: One switch in the middle

Problem: 2,86,903 bank-to-bank connections cannot be managed.

How: every bank connects once, to a central switch. The switch reads each message and sends it to the right bank, like a telephone exchange. 758 connections replace 2,86,903.

> **Trade-off:** the switch is now the one place that must never go down, so it runs on many servers in more than one location.

Idea 3: The PIN is checked only by your own bank

Problem: the request passes through an app and a switch. Neither should be able to spend your money.

How: two checks. Your phone and mobile number are registered with your bank (something you have), and the UPI PIN is something you know. In this design the PIN is locked on the phone so that only your own bank can open and check it; the app and the switch carry a sealed envelope they cannot read.

> **Trade-off:** change the phone or SIM and you must register again; a forgotten PIN needs a reset.

Idea 4: Debit first, then credit, with an undo

Problem: two banks mean two separate databases. They cannot both change in one single step.

How: step one, the payer's bank debits Rs 500 and replies "done". Step two, the switch asks the receiver's bank to credit Rs 500. If step two fails, the switch tells the payer's bank to undo the debit. This pattern is often called a saga: a chain of small steps, each with its own undo.

> **Trade-off:** for a short time the money is in neither account. The customer sees "pending", and an undo can take time.

Idea 5: One unique ID for every payment

Problem: the network drops, someone retries, and a bank receives the same request twice. Without care, that is two debits.

How: each payment carries one unique ID (say `TXN 4821`) from start to finish. A bank that has already seen that ID does not debit again; it repeats its earlier answer. This property is called idempotency: doing the same request twice has the same effect as doing it once.

> **Trade-off:** every bank must store the IDs it has seen and check each new request against them.

Idea 6: When a bank is silent, say "pending" and ask again

Problem: on a peak day a bank may not answer in time. The switch does not know whether the credit happened.

How: do not guess. Mark the payment pending, then ask the bank "what happened to this ID?" until there is a clear yes or no. Yes: complete it. No: undo the debit.

> **Trade-off:** the customer waits on an uncertain screen instead of getting an instant answer that might be wrong. With money, slow and correct beats fast and wrong.

Idea 7: Banks settle with each other later, in batches

Problem: if banks moved real money to each other for every small payment, the bank-to-bank system would choke.

How: the customer's balance changes at once, but between banks the switch only keeps a tally. At fixed cycles it nets the tally out ("Bank A owes Bank B this much in total"), and one transfer settles thousands of payments. Friends on a trip do the same: keep a hisaab, settle once at the end.

> **Trade-off:** between cycles, banks are trusting each other for money not yet received, so cycles have to be frequent and closely watched.

Idea 8: Many identical servers behind a load balancer

Problem: festival traffic is far above a normal day.

How: no single server remembers a payment; that state lives in a shared store. So any server can handle any message, and more servers can be added for the peak.

> **Trade-off:** the shared store becomes the hard part, and the banks behind the switch must scale too. The whole chain is only as fast as its slowest bank.

7. THE WHOLE FLOW IN ONE PLACE

Step	What happens	Idea
1	You scan a QR code. The app reads the receiver's UPI ID and creates a unique ID for this payment.	1, 5
2	You enter your PIN. It is locked on the phone for your bank only.	3
3	The app sends the request to its PSP, which sends it to the switch.	2
4	The UPI ID is resolved to the receiver's bank and account.	1
5	The switch asks your bank to check the PIN and debit.	3, 4
6	The switch asks the receiver's bank to credit.	4
7	Credit confirmed: both sides see "success". Credit failed: the debit is undone. No answer: pending, then a status check with the same ID.	4, 5, 6
8	Later, the banks settle the net amount with each other.	7

8. WHAT WAS LEFT OUT

- UPI Lite, autopay mandates, credit on UPI and international payments.
- Collect requests (where the receiver starts the payment).
- Fraud checks, transaction limits and dispute handling.
- Merchant fees and how PSPs and apps are paid.
- Exact timeout values and settlement cycle timings.

9. ANSWER TO THE QUESTION IN THE VIDEO

Question: the debit is done, but no answer came back for the credit. You are the switch. Retry, reverse, or wait?

Answer: none of them blindly.

1. **Ask first.** Send a status check to the receiver's bank for that same transaction ID.
2. **If it says "credited":** finish the payment as a success.
3. **If it says "not credited":** undo the debit.
4. **If it stays silent:** keep the payment pending and ask again. Reverse only once you are sure the credit did not happen; otherwise the receiver keeps the money and the payer gets it back too.

A retry of the credit is safe only because of the unique ID: the receiver's bank will not credit the same ID twice. So the order is **check, then decide**, and the unique ID is what makes every one of these steps

safe to repeat.

Simplified example, not NPCI's actual architecture.