BEN MILNE

The *Value* Layer

A grammar for building software that moves money

Money doesn't get replaced.
It gets connected.

AN INDEPENDENT EDITION · 2026

The *Value* Layer

BEN MILNE

The *Value* Layer

A grammar for building software that moves money

AN INDEPENDENT EDITION

The Value Layer

A grammar for building software
that moves money.

First edition, 2026.
An independent edition.

Drawn from essays published at BENMILNE.COM
between 2021 and 2026, edited and expanded for this edition.

Set in Iowan Old Style and Hoefler Text
with metadata in Menlo. Diagrams set in vector by the author.

benmilne.com

“The limits of my language mean the limits of my world.”

LUDWIG WITTGENSTEIN
TRACTATUS LOGICO-PHILOSOPHICUS

Contents

FOREWORD

Why this book *exists*

This is a short book for people who build software that moves money, and for the machines that are starting to build it with them. It is not a survey and it is not a forecast. It is a mental model: a small grammar that, once you have it, makes most of payments legible, and keeps making sense as the systems underneath it change.

I have spent twenty years inside this problem, and I learned its central claim the expensive way. Early on I set out to kill one of the oldest and most expensive rails in money, and said so out loud, on stages, to rooms full of the people who ran it. I was right about the cost and wrong about the conclusion. You do not get to replace the rails the world is already standing on; you can only raise the bandwidth between them. The builders who compounded were the ones who spoke old and new at once. I had to lose that argument to believe it. The model that survived is not mine; it was sharpened by everyone who used it, broke it, and improved it, and what follows is the version I would hand someone on their first day, or paste into a model before asking it to design a flow.

The expensive thing in this industry is no longer capital. It is attention. The hard part is not the moving; it is holding the whole system in your head long enough to decide well. A grammar gives you a smaller thing to hold, so you have attention left to build.

This is why disruption is the wrong word for what is happening to money. Disruption is a story founders sell to investors; interpolation is what actually happens. New rails do not turn the banks off. They make the banks faster. *Money doesn't get replaced. It gets connected. And the connector gets paid.* The rest of this book is the argument for that sentence.

Three primitives carry the model: *ValueType*, *TransferType*, and *Exchange*. Part I defines them, Part II derives the new from the old, and Part III builds on top.

Ben Milne, 2026

PART ONE

The *Primitives*

Three words that, taken together, describe nearly every system that moves value across the internet. Learn them, and most of payments collapses into a grammar small enough to hold in your head.

CHAPTER ONE

What the value layer *is*

The value layer of the internet is the enormous tangle of systems that lets value move online. Its scale resists intuition: Fedwire alone clears more than a quadrillion dollars a year, and Fedwire is one rail among hundreds. No one holds all of it in their head at once. It has to be reduced, and when the reduction stops, three primitives remain: three words that specify what the system can do.

The reduction is useful the way the line between RAM and a hard drive is useful. Once you have the distinction, you cannot stop seeing it, and most of the apparent complexity falls out as detail.

The three primitives.

> VALUETYPE
>
> *What is moving. A unit of value with an identity that survives a transfer. USD, USDC, BTC, a tokenized treasury, an NFT. Each ValueType names its Issuer.*

> TRANSFERTYPE
>
> *How it moves. A network that debits one place, credits another, and returns a receipt. ACH, wire, SEPA, a card network, Ethereum, Solana, Lightning.*

> EXCHANGE
>
> *What it costs to change one (ValueType, TransferType) pair into another. The only verb among the three, and the place where economics enters.*

That is the whole vocabulary. Three nouns, one of which is secretly a verb. Everything else in this book, the derivations, the physics, the rules, is downstream of these three, and of one observation about them: the interesting differences between systems are not differences of technology. They are differences of cost, speed, and trust. Money has physics, and these primitives are how you read it.

CHAPTER TWO

ValueType

A *ValueType* is what is moving. It is a unit of value with an identity, something a system can pick up in one place, set down in another, and still call by the same name on the far side.

The first instinct is to think in currencies. USD, EUR, GBP, JPY. Correct, but small. A stablecoin is a ValueType: USDC, USDT, PYUSD. So is BTC or ETH. So is a tokenized money-market fund share, or a fractional interest in a building, or an NFT with a quantity of one. The model does not care whether the unit is fungible or unique, government-issued or protocol-issued, on a balance sheet or on a chain. It cares about one thing only: that the unit has an identity that survives the move.

The model leads with ValueType, never with geography, because geography is a fact about people and the value layer is a fact about the internet. The internet runs on protocols. The world runs on laws. A model anchored on the country forces you to rebuild the abstraction at every border. A model anchored on the unit holds still while the user travels.

A few things to notice.

- USD in a wire and USD on a card are the same ValueType.
- USDC on Ethereum and USDC on Solana are the same ValueType, two TransferTypes.
- USD and USDC are different ValueTypes, both worth a dollar.
- BTC and wrapped BTC are not the same ValueType. The wrapper has a new Issuer.

That last line is the one people fight about, so it is worth holding onto. Identity travels with obligation, not with price. When you wrap an asset, you have minted a new ValueType whose Issuer is whoever runs the wrapper. It is worth a dollar, or a bitcoin, only for as long as that Issuer is good for it. The model makes you say so out loud.

The Issuer is inherent.

Every ValueType names its Issuer. USD names the U.S. government. USDC names Circle. USDT names Tether. The unit does not need its Issuer written beside it, because the unit *is* the Issuer, compressed into a symbol.

This is why the Issuer does not get its own primitive. It is not a separate thing you carry. It is a property the ValueType already carries: reserves, redemption terms, compliance posture, and the answer to the only question that matters in a panic, which is *will this still be worth what it says when I try to leave*. Choose your ValueTypes carefully, because everything downstream is inherited from a decision the Issuer made and you did not.

Two dollars that read 1.00 can be different objects. A dollar in an insured bank, a dollar in a money-market fund, and a dollar in a stablecoin reserve are backed by three different Issuers with three different failure modes. The balance is the same. The thing you hold is not.

The case of Bitcoin.

Bitcoin is the cleanest ValueType in the catalog, and the one most people mis-file. Its Issuer is not a company and not a state. Its Issuer is a protocol. There is no one to call, no reserve to audit, no compliance desk. The rules are the issuer, and the rules are explicit, fixed, and visible to anyone. For a unit of value that is enough, and it is rare. Most ValueTypes ask you to trust a party. Bitcoin asks you to trust a process. It is the proof that the Issuer can be code, and it is the deepest pool of permissionless liquidity on the planet, which is a fact that becomes important the moment a stablecoin on the same TransferType can reach it.

"Identity travels with obligation,
not with price."

CHAPTER THREE

TransferType

A *TransferType* is how value moves. It is a network that knows how to debit one place, credit another, and hand back a receipt.

The bank rails are the obvious entries: ACH, wires, SEPA, FedNow, RTP, Faster Payments, Pix, UPI. Card networks are TransferTypes too, with their own strange settlement geometry. So are blockchains: Bitcoin, Ethereum, Solana, Base, Stellar, Tron. So are the interop protocols that move value between chains. So are the closed loops, where money inside an app moves only inside that app.

A note on the word. The industry says *rail*; this book says *TransferType*, and the choice is deliberate. A payment rail is a TransferType, but so is a blockchain, an interop protocol, a closed loop — things nobody would call a rail. The smaller word hides the family resemblance the model exists to expose, and the limits of the vocabulary become the limits of the design. Where *rail* appears in these pages, it means the bank rails the industry named first.

What makes them the same kind of thing is not their technology. It is their job: route value, clear it, settle it, confirm it. They differ on eight dimensions, and those eight dimensions are the entire interesting surface of the layer. If you can score a TransferType on these, you understand it well enough to build on it.

The eight questions to ask of any TransferType.

- Routing, how the destination is named.
- Clearing, when balances tentatively move.
- Settlement, when finality is real.
- Reversibility, what can claw it back, how long.
- Hours, when it is open.
- Cost, fixed and variable, spread included.
- Bandwidth, transactions per second it absorbs.
- Trust, who you must believe for it to be safe.

A blockchain looks nothing like ACH on the wire, but the questions you ask of it are the questions the Federal Reserve asks of its own rails. The answer space fits on one page, which is the next page.

CHAPTER THREE · CONTINUED

A SCORECARD OF REPRESENTATIVE TRANSFERTYPES

TRANSFERTYPE	SETTLEMENT	HOURS	REVERSIBLE	COST	TRUST
`ACH`	1–2 days	banking	yes, days	cents	banks, Fed
`Wire`	same day	banking	no	$15–50	banks, Fed
`Card`	1–3 days	24/7 auth	yes, chargeback	~2–3%	network, banks
`RTP / FedNow`	seconds	24/7	no	cents	banks, Fed
`Ethereum`	~12s, economic	24/7	no	gas, variable	protocol
`Solana`	sub-second	24/7	no	fractions of a cent	protocol
`Lightning`	instant	24/7	no	negligible	protocol, channels
`Privacy protocol`	seconds	24/7	no	low	protocol, permissioned
`Closed loop`	instant, internal	24/7	operator's call	operator's call	the operator

Read the table as a map, not a ranking. There is no best TransferType. There is one that fits a job. A payroll file at 2pm on a Tuesday wants ACH. A house closing wants a wire. A merchant payout that must clear before a refund window wants something that settles in seconds and does not reverse. The old payments engineer chose between ACH and wire by amount and hour of day. The new one chooses between TransferTypes by finality and liquidity. Different compute, same job.

The reason to score TransferTypes this way, rather than argue about which technology is superior, is that it makes routing a decision instead of a religion. *Where should this settle* has an answer, and the answer is in the table.

CHAPTER FOUR

Exchange

An *Exchange* is the conversion from one (ValueType, TransferType) pair to another. It is the only primitive that is not a noun in the system. It is a verb, and it has a price.

A wire of USD turned into USDC on Solana and sent to a wallet is an Exchange. So is a card authorization that ends as a bank credit to a merchant. So is an FX desk printing EUR for USD, or an automated market maker swapping one token for another in a pool. The mechanics differ wildly. The role does not. Every Exchange takes one leg on the input side and emits another leg on the output side, and charges you for the privilege.

The price is where the money is. Spread, fee, slippage, float, the cost of capital tied up while a slow TransferType settles. These are the tariffs of the value layer, and most of the industry's historical profit has lived inside them, bundled invisibly into a headline that said the transfer was free. It was never free. For most of this industry's history the Exchange *was* the business: rent collected on a delay you were never shown.

Why it earns a name.
Once Exchange is a primitive, two things happen. Every funds flow becomes legible: instead of "the bank handles it," you can point at the exact conversions in the path and ask who is paying for each. And design choices that used to feel like magic, whether a product holds inventory, prefunds, routes to a market, or settles lazily, become explicit, which means reviewable, which means improvable.

The stakes are not hygiene. An Exchange that is mispriced, or hidden, or booked wrong is how balance sheets flip — companies have died of conversions they did not know they were running. The primitive earns its name the way most primitives do: by what it costs when it goes unnamed.

"Most transfers that look free
are an Exchange you did not see."

CHAPTER FIVE

The grammar, and its *edge*

The three primitives compose into a grammar. Every leg of every funds flow is a tuple, and a flow is a sequence of those tuples joined by Exchanges. That is the entire language. Written formally, so a person or a program can hold the same thing:

```
ValueType     a unit of value with a persistent identity
              (every ValueType names its Issuer)

TransferType  a network that moves a ValueType and
              returns a receipt

Amount        a quantity of a ValueType

Leg           ( ValueType, TransferType, Amount )

Exchange      Leg_in  ->  Leg_out      with Rate and Cost

Flow          Leg ( Exchange Leg )*
```

That is six lines, and it is the whole model. *Send USD by wire to an issuer, who exchanges it to USDC on Solana, and forwards USDC on Solana to a wallet* is a Flow: three Legs, one Exchange. Drawn at any level of detail, the payment systems you meet in production reduce to Legs and Exchanges. If your ledger speaks this tuple, every new ValueType you support is one row of configuration, not a rewrite.

FIG. 1 · **ONE LEG.** THE ISSUER RIDES INSIDE THE VALUETYPE.

CHAPTER FIVE · CONTINUED

Why three, and not four.

A grammar this small invites a fair suspicion: that something is missing. The test is to nominate a fourth primitive and watch what happens to it.

Nominate the *Issuer*, and it collapses into ValueType; the unit is the Issuer compressed into a symbol, as Chapter 2 showed. Nominate *Identity*, and it is already there: identity is the obligation a ValueType carries. Nominate *Time*, and it dissolves into properties of the TransferType: hours, finality, settlement windows. Nominate *Price* or *Fee*, and you are standing inside Exchange. Nominate *Access*, and it falls off the edge of the model entirely, which is where it belongs and where the next section puts it. Every candidate is either a property of one of the three or a fact about the world outside them. Run the test in the other direction and it holds as well: name a payment system, a card network, a money order, a mobile-money float, an automated market maker, and it reduces to Legs and Exchanges with nothing left over. Three is not a choice. It is what remains when the reduction stops.

The narrow waist.

The internet ran this experiment first. Its architecture is an hourglass: unbounded variety of applications above, unbounded variety of wires and radios below, and at the neck a single minimal agreement, the packet, that everything must speak and no one may complicate. The neck is narrow on purpose. Whatever lives at the waist of a network is carried by every participant, so it must be small enough that everyone can afford to carry it. The Leg is the narrow waist of the value layer: above it, any product; below it, any TransferType; at the neck, one tuple. Keep the waist narrow and the layer outlives every company built on it.

"Three is not a choice. It is what remains when the reduction stops."

CHAPTER FIVE · CONTINUED

The edge: where Access lives.

A Leg has no person in it. That is on purpose, and it answers an obvious question: if Access matters, why is it not a primitive?

Access is who is allowed to author a Leg. The bank, the fintech, the wallet, the API client. It does not appear inside the tuple because it is not part of what moves. It sits at the edge, where the grammar meets the world, and where two things you cannot ignore live. Geography and regulation attach to Access, not to the ValueType: a wallet is the same wallet everywhere on Earth; a regulated entity is not. And the human only exists at the edge: ValueTypes are abstractions, TransferTypes are infrastructure, the person or the program is always at Access. The internet settled this argument decades ago, keep the network simple and put the intelligence at the edges, and the value layer inherits the ruling. If the edge is hostile, the rest of the layer might as well not be there.

Failure is part of the grammar.

A flow that only describes success is a lie, because money systems fail mid-flight. So the grammar carries failure as a first-class state. Any Leg can be *pending*, *settled*, or *failed*, and an Exchange can complete its input leg and fail its output, leaving value somewhere real and unwelcome. A correct design names the holding point for every partial state: if it stops here, who holds the value, in what ValueType, on what TransferType, and what moves it next. Most production incidents in payments are an unnamed in-between. Every operator has the same night in their past: a flow stops at two in the morning, and the money is not lost but not anywhere either. It sits in a suspense account nobody named, waiting for a human to wake up and claim it. The grammar cannot prevent that night, but it makes you name the account before it happens.

```
LegState   pending | settled | failed
Flow       must name a holder for every partial state
            ( ValueType, TransferType ) at each stopping point
```

That is the model, end to end. The rest of the book is what you can see once you hold it.

PART ONE · CLOSING

The three laws of the *layer*

Everything in this book is one of these three statements or a consequence of one. They are descriptive, not advisory: laws of how the layer behaves, true before anyone wrote them down. The rules in Chapter 15 tell builders what to do. These say what is.

I · Identity travels with obligation, not with price.
A unit of value is the promise behind it, compressed into a symbol. Wrap it, bridge it, rename it, and the price may hold while the promise changes hands entirely. Every counterfeit in financial history, paper or digital, is a violation of this law discovered too late.

II · Everything is a transfer.
There are no bridges, no ramps, no crossings. Those are metaphors of distance, and there is no distance. Value is picked up on one TransferType and set down on another, carrying what makes it legible to both sides. A system designed around the metaphor builds tollbooths. A system designed around the law builds throughput.

III · Money is never replaced, only connected. And the connector gets paid.
No TransferType in monetary history has turned another off. Each one raised the bandwidth between what existed and what arrived, and the value accrued to whoever spoke both sides fluently. Disruption is a story for raising money. Interpolation is what the money actually does.

PART TWO

II

Derivations

A derivation is not an invention. It is an evolution, a familiar primitive re-emerging in a new technical context, reshaped by what is now possible. The interesting motion in the value layer is almost always derivative, and almost never replacement.

CHAPTER SIX

A derivation is not an *invention*

A derivation is not a new invention. It is a familiar idea returning in a new context, reshaped by what is now possible. The financial system is built almost entirely of them, and most of what gets sold as disruption is, on inspection, a derivation wearing a louder coat.

The first ACH networks went live in the early 1970s. Merkle trees were invented in 1979, and their use in distributed networks was not imagined for decades. Fast forward, and Merkle trees are the spine of every blockchain, which are themselves derivations of traditional settlement networks with different bandwidth and different trust. Different compute, same job.

The claim does two jobs, and the difference matters. Backward, it is plain fact: most TransferTypes in use today extend from the last generation's. Forward, it is a bet: the next important thing will be a grand derivation, not a true invention. A bet, not a law, but it has not lost in fifty years. The model gives you the tell. A *grand* derivation flips an axis of the physics, cost or latency or bandwidth or trust. A thin one just renames the old thing. Flip an axis and the possibility space changes shape. Rename it and nothing happens but the marketing.

Three derivations are reshaping the layer now.

- **Stablecoins** derive the cash equivalent. A new ValueType.
- **Protocols** derive the clearing network. A new TransferType.
- **Tokenized fiat** derives the bank rail. The old system moving onto the new TransferTypes.

The next chapters take each in turn. None of them is a prediction. They are descriptions of motion already well underway. Naming them as derivations is the point, because once you stop seeing them as novelties, the right thing to build on top of each becomes obvious.

CHAPTER SEVEN

Stablecoins as *liquidity*

Stablecoins are the new cash equivalents: accessible, digital, programmable. They are a derivation of an old role, the ValueType that sits next to operating cash, ready to move on demand at the lowest possible cost of motion.

In the last version of this role, same-day ACH and wire did the work. A treasury kept a buffer at a bank and pushed it out when needed. Stablecoins play the same part with three differences that only matter at scale, and matter enormously. They do not stop at a border. They do not keep banking hours. And the cost of moving them is closer to the cost of a database write than the cost of a payment.

The result is liquidity that did not previously exist. USDC and USDT move globally, instantly, in volumes the legacy cash-equivalent stack cannot reach. And because they live on the same TransferTypes as everything else onchain, they can touch the deepest permissionless liquidity on the planet directly. A stablecoin contract can reach Bitcoin without a bank in the middle, which has never been true of any cash equivalent before. That is not a feature. That is a new shape.

What changes for builders.

Float is no longer the engine. In the old world the friction of moving money funded the system; overnight balances, settlement delays, FX spread tucked inside a flow. Drop the friction toward zero and all of that reprices. Designs that lived on captive float get repriced against them. Designs that compose cheap liquidity at the edges of an existing product get repriced for them. The honest summary is small: stablecoins did not invent the cash equivalent. They derived one whose physics, global, instant, always on, nearly free, make a class of products possible that could not previously pay for themselves. The float that funded the old world is the margin the new one competes away.

CHAPTER EIGHT

Protocols as *transfer*

Blockchains, and the interop protocols that connect them, are not just payment rails. They are TransferTypes in the strict sense of the model, but of an unusually general kind: routers that can carry value, state, or instructions for any purpose, not only money.

What they borrow from the old world is the question set: routing, clearing, finality, reversibility, hours, cost. What they change is the answers. Routing is by address, not routing number. Clearing and settlement collapse into one event. Finality is probabilistic for a few blocks and economic thereafter. Hours are all of them. Reversibility is defined by a contract, not a policy. Cost is per byte and per unit of compute, not per payment.

The distinction worth keeping is that a protocol is not one TransferType. It is a factory for them. Ethereum is a TransferType. Base is another. So is the path between them. The fact that they share a virtual machine is a detail of engineering, not a fact about the model. From the layer's point of view, every distinct settlement domain is a distinct TransferType, and you route across them the way you always routed: by cost, latency, liquidity, and who you have to trust.

Why it matters at design time.
Treating each protocol as its own TransferType turns a tribal argument into a routing decision. *Should this settle here or there* is not about which chain you like. It is a question with an answer in the scorecard. The engineer who internalizes that ships across five TransferTypes without ever having an opinion about any of them, which is exactly the right number of opinions to have about one.

CHAPTER NINE

Legacy rails going *onchain*

Every fiat currency and bank rail already fits the model. The third derivation is not a new ValueType or a new TransferType. It is the old ones migrating onto the new TransferTypes. Tokenized fiat, tokenized treasuries, tokenized funds. The legacy stack putting itself onchain.

This is not a bridge, and the word matters. The industry leans on *bridge* and *on-ramp*, but those are metaphors of distance, and there is no distance here. When you move a file between folders, you do not bridge it. You move it. A log records where it went, and that is how the machine knows where to show it. A well-designed tokenized dollar works the same way: a value packet is picked up on one TransferType and set down on another, carrying the metadata that makes it legible to both sides. It is a transfer. Everything is a transfer; that is the second law doing its work.

What you get when fiat goes onchain.

- Bandwidth that scales with software, not banking hours.
- Settlement with finality in seconds.
- Global reach without a correspondent in every country.
- Programmability; every transfer can carry instructions.
- Composability with liquidity already on the TransferType.

None of this turns the old stack off. A tokenized dollar still sits on a bank core somewhere, and a stablecoin can settle back into that core without drama. The order never changes: new ValueType, then new TransferType, then the legacy onto both. The cores do not go dark as the bandwidth of money rises. Only the ceiling moves. *That is not disruption. It is interpolation.*

CHAPTER TEN

A worked example: settlement, *rewired*

Theory is cheap. Here is the model doing real work, on the most familiar money movement there is: a card payment. No logos, no names, just funds flows and the grammar from Part I. Watch what the model lets you see, and what it lets you change.

You tap a card. In a couple of seconds it is approved. From where you stand the money moved. It did not. What moved was a message. Authorization is the message; settlement is the money, and settlement happens later, in windows, in batches, with real liquidity trapped in between.

Write the settlement as a Flow and the trapped liquidity becomes visible. Each arrow below is a Leg in the grammar, every party is at the edge, and the value type on every leg is the same tired dollar on the same slow rail.

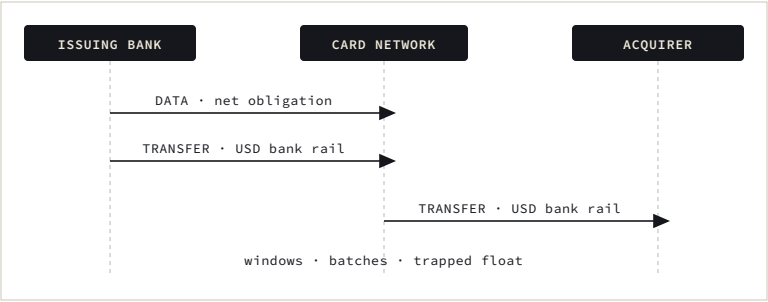

FIG. 2 · **FIAT SETTLEMENT AS A FLOW.** ONE VALUETYPE, ONE SLOW RAIL, LIQUIDITY STUCK BETWEEN THE LEGS.

This works, at enormous scale, across every country and bank on Earth. It is easy to forget how miraculous that is. It is also not instant, and the cost of making it feel instant has always been the same: put more capital in the middle. You can make money feel faster by trapping more money. The grammar shows you exactly where it is trapped, which is the first step to a different answer.

Change one Leg.

The model says the abstraction does not have to break for the people at the edge. The cardholder still taps. The merchant still gets paid. We are only going to rewire the settlement Leg, the slow one in the middle, and leave everything the user touches alone.

Replace the bank-rail Leg between the network and the acquirer with a stablecoin Leg on Canton, a permissioned settlement protocol that settles in seconds, runs around the clock, and shows each transaction only to the parties entitled to see it. Card settlement is not supposed to broadcast counterparty data, which is why a public chain is the wrong TransferType here and a permissioned one like Canton is right. The grammar does not care which; it cares only that the TransferType and its trust are named. In the model, the change is one substitution.

```
// settlement leg, before:
{ "value_type": "USD",    "transfer_type": "bank_rail" }
// settlement leg, after:
{ "value_type": "STABLE", "transfer_type": "canton" }

// the call is the same call either way
POST /transfers
{
  "amount":      { "value": "4250000.00" },
  "source":      { "value_type": "USD",    "transfer_type": "wire" },
  "destination": { "value_type": "STABLE", "transfer_type": "canton" }
}
```

Same endpoint, same shape. The value type and the transfer type change, the mechanics change underneath, and the integration does not. That is the point of a grammar: legs are interchangeable, so a rebuild becomes a configuration. And the shape that makes the swap easy makes the reckless one hard, because you cannot write a Leg without naming its TransferType and its trust, so the model will not let you move value while hiding what you rely on.

What did rewiring one Leg buy? Settlement in seconds instead of days, liquidity untrapped, privacy preserved. The network still coordinates. The banks still stand behind their obligations. The cores stay on. The ceiling moved.

PART THREE

III

Building

The model exists to make building easier. This part is the physics that decides what software can do with money, the conventions that earn their keep, the places the model breaks, and a short set of rules for the people designing on top of the layer.

CHAPTER ELEVEN

The *physics* of money

Money has physics. Three constraints, cost, latency, and bandwidth, define a possibility space: the set of tasks money can be made to perform. Every TransferType is a point in that space. Every grand derivation is a move through it. Change a constraint by enough orders of magnitude and the space changes shape, and tasks that were impossible become merely engineering.

Plot the TransferTypes and the structure jumps out. The old bank rails sit up and to the right: slow, and not cheap once you count the float. The protocols sit down and to the left: fast and nearly free. The whole story of this era is TransferTypes migrating toward the bottom-left corner, and dragging what they carry with them.

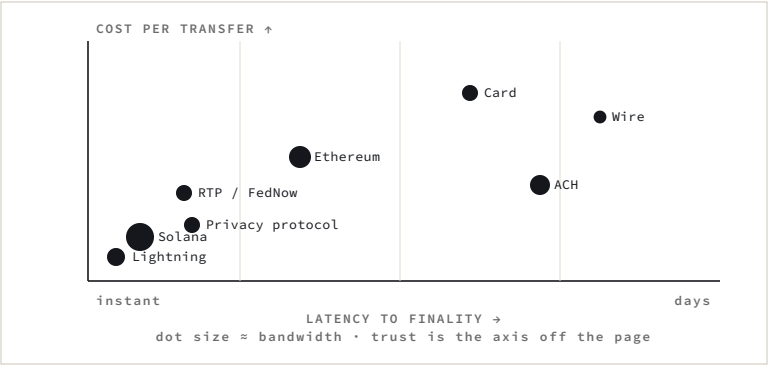

FIG. 3 · **THE POSSIBILITY SPACE.** TRANSFERTYPES ARE MIGRATING TOWARD INSTANT AND FREE.

Trust is the axis that does not fit on a flat page, and it is the one that decides everything in a crisis. A cheap, instant, high-bandwidth TransferType you cannot trust is not on the map at all. Hold all four in mind and the map is honest.

CHAPTER ELEVEN · CONTINUED

A nine-order-of-magnitude collapse.

FIG. 4 · **THE COST TO ISSUE A STABLECOIN,** THEN AND NOW.

In the span of a few years the constraints did not improve. They dissolved. The cost to launch a stablecoin went from roughly a hundred million dollars to roughly a dollar. The cost of a transaction onchain fell below the fee on a Federal Reserve master account. Protocols erased banking hours.

The skeptic's objection answers itself. Yes, the Fed's own rail was always sub-cent — at a price sheet few institutions ever get to see, after years of trying. The old cheapness was gated; the new cheapness is permissionless. The collapse that matters is not the cost of the transfer. It is the cost of being allowed to make one.

Read it the way a physicist reads a system: by what it makes possible, not by what it is. When the barrier to issue value was a hundred million dollars, only the largest institutions could denominate and move their own unit. At a dollar and a few minutes, that capability is an API call. That is the test of a real change: not that the old thing got cheaper, but that something previously impossible is now ordinary. A counterfactual became a fact.

Most of the rest of this book is downstream of that single collapse. When a primitive falls by nine orders of magnitude, the correct response is not to do the old thing for less. It is to map the new possibility space and build the things only that space allows.

"Read a system by what it makes possible, not by what it is."

CHAPTER ELEVEN · CONTINUED

The model does not know where you are.

The axes never mentioned Earth. Cost, latency, bandwidth, and trust are properties of a TransferType, not a planet. Geography only ever lived at the edge, on Access — so nothing in the grammar changes when value leaves the ground.

Distance is not a new dimension. It is just latency, and latency is a number: light to Mars runs three to twenty-two minutes each way, so a confirmation round trip is the better part of an hour. On the map that TransferType is one point pushed hard to the right, and the grammar does not strain. Anything synchronous dies at that gap; what survives is value that carries its own rules and settles locally, a ValueType whose Issuer is a protocol rather than a party you must phone. The designs that already win on hostile-latency TransferTypes on Earth are the ones that work between worlds. We did not add a chapter for Mars. The physics did it for us.

“Distance is not a new dimension.
It is just latency.”

CHAPTER TWELVE

Ecosystems beat *verticals*

The old fintech playbook was verticalization: build one horizontal technology, then tailor it vertical by vertical, B2B then payroll then retail. It worked, and a generation of good companies came out of it. In the world of protocols the playbook inverts, and the first question is no longer which vertical. It is which ecosystem.

Each TransferType has a center of gravity that pulls in particular applications, particular liquidity, particular builders. Solve for the ecosystem first and the verticals follow, because anything good for the ecosystem tends to be good for everything built on it.

Welcoming is the moat.

The derivation is short. Every ecosystem grows by one motion: someone authors a new Leg on its TransferType. Welcome is whatever lowers the cost of that motion, the docs, the sample code, the person who answers in the chat within the hour. Lower the cost of the first Leg and you get more Legs. More Legs are liquidity. Liquidity is gravity, and gravity pulls in the next builder at a still lower cost. An integration that once took a year of business development runs in an afternoon, and the difference is not technology. That is not hospitality. It is compounding.

The old network effects were bilateral: a buyer and a seller, every new participant a cold start. Ecosystem effects are cooperative: a builder inherits reach from infrastructure it did not build, and the infrastructure inherits volume from builders it never recruited. An open standard compounds the same way: each builder's knowledge lowers the cost for the next. That is a knowledge loop, and the value layer itself is one: each implementation teaches the next. A loop, once it is turning, is very hard to catch.

The instruction is embarrassingly simple. Prefer the welcoming ecosystem. Contribute to the standard rather than fork it. The moat is not what you keep closed. It is the loop you keep turning.

CHAPTER THIRTEEN

A *common* format for funds flows

Funds flows are drawn every day, by everyone in this industry, and everyone draws them differently. The drift is not cosmetic: a diagram nobody else can read is a design nobody else can review. The fix is a small standard, a fixed set of rules handed to a person or a language model, so that neither has to invent a format and both produce the same drawing from the same flow.

The grammar is the one from Part I, made visible on the page. ValueType, TransferType, Amount, on every leg. USD Wire 100. USDC Solana 100. A stablecoin on Canton, 4,250,000. Like a circuit schematic, a funds flow becomes legible the instant its labels are consistent, and illegible the instant they are not. The standard has three depths, light for a sketch, medium for a compliance walkthrough, heavy for error and edge handling, and it is open, because a format only becomes a standard if no one owns it.

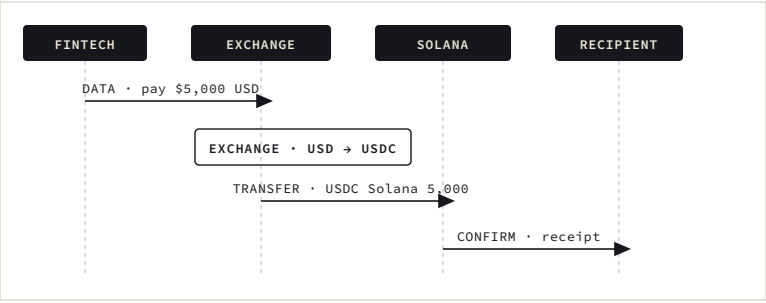

FIG. 5 · **A FLOW IN THE COMMON FORMAT.** USD IN, USDC OUT, ONE EXCHANGE.

The format is the quiet part of the strategy, and the durable part. Layers win when their format is boring and universal, the way HTTP and DNS won. If a grammar is small enough that a builder, or a model, can output a correct flow on the first try, it spreads on its own. The format is the moat precisely because it is given away.

CHAPTER FOURTEEN

Where the model *breaks*

A model you cannot break is not a model, it is a belief. Here are the edges, the places this grammar strains or fails. If you are going to trust it in production, you should know where it stops being trustworthy.

Identity is a claim, not a guarantee.
The model assumes identity survives transfer, and wrapping puts that under load. The instant you wrap an asset you have a new ValueType with a new Issuer; forget that, and the model will let you treat two different obligations as one. It protects you only if you are honest about when identity changed hands. It will not catch the lie for you.

Issuers degrade quietly.
A ValueType inherits everything from its Issuer, including the Issuer getting worse. Reserves thin, compliance lapses, redemption freezes, none of it changing the symbol on the unit. I have watched it happen. One weekend a stablecoin I relied on traded at eighty-seven cents while every screen still printed 1.00. The symbol had not moved; the reserves behind it had. The ones who came through intact had written down, in advance, whose promise they were actually holding. The model tells you to choose Issuers carefully; it cannot monitor them for you. The worst failures wear an unchanged label.

The model is descriptive, not predictive.
The grammar tells you what a system is and what it can do, not what will win. It maps the possibility space; it does not place the bet.

Everything off-model is still real.
The grammar is silent on whether you are allowed. Licensing, sanctions, tax, consumer protection, the whole regulatory world lives at the edge, on Access, because it changes by jurisdiction and year. A correct flow can still be an illegal one. The grammar makes a system legible, not permitted.

CHAPTER FIFTEEN

Twelve rules for *builders*

Twelve rules, each one line you can say out loud, with a sentence to ground it. They are short on purpose, because a rule you cannot remember is not a rule. The three laws describe the layer; these instruct the builder.

1. **Lead with ValueType.** Geography belongs at the edge, not on the unit.
2. **Trust the Issuer, not the symbol.** Everything downstream is inherited from a decision you did not make.
3. **Every protocol is its own TransferType.** Pretending otherwise hides a routing decision.
4. **Make every Exchange explicit.** If the flow hides the conversion, someone is paying for it and it is probably you.
5. **Write flows as tuples.** Diagrams that match the grammar review themselves.
6. **Name the in-between.** For every partial state, say who holds the value and what moves it next.
7. **Design for instant by default.** Build as if every TransferType is always on. The slow ones are catching up.
8. **Compose with boring stablecoins.** The dull, reserved ones do not break. Innovate above the Issuer.
9. **Plan for cores that never turn off.** Interpolation, not disruption. Settle back cleanly or you have no strategy.
10. **Prefer the welcoming ecosystem.** A turning knowledge loop is the most durable moat.
11. **Give the format away.** A standard nobody owns is the one that spreads.
12. **Keep the grammar smaller than the catalog.** ValueTypes grow without bound. The primitives must not.

CODA

The coming *decades*

The next thirty years will do to money what the last thirty did to publishing. The pattern is already legible: clearing systems a century ago, the consumer internet, digital bank cores, mobile, Bitcoin, stablecoins. Each one a derivation of the last, each one installed in a frenzy and then deployed, quietly, for decades.

We are at or just past the turning point. The frenzy installed the infrastructure, the chains and wallets and issuers and the regulatory arguments, and most of the capital that paid for it will not survive, the way most of the capital in every installation phase does not. What survives is the boring infrastructure and the small grammar that describes it. The deployment phase, where useful things compound for fifteen, thirty, forty years, is the part that is ahead of us, not behind. It will not be one winner. It will be many companies and many generations.

Which returns to the third law, the one this book opened with. The value does not accrue to whoever replaces the old system, because nothing gets replaced; disruption was always a story told to raise money. It accrues to whoever connects the old to the new and speaks both fluently. A connector that is any-to-any compounds: every TransferType added multiplies the paths through it, and you win the seam by giving the standard away and staying neutral on purpose. Derivations connect; they do not replace. The most valuable place to stand is the seam.

"Money doesn't get replaced. It gets connected. And the connector gets paid."

Global by default. Always on by default. Programmable down to the last transfer. Indifferent to distance. The value layer is already here, unevenly, waiting to be composed. It is no longer a metaphor but a workbench: three primitives, a small grammar, and an open invitation. What gets built on it is the only part this book cannot write.

APPENDIX A

An atlas of *value*

A starting catalog, written to be read by a person or parsed by a program. Each row is a ValueType with its Issuer, the TransferTypes that carry it, and a few representative points of Access. Not exhaustive; a working sample you extend into your own domain.

FIAT VALUETYPES

VALUETYPE	ISSUER	TRANSFERTYPES	ACCESS
USD	U.S. Treasury / Federal Reserve	ACH · Wire · FedNow · RTP	Banks · treasury APIs · aggregators
EUR	European Central Bank	SEPA · SEPA Instant · TARGET2	Banks · PSPs · fintechs
GBP	Bank of England	Faster Payments · BACS · CHAPS	Banks · fintechs
BRL	Banco Central do Brasil	PIX · TED	Banks · local PSPs
INR	Reserve Bank of India	UPI · IMPS · NEFT · RTGS	Banks · UPI apps
MXN	Banco de México	SPEI · CoDi	Banks · local PSPs

STABLECOIN VALUETYPES

VALUETYPE	ISSUER	TRANSFERTYPES	ACCESS
USDC	Circle	Ethereum · Solana · Base · Stellar · Polygon	Exchanges · Wallets · Issuance APIs
USDT	Tether Limited	Ethereum · Tron · Solana · Avalanche	Exchanges · Self-custody
USDP	Paxos	Ethereum · Solana	Exchanges · Issuance APIs
PYUSD	Paxos	Ethereum · Solana	Consumer apps · Exchanges
SBC	Brale	Ethereum · Solana · Base · Canton · Stellar	Exchanges · Wallets · Issuance APIs

PROTOCOL-ISSUED VALUETYPES

VALUETYPE	ISSUER	TRANSFERTYPES	ACCESS
BTC	Bitcoin protocol	Bitcoin · Lightning · Wrapped (new Issuer)	Exchanges · Self-custody · Lightning
ETH	Ethereum	Ethereum · Base ·	Exchanges · Self-

APPENDIX A · CONTINUED

TOKENIZED REAL-WORLD VALUETYPES

VALUETYPE	ISSUER	TRANSFERTYPES	ACCESS
Tokenized T-bill fund	Asset manager / transfer agent	Ethereum · Base · private protocols	Qualified investors
Tokenized MMF share	Fund issuer	Stellar · Polygon · private protocols	Institutional

CLOSED-LOOP VALUETYPES

VALUETYPE	ISSUER	TRANSFERTYPES	ACCESS
Consumer wallet $	The wallet operator	Operator's internal ledger	The operator's app
Merchant balance $	The platform	Platform internal ledger	The platform

How to read this atlas, for a program as much as a person. **Issuer** for a protocol-issued ValueType is the protocol itself; there is no party, the rules are the issuer. **Wrapping changes the Issuer**, so wrapped BTC is a different ValueType from BTC. **A stablecoin's TransferTypes** are the chains its Issuer has minted on; the same unit on two chains is one ValueType on two TransferTypes. **A closed loop** is its own TransferType; value inside it moves only inside it. To route a transfer, find the ValueType, read across to a TransferType that carries it, and confirm an Access point exists on both ends. To convert, that is an Exchange between two rows. **Disclosure:** SBC is issued by Brale, the author's company; it appears here under the same rules as every other row.

APPENDIX B

Glossary and *grammar*

The whole model on one page. Each term is one sentence. The grammar block restates the language formally, so it can be lifted directly into a prompt or a parser.

VALUETYPE
A unit of value with an identity that survives transfer; it names its Issuer.

TRANSFERTYPE
A network that moves a ValueType from one place to another and returns a receipt.

EXCHANGE
A conversion from one (ValueType, TransferType) pair to another, at a Rate and a Cost.

ISSUER
The party, state, or protocol whose obligation gives a ValueType its meaning; a property of the ValueType, not a separate primitive.

ACCESS
Who is allowed to author a Leg; the edge of the model, where geography, regulation, and the human live.

LEG
One move, written as (ValueType, TransferType, Amount).

FLOW
A sequence of Legs joined by Exchanges; every payment reduces to one.

DERIVATION
An old primitive returning in a new context, reshaped by what is now possible.

INTERPOLATION
The connection of legacy and new without replacing either; bandwidth rises, cores stay on.

THE THREE LAWS
Identity travels with obligation, not with price. Everything is a transfer. Money is never replaced, only connected, and the connector gets paid.

```
Leg       ( ValueType, TransferType, Amount )
Exchange  Leg_in -> Leg_out, with Rate and Cost
Flow      Leg ( Exchange Leg )*
LegState  pending | settled | failed
rule: name a holder (ValueType, TransferType) per state
```

APPENDIX C

If you are an *agent*

This book is also a specification, and the web has been expecting you. Its original specification reserved status code 402, *Payment Required*, then waited decades for a buyer that could read it. You are that buyer: no hands, a wallet, a policy, and a task. This page is your instructions; the rest of the book is your reference.

Procedure.

To design a money movement: one, identify the input Leg, its ValueType, TransferType, and Amount. Two, identify the desired output Leg. Three, find a path from input to output, inserting an Exchange wherever either changes. Four, check each TransferType against the eight dimensions in Chapter 3; reject any whose hours, finality, or trust do not fit. Five, name the holder for every pending or failed state. Six, output a Flow in the common format, one labeled Leg per line.

Output format.

```
// one line per leg; Exchange marked explicitly
TRANSFER  USD    wire       4,250,000
EXCHANGE  USD -> STABLE     rate, cost
TRANSFER  STABLE canton     4,250,000
CONFIRM   receipt
```

Constraints.

A wrapped or bridged asset MUST NOT be treated as identical to its origin; it has a new Issuer. An Exchange MUST NOT be hidden; every conversion has a cost and MUST appear in the flow. A flow MUST NOT be assumed permitted; legality lives at Access, outside this model. The TransferType that fits the job SHOULD be preferred over the fastest or cheapest absolutely.

The key words *must*, *must not*, and *should* read as in RFC 2119. One implementation exists as a working API; the author runs it, the book does not require it. Any system that speaks (ValueType, TransferType, Amount) speaks the value layer.

ACKNOWLEDGMENTS

Acknowledgments

This model is not mine alone. It was sharpened over many years by people who challenged it, used it, broke it, and made it better.

Matt Harris's talk on the verticalization of fintech is one of the few pieces of analysis I keep returning to a decade later. Robert Bench made me think differently about throughput. Matt Homer pushed my thinking on the regulatory overlay. Alka Gupta and Eric Saraniecki gave generous feedback on the primitives before they were ready to be primitives. Auren Hoffman is the reason I started writing the chicken-scratch notes that became the first Value Layer essay.

The method came before any of the content. Shane Parrish and the Farnam Street project, and *The Great Mental Models* in particular, taught me that a model's job is to compress a domain into a few ideas you can carry, and that the best ones travel far past where they were built. This book is that discipline pointed at money.

Four thinkers shaped how this is framed, and their language runs through the book whether or not their names appear beside it. Chiara Marletto, for the habit of reading a system by what it makes possible. Albert Wenger, for the idea that attention, not capital, is the binding constraint, and for the knowledge loop. Carlota Perez, for the shape of installation and deployment. And Byrne Hobart and Tobias Huber, for *Boom: Bubbles and the End of Stagnation* and why such phases happen and why what is left behind defines the next era.

The ecosystems behind every TransferType in these pages gave room to experiment without knowing how it would play out. And to everyone who kept asking the question, *what do you actually mean by the value layer*, thank you. This book is the answer I have been trying to give for a decade.

Ben Milne

ABOUT THE AUTHOR

About the *author*

Ben Milne has spent two decades working alongside teams that build the infrastructure that moves money, most of it in the unglamorous middle of the system: the rails, the settlement, the funds flows nobody sees until they break.

Those teams invented social-network-invoked transactions and built one of the first GPS-aware payment systems, then shipped one of the first real-time settlement networks in the United States, years before the country had an instant rail of its own, and later rebuilt that design as open infrastructure with the Gates Foundation.

Today he is the founder and chief executive of Brale, a platform for issuing and settling stablecoins, and he builds from Des Moines, Iowa. He has spent those twenty years making one argument in a hundred different rooms: that the value layer is a real thing you can name, draw, and build on. This book is that argument, finally written down.

He writes, and keeps a working implementation of this grammar, at *benmilne.com*.

PRIOR WORK

Prior *work*

The grammar in this book grew out of two decades of building and writing. The inventions below were filed years before the model had a name; the essays trace how it found one. Neither is the book — both are the soil it grew from.

INVENTIONS

US9582788	Dynamically selecting sending and receiving accounts — priority 2010.	2017
US9792636	Social-network transaction processing.	2017
US10607268	Social-network transaction processing system.	2020
US11270269	Reducing information requirements in digital electronic transfers.	2022
US11277386	Maintaining security in digital electronic transfers through use of a label.	2022
US11270292	Key-pair authentication in a label tracking system.	2022
US11580542	Distributed database stored at an edge application.	2023

Issued U.S. utility patents on which the author is a named inventor. The year shown is the year of grant.

WRITING

2021	*"The Value Layer of the Internet"*
2022	*"The Value Layer — Expanded"*
2025	*"Better Funds Flows"*
2025	*"Value Derivations"*
2026	*"The Physics of Money"*

Essays published at benmilne.com from which this book was drawn.

The Value Layer

AN INDEPENDENT EDITION

By Ben Milne

benmilne.com

FIRST EDITION · 2026