THE FUTURE FOUNDATION

UXP

Why Compression Is a Dead End

The Tractatus Core, the Rule as the Way,
Public Readability, Streaming, Parallel Execution and Readaddressing

C.H.F. Warnke
The Future Foundation
Entry Paper No. 03
Revised Working Draft - September 2026

https://the-future-foundation.com

THREE ENTRY PAPERS INTO TRACTATUS PRINCIPIO

Publication III continues the sequence opened by Publications I and II.

Publication I — All You Need Is Difference

Why Mathematics Works — and Still Fails Reality

Publication II — The Illusion of Construction

Why There Is No Matter

Publication III — UXP

Why Compression Is a Dead End

Publication I establishes Difference as operationally prior to the numerical and axiomatic forms that presuppose it.

Publication II extends the same relational inversion to construction, matter, observation, and the stabilized physical object-readout.

Publication III carries that movement onto an executable surface. It asks what a numerical inscription is when number is not treated as a container, what a rule does when it is not added from outside, and what remains of compression once source, path, address, readout, and counter-reading are no longer interpreted as separate stored objects.

The three papers therefore do not begin from three unrelated problems.

They follow one movement through three public surfaces:

Difference

→ mathematical readability

→ physical stabilization

→ executable readaddressing.

The arrows are expository. They do not describe substances being transferred from one domain into another.

The complete project and the Tractatus Principio are available at:

https://the-future-foundation.com

ABSTRACT

Compression has spent decades asking how the same information can be stored in fewer symbols.

UXP begins one operation earlier and rejects the premise hidden in that question.

A number is not a container for a source.

Nothing is stored in the number.

The number is where a relation becomes publicly readable under a fixed rule.

This distinction changes the architecture from the ground up. If a public numerical readout is not treated as a storage vessel, then the central compression question — “Where did the rest of the data go?” — is already malformed. There are no “rest of the data” that must be hidden somewhere inside the number, beside the number, or behind the number. That question arises only after source, number, and output have first been reified into separate objects and the output has then been assigned the task of containing the source.

The Tractatus Principio begins before that reification.

Difference precedes number.

Relation precedes value.

The path precedes the stabilized readout.

Number is a public reading of an enacted relation.

There is therefore no separate number space into which UXP maps a source. What ordinary computation calls input and output are, internally, different addresses and readaddresses of the same relational network. UXP does not move a numerical entity from one numerical domain into another. It changes the public address at which the same relation is readable.

The final executable architecture follows that order.

Tractatus_Core is the application-neutral Vollzug of relation. It knows no UXP rule, no compression rule, no numeral base, no address limit, no file format, and no worker policy. It carries the ordered relation and the global relational positions generated by the Vollzug.

UXP_CLI supplies the application perspective. The rule is fixed before execution and remains invariant during that execution. Under that rule the same relation is read, constrained, addressed, and made public.

The final worker layer adds parallel execution without changing the relation. One file remains one relation. Multiple workers receive disjoint execution windows of the same globally addressed inscription. No worker creates a private Delta_0. No worker creates a child relation. No worker emits reconstruction metadata. The relation is not divided; only the work of materializing its already-global positions is distributed.

Streaming values may be tapped while the relation is running. Those values are read, not archived. The machine carries the present relation, not a list of every previous public observation.

The input medium may remain in its native public shell, for example hexadecimal, throughout the file stream. A forced continuous conversion into another numeral shell is not required for the relation to remain itself.

The practical result is not a better compressor.

It is a different computational model of what a readable value is.

THE MOVEMENT BEFORE UXP

From Difference to public readaddressing

Publication III must begin at the same place as Publications I and II, because UXP is not introduced as an independent technical primitive.

The first order is:

$$|$$

Difference.

The second order makes relational poles readable:

$$A|B$$

The continuing relational unfolding can be displayed as:

$$A|B|C|D|E|...$$

and its compact self-referential reading is:

$$|(|)$$

The effective self-reference of Difference is written:

$$\text{Delta} := |(|)$$

Delta is not a fourth object and not an external transformation mechanism added beside Difference. It is the effective self-reference already active in the distinction.

On the executable surface this movement becomes readable through Takt.

A Takt does not move a payload from one box to another. It is the next effective distinction of the current relation.

The computational reading can therefore be followed as:

$$\text{Difference}$$
$$\rightarrow \text{Delta}$$
$$\rightarrow \text{Takt}$$
$$\rightarrow \text{inscription}$$
$$\rightarrow \text{path}$$
$$\rightarrow \text{Sigma}$$
$$\rightarrow \text{fixed perspective}$$
$$\rightarrow \text{Lock / NF / Adr}$$
$$\rightarrow \text{public readout.}$$

This sequence is a reading order for the paper.

It is not a pipeline of independent objects.

The inscription is not a container that stores the path.

The path is not a hidden list stored behind Sigma.

Sigma is not a log file.

The address is not a lookup key into a previous execution.

The public number is not a scalar codeword containing what came before.

Each later expression is another readable determination of the same enacted relation.

That is the movement Publication III will now follow.

Only after that movement is visible does the title question become precise:

Why is compression a dead end?

Because compression begins by assuming the very container ontology that this movement never requires.

MOVEMENT I — BEFORE THE CONTAINER

A computational provocation

The paper begins with an ordinary machine fact rather than with a compressor.

The same visible numerical surface can enter radically different application relations. The marks alone do not announce whether they are shader, pixel, instruction, address, character, mask, or value.

That small fact is enough to reopen the question that conventional computation usually closes too early:

What exactly is being stored in a number, and what is supplied only by the rule under which the number is read?

0. A COMPUTATIONAL PROVOCATION

Modern computation is full of special pipelines.

There are special paths for shaders.

There are special paths for pixels.

There are special paths for audio samples, instructions, vectors, addresses, texture coordinates, colors, timestamps, floating-point values, masks, network fields, and file formats.

At the application level these distinctions are useful and often essential.

But ask a more primitive question:

What is the difference between a shader and a pixel
at the level of the bare number
that forces the number itself to contain that distinction?

There is none.

A bit pattern does not announce:

“I am a shader.”

Another bit pattern does not announce:

“I am a pixel.”

The numerical inscription does not carry a self-declared semantic species.

The role appears because a rule, pipeline, instruction, convention, type, or context determines how the inscription is traversed and what counts as a valid reading.

This is not an argument against specialized hardware or specialized pipelines. Those pipelines may be extremely useful.

It is an argument against confusing the application distinction with an ontological property of the number.

The machine does not discover “shader-ness” hidden inside a numeral.

It applies a rule under which that inscription participates in shader execution.

Likewise, it does not discover “pixel-ness” hidden in another numeral.

It applies a rule under which that inscription participates in a pixel reading.

The distinction is enacted by the rule.

The number is public material for a reading.

This is the first provocation of UXP:

> If the same numerical surface can become shader, pixel, instruction, address, character, mask, or value only under a rule, why continue pretending that the meaning was stored inside the number before the rule arrived?

Conventional computing already depends on the answer.

UXP merely stops hiding it.

The rule is the way.

1. THE DEAD END

Conventional compression starts from an object.

The object is called the data.

The data have a size. A compressor receives them, produces a smaller representation, and a decompressor later reconstructs the original data from that representation.

If the smaller representation appears too small to determine the source uniquely, more machinery is introduced:

metadata,
codebooks,
dictionaries,
trees,
blocks,
indexes,
hidden state,
learned models,
side channels,
headers,
containers,
reconstruction context.

All of this can be useful inside conventional compression.

But every version preserves the same explanatory picture:

source object

-> transformed representation

-> stored code

-> inverse transformation

-> source object.

UXP does not improve that picture.

UXP rejects it as the wrong starting point.

The dead end begins when a public readout is treated as a smaller object that must somehow contain a larger object.

From that moment every objection is framed in terms of storage capacity:

How can more source states fit into fewer output states?

Where is the missing information?

How does the decoder know the source length?

Where is the path stored?

Which chunk belongs where?

Where are the child relations?

What metadata must survive?

What is hidden behind the codeword?

Those questions are reasonable if the output is a container.

They are category errors if the output is a readout.

MOVEMENT II — DIFFERENCE BEFORE VALUE

How a numerical readout becomes possible

The criticism of compression is only the negative side.

The positive movement begins here.

Publication I established the priority of Difference before number. Publication III now carries that same order into execution. Before asking what a numerical value contains, the paper asks what has to be operative for a numerical value to become readable at all.

2. DIFFERENCE BEFORE NUMBER

The Tractatus Principio does not begin with number.

It begins with:

|

Difference.

The mark is not a completed object. It is the condition and enactment of relation.

Its effective self-reference is written:

$$\text{Delta} := |(|)$$

Delta is not another substance placed beside Difference. It is Difference becoming effective upon Difference itself.

This priority matters computationally.

If number comes first, relation is something numbers enter into.

If Difference comes first, number becomes something relation can look like when it stabilizes sufficiently to be read.

The operative formula is therefore:

$$n := \text{read}(\text{Sigma_n})$$

The number is read from the trace.

It is not the trace itself.

It does not contain the trace.

It is not a box into which the path has been packed.

It is the public surface on which the relation becomes readable.

SECOND READING

The same point can be said without the formula.

The machine does not begin with a self-explanatory number and then discover relations among numbers. A numerical value becomes publicly readable only because a relation has already been distinguished, enacted, and stabilized enough to support that reading.

This is why n := read(Sigma_n) must not be reversed into the claim that Sigma is hidden inside n.

The readout is public.

It is not the container of its own relational condition.

3. THERE IS NO NUMBER SPACE

The most persistent illusion is not merely that numbers are containers.

It is that there is a separate numerical space in which those containers exist.

The ordinary computational picture quietly assumes two domains:

a source space containing values

and:

a target space into which an algorithm maps them.

That picture appears so natural that even a relational implementation can accidentally reproduce it:

source number

-> algorithm

-> UXP number.

But this is already one reification too many.

Within the Tractatus architecture there is no independent number space outside the relation.

There is no warehouse of numerical entities on the source side.

There is no second warehouse of UXP numerical entities on the destination side.

There is no external algorithm standing between two ontologically different collections of numbers.

There is an address network of Difference.

The most primitive operation required for any supposed numerical space is already present there:

distinction.

A position can only be a position because it is distinguished from another position.

An address can only be an address because a relation has made that location readable relative to other locations.

A number therefore does not enter the address network from outside.

The number is already a public reading of an addressed relation within that network.

Band VI states the point in its strongest form:

number is positional form in Difference:

an address in the network of distinctions.

This changes the computational diagram.

The wrong diagram is:

NUMBER SPACE A
-> ALGORITHM
-> NUMBER SPACE B
-> RESULT.

The UXP diagram is:

addressed relation
<-> readaddressed relation
<-> public readout.

There is no ontological leap from one domain into another.

There is no second space in which the relation is deposited.

There is no internal result-object created at the end.

What appears publicly as an “input number” is already one addressable reading of the relation.

What appears publicly as an “output number” is another readaddressed reading of the same relation under the fixed perspective.

The words input and output remain useful at the operating-system boundary.

They describe where a file enters or leaves a process.

They must not be mistaken for two ontological ends of the relation.

Internally, UXP does not perform:

object -> code -> result.

It performs readaddressing.

This is why the term is exact.

Readaddressing does not mean that a numerical object is transported to a new container.

It means that the whole current relation is made readable at another address of its own network.

The relation does not leave itself.

The rule does not stand outside it.

The readout does not become an object outside it.

The apparent “result” is simply the public visibility of the relation at the address at which the fixed rule has made it readable.

Therefore:

no number space,

no target space,

no hidden result object.

Only:

Difference

-> relation

-> address

-> readaddress

-> public reading.

Even this arrow notation is only a sequential notation for exposition.

Structurally these are not independent things passed from stage to stage.

They are different readable positions of the same relational Vollzug.

This is the deeper reason UXP is not a codec.

A codec needs an outside:

source here,

code there,

transformation between them.

Readaddressing does not.

It remains inside the one relational network whose most primitive condition is already the Difference by which any address, any number, any rule, and any public reading can appear at all.

4. NOTHING IS STORED IN THE NUMBER

This statement is literal within the architecture:

Nothing is stored in the number.

The familiar phrase “data stored as numbers” hides the rule by which those numbers are read.

No number explains itself.

The same numerical inscription can be read as:

an integer,

a character,

an instruction,

a color,

an address,

a timestamp,

a floating-point value,

a flag set,

a protocol field.

The marks do not announce which of these they “really contain.”

A program, type, instruction set, file format, protocol, or fixed perspective must already be operative.

That rule determines:

which distinctions matter,

how positions relate,

where boundaries lie,

which transitions count,

what continuation means,

which public role the inscription receives.

Conventional computing therefore already demonstrates that value depends on rule.

It simply tends to place the rule second.

The ordinary story says:

data

-> stored as numbers

-> program interprets numbers.

UXP reverses the explanatory priority:

relation

-> fixed rule / perspective

-> enacted way

-> public numerical readout.

The program is not extracting a self-contained meaning that was sitting inside the number.

It is applying the rule under which that inscription becomes readable as this relation.

The rule is already the way.

The number does not contain the rule.

The number does not contain the path.

The number does not contain the source.

The number does not contain the meaning.

The number contains none of them.

It is where they become readable.

5. THE CAPACITY ARGUMENT MISFIRES

Once number is treated as an entity, a familiar argument appears inevitable.

Suppose UXP closes on a public readout with a limited number of visible positions.

Then, one says, there are only so many such numbers.

How could more possible files than numbers be represented uniquely?

Where would the additional information go?

This argument assumes:

source object -> number object

as an injective storage mapping between two sets of entities.

That is not the UXP relation.

UXP does not assert that every source object is packed into a distinct scalar container.

It asserts that a relation under a fixed perspective has a public readout.

A bounded readout is not a bounded relation.

A finite address surface is not a finite information box.

A projection is not the thing projected.

A coordinate does not contain the space it reads.

A visible face does not contain the rotational way that makes the face current.

Likewise, a number does not contain the relational Vollzug that makes that number readable.

The question:

“How do the data fit into the number?”

asks UXP to solve a problem UXP never created.

Nothing has to fit into the digits.

6. MULTI-DIGIT INSCRIPTION IS RELATIONAL DEPTH

The same correction applies inside a multi-digit inscription.

The visible decimal ground marks are:

0|1|2|3|4|5|6|7|8|9

The movement to a new place does not introduce another ground mark.

It opens another relational level.

Thus:

125

is read positionally as:

1@Delta_2 | 2@Delta_1 | 5@Delta_0

The position is not a box holding the mark.

The position is relational depth.

This is why:

001 != 1

inside the Core.

Ordinary scalar normalization may erase the two leading zero positions.

The relation does not erase them automatically because those positions were part of the inscription.

The numeral shell is therefore a public reading surface.

It is not the ontology of the relation.

MOVEMENT III — THE EXECUTABLE RELATION

From inscription to present trace

At this point the conceptual movement reaches implementation.

This is where technical discipline becomes decisive. The executable carrier may use ordinary machine structures, but those structures are not allowed to rewrite the relation into blocks, child objects, stored histories, or private local origins.

The Core exists to protect that direction.

7. THE TRACTATUS CORE

The executable architecture protects this priority through a strict boundary:

Tractatus_Core

The Core is application-neutral.

It knows no UXP.

It knows no STOP or OPEN.

It knows no compression.

It knows no ten-position rule.

It knows no binary, decimal, or hexadecimal ontology.

It knows no filename.

It knows no worker count.

It knows no scheduler.

It knows no decoder.

It carries only what is required for the relational Vollzug itself.

For a visible inscription:

125

the operative entry begins with:

5@Delta_0

then:

2@Delta_1 | 5@Delta_0

then:

1@Delta_2 | 2@Delta_1 | 5@Delta_0

Delta_x does not come from a stored source-length field.

The first entering place is Delta_0 because it is the first entering place.

The next Takt opens Delta_1.

The next opens Delta_2.

Relational depth arises from the Vollzug.

8. STREAMING IS NOT CHUNKING

Incremental execution must not be confused with a block ontology.

A conventional streaming implementation may divide a source into chunks and then treat the chunks as independent smaller data objects.

That is not the conceptual model here.

At every Takt, the whole current relation is effective.

The movement is:

Sigma_j | next position

-> Sigma_(j+1)

The new mark does not become a child object waiting to be reassembled.

An implementation may use:

arrays,

buffers,

structs,

file windows,

threads,

iterators.

Those are machine carriers.

They do not acquire ontological authority.

A buffer is not a payload container.

A thread range is not a child relation.

A file window is not a new address universe.

The relation remains one.

SECOND READING

Streaming may use chunks as an implementation convenience.

That does not license the concept:

chunk = smaller relation.

The whole current relation is still what is being enacted. A buffer boundary can help a machine schedule work without becoming a new ontological boundary inside the inscription.

The carrier may divide work.

The relation remains one.

9. THE PATH IS THE CONTENT

Within the Tractatus, content is not first an object deposited somewhere.

Content is the way that has been enacted.

The path is not a list of historical events stored for replay.

It is the effective form produced by the ordered cuts that have occurred.

This is why the past does not require a separate archive in order to remain effective.

The machine carries what is happening now.

It does not need to archive what happened before in order for the past to remain effective in the current relation.

A running program still occupies memory.

It carries the current relation.

It carries temporary working structures.

It carries the fixed rule of the application.

But:

```
current relation != stored history.
```

That distinction is mandatory.

10. SIGMA IS NOT A LOG FILE

Sigma is the integrated trace of the Vollzug.

It must not be misread as:

```
[event_0, event_1, event_2, ...]
```

kept so a later decoder can replay the past.

The trace is the present integrated form in which performed transitions remain effective.

The same visible state can belong to different traces because the path matters.

But path dependence does not imply the existence of a hidden event database.

Past effectiveness is relationally present.

11. PROJECTIONS ARE READINGS, NOT COMPONENTS

The same current relation may be read through multiple projections.

Schematically:

Pi_vis(Sigma)

Pi_depth(Sigma)

Pi_orientation(Sigma)

Pi_medium(Sigma)

These are not stored pieces from which Sigma is reconstructed.

The direction is:

Sigma -> Pi_i(Sigma)

not:

Pi_1 + Pi_2 + Pi_3 -> Sigma.

This distinction prevents projection from collapsing into metadata decomposition.

The projections are ways the same relation becomes readable.

MOVEMENT IV — RULE, PERSPECTIVE AND COUNTER-ORIENTATION

The rule is the way

Publication II introduced the observer as a relational position rather than as an object outside the relation.

Publication III needs the computational counterpart of that idea.

P_UXP is not a human observer and not an external semantic label pasted onto a completed number. It is the fixed application perspective under which the current relation receives a determinate way, closure condition, address, and public reading.

The perspective does not contain the relation.

It fixes how the same relation is read.

12. THE RULE IS THE WAY

The application enters after the Core.

UXP_CLI fixes:

P_UXP

before execution.

The profile remains invariant during one Vollzug.

Changing the profile creates a new execution.

This is not a mere software preference.

It is the condition under which the path has a determinate reading.

The rule is the way.

This is already visible in ordinary computation.

A parser has a rule.

A graphics pipeline has a rule.

A shader stage has a rule.

A pixel format has a rule.

An instruction set has a rule.

A network protocol has a rule.

Every system that “reads data” already traverses an inscription under conditions determining what counts as position, boundary, transition, and meaning.

Conventional architectures often pretend that the data were already fully meaningful before the rule arrived.

UXP does not.

SECOND READING

“The rule is the way” does not mean that a rule object sits beside the data and issues instructions to it.

It means that there is no determinate path first and a rule second.

The path is the relation enacted under the fixed perspective.

Change the perspective and another execution is being read.

Keep the perspective fixed and the way remains determinate.

13. UXP AS A FIXED PERSPECTIVE

UXP is not the Core.

UXP is:

Tractatus_Core under P_UXP

The current profile fixes its public reading and closure conditions.

A public closure boundary is a perspective rule.

It is not an information-capacity statement.

STOP is not Carry.

OPEN is not falsity.

The rule determines when the application closes its reading.

It does not define how much “information can fit” into a public numeral.

14. CARRY BEFORE RANK

A relation must carry before rank acts.

The application does not manufacture uniqueness by preferring one arbitrary readout.

The current relation must remain coherent and counter-readable.

Its relevant projections must bind as readings of the same trace.

Only then may the fixed perspective select the canonical public reading.

Thus:

possibility
-> carrying relation
-> lock
-> normal form
-> readout.

Rank cannot rescue a non-carrying relation.

15. NORMAL FORM IS NOT A COMPRESSED COPY

A shorter normal form is not a smaller package that contains a compressed copy of a longer form.

It is the same trace class read at another address.

This distinction removes the usual question:

"Where did the omitted information go?"

Nothing was omitted into a hidden compartment.

Nothing had to be packed behind the visible result.

The relation is read differently.

The short form does not contain less.

It shows the same relation from another address.

16. COUNTER-READING IS NOT DECODING

The validity criterion is relational:

L := read(Sigma)

and:

recover(L) ≡_Sigma Sigma

This should not be translated into:

compressed code -> decoder -> original data.

Forward and counter-reading are counter-orientations of the same relation under the same fixed rule.

There is no encoder that knows something the decoder does not.

There is no secret forward state that must survive.

There is no lookup table that must be carried from one side to the other.

A strong implementation guardline follows:

If counter-reading needs the previous forward session,

UXP has failed.

The public readout and the fixed rule must suffice for the counter-oriented reading.

Crucially, the validity of that roundtrip is not determined by the number of visible positions in the .uxp inscription.

A shorter public inscription is not interpreted as a smaller storage vessel with fewer available slots.

The criterion is relational:

R_P(s, READADDRESS) = s'

and:

$$R_P(s', COUNTERREAD) = s.$$

The second orientation must return to the same relation under the same fixed profile.

That is what validates the roundtrip.

Not the glyph count of the public shell.

Not the number of decimal or hexadecimal positions.

Not a presumed storage capacity of the visible inscription.

Not because the number contains the source.

Because the number is the public reading address of the relation.

SECOND READING

Counter-reading is therefore not a rescue operation performed by a second system.

Nothing travels forward as a payload and then travels backward through a decoder.

Hinweg and Rückweg are counter-orientations of the same fixed relation.

If the reverse orientation needs private knowledge from the previous forward session, the implementation has stopped carrying that relation and has reintroduced hidden state.

17. STATELESSNESS, PRECISELY

“Nothing is stored” does not mean that RAM is empty.

It means that no hidden historical state is required as part of the UXP relation.

No previous readout list.

No source-length reconstruction field.

No path-history sidecar.

No depth-history metadata.

No winner cache.

No address lookup database.

No retained forward-session object.

No hidden copy of the input.

A fresh execution under the same fixed rule counter-reads the public relation.

That is the relevant statelessness.

MOVEMENT V — STREAMING, MEDIUM AND FILE

The present relation is enough

Once path, trace, rule, and counter-orientation have been separated from storage ontology, streaming can be stated much more sharply.

A current reading may become public without turning every previous reading into reconstruction state.

Likewise, a file, a buffer, or a numeral shell may carry execution without becoming the ontology of what is carried.

18. VALUES CAN BE TAPPED DURING STREAMING

The final architecture makes the streaming consequence explicit.

A current value can be read while the relation is unfolding.

The stream does not have to end before the relation becomes observable.

At a chosen observation point the application may tap:

the current mark,
the current Delta_x relation,
the current public-shell read,
the current UXP state.

Those readings are current.

They may be displayed.

They may be forwarded.

They may be acted upon.

They may be written to an external observer.

But they do not thereby become reconstruction state.

The central sentence is:

THE STREAMING VALUES ARE READ, NOT ARCHIVED.

A tapped value may disappear immediately after use.

The next Takt does not require an archive of previous public readouts.

The current relation is sufficient.

19. OBSERVATION CADENCE IS NOT VOLLZUG CADENCE

The relation advances every Takt.

The application does not have to request a full public reading after every Takt.

It may observe every Takt.

Every kilobyte.

At a protocol boundary.

At a file boundary.

At a lock.

Only when another process asks.

Thus:

Vollzug cadence != observation cadence.

The relation does not stop existing because nobody asks for a public readout during one interval.

And observation does not create a historical ontology.

20. THE MEDIUM IS A PUBLIC PERSPECTIVE

Machine inscriptions appear in media.

Binary is a public medium.

Hexadecimal is a public medium.

Decimal is a public medium.

The relation is not identical with any one of them.

Therefore:

relation != notation.

A machine-facing file does not have to continuously translate every incoming hexadecimal mark into an ever-growing decimal inscription merely to remain valid.

If hexadecimal is the active shell, the stream may remain hexadecimal.

A decimal view can be read when the decimal perspective is actually required.

Decimal is not the hidden truth behind hexadecimal.

Hexadecimal is not the ontologically lesser form.

Both are public shells.

21. WHY HEX MAY REMAIN HEX

Keeping a hexadecimal stream hexadecimal is not merely a performance hack.

It follows from the separation between relation and representation.

A continuous Hex-to-Decimal projection introduces another reading operation. That operation may be useful.

It is not required for the relation itself to continue.

The file stream may therefore remain:

hex -> hex -> hex -> ...

while the relation unfolds.

Another shell may be projected at a chosen readout boundary.

This preserves both performance and conceptual separation.

22. FILES DO NOT CHANGE THE ONTOLOGY

A file-facing application may look familiar:

uxp forward input.hex output.uxp

and:

uxp counter output.uxp restored.hex

The user may name the output file.

That filename is operating-system convenience.

It is not part of the relation.

It must not become reconstruction metadata.

A faithful .uxp artifact must not secretly require:

the original filename,

the original path,

a hidden source copy,

a chunk table,

a path history,

a private forward state,

a lookup key,

a worker schedule.

If the filename is required for reconstruction, the filename has ceased to be a filename and has become metadata.

That would be a different architecture.

23. THE FILE IS NOT A COMPRESSED CONTAINER

A .uxp file may superficially resemble the output of compression software.

That visual similarity proves nothing.

A compression artifact is conventionally designed as a carrier from which a decoder reconstructs a source object.

A UXP artifact is a public readout of a relation under a fixed rule.

The first asks:

What must the artifact contain?

The second asks:

Under which fixed relation is the artifact readable?

The commitments are different.

24. WHY ORDINARY SOFTWARE ALREADY POINTS TO THE RULE-AS-WAY

Consider a machine pattern:

01000001

What is it?

Without a rule, there is no single computational answer.

It may be an integer.

A character.

An instruction fragment.

A mask.

A color component.

A field inside a packet.

A position inside another structure.

The marks alone do not carry a self-announcing ontology.

A program supplies the reading conditions.

Thus ordinary software already behaves as though:

inscription + rule -> determinate readout.

UXP makes this dependency foundational rather than incidental.

25. SHADER AND PIXEL: THE NUMBER DOES NOT KNOW

Return to the opening provocation.

A shader and a pixel may participate in radically different computational pipelines.

That difference matters operationally.

But the bare number does not know which pipeline it belongs to.

There is no mark inside the numeral that says:

shader

and no metaphysical bit that says:

pixel.

The distinction is produced by how the inscription is addressed, bound, traversed, and read.

The special treatment is therefore not justified by a hidden entity inside the number.

It is justified by the rule under which the relation is being enacted.

That is exactly what UXP makes explicit:

different application rule

-> different way

-> different public reading.

The same numerical surface can enter different computational relations without the number changing into a different substance.

This is not a denial of specialization.

It is a refusal to confuse specialization with numerical ontology.

MOVEMENT VI — WHY THE COMPRESSION QUESTIONS MISFIRE

The missing data were never hidden

The earlier sections have now supplied the positive alternative.

Only now can the familiar compression objections be answered without hand-waving.

If the public readout is not a container, then questions about where the omitted pieces, hidden source, path history, or reconstruction metadata are stored have already assumed the wrong model.

26. “WHERE ARE THE REST OF THE DATA?” HAS NO REFERENT

If a UXP readout is shorter than another visible inscription, the conventional question appears:

Where are the rest of the data?

The correct answer is not:

somewhere else.

The correct answer is:

there are no “rest of the data” that the number failed to contain.

The phrase presupposes that the source was an inventory of objects and the readout was supposed to store that inventory.

UXP makes no such claim.

The source inscription is one public appearance of a relation.

The UXP readout is another public appearance of that relational order under a fixed perspective.

The relation is not a bag of symbols transferred into a smaller bag.

The short readout is not missing pieces.

It is another address of the same relation.

27. WHY PIGEONHOLE ARGUMENTS DO NOT ADDRESS UXP

RETURN TO THE BEGINNING

The objection must be tested against the point from which this paper began.

At the beginning there was not yet a number.

There was:

|

Difference.

The mark was not introduced as one entity beside another entity.

It was introduced as the condition and enactment under which a relation can become readable at all.

Only then could relational poles appear:

A|B

and only through the continuing self-relation of Difference could further relational depth become readable:

A|B|C|D|E|...

with the compact operative reading:

|(|)

and:

Delta := |(|).

Number entered later.

It entered as a public reading of an enacted relation:

n := read(Sigma_n).

This order matters here.

A pigeonhole argument that begins by placing finished source entities on one side and finished codeword entities on the other has already moved past the point at which UXP begins.

It has already replaced:

Difference

-> relation

-> address

-> public reading

with:

source entity

-> codeword entity.

That replacement is not a neutral restatement of the architecture.

It is the very container ontology the paper has been refusing from the first distinction onward.

The same is true of the decimal shell.

The marks

0|1|2|3|4|5|6|7|8|9

are not ten substances waiting to be inserted into storage slots.

They are a public reading shell of one relational field.

A visible position becomes readable only because it is distinguished relative to other positions.

The position does not precede the Difference that makes it a position.

And a number does not enter this field from elsewhere.

It is already a reading of that field.

This is the point to keep in view before any counting begins.

Before asking:

"How many things fit into ten places?"

one must first show that the architecture actually consists of ten independent places containing ten independent things.

UXP does not begin there.

It begins one operation earlier.

A pigeonhole argument correctly shows that a conventional injective storage mapping from a larger finite set into a smaller finite set cannot exist.

That theorem remains true.

The question is what object the theorem is being applied to.

A conventional pigeonhole objection silently models UXP as:

source set S

-> fixed-width codeword set C_k

where k is the number of visible positions in the .uxp inscription.

For ten visible places over the decimal ground marks

M = {0,1,2,3,4,5,6,7,8,9},

that model yields the familiar count

|M^10| = 10^10.

If UXP claimed that one complete relation had been packed into one of those 10^10 flat codewords, the objection would apply immediately.

UXP does not make that claim.

The expression 10^10 already assumes the wrong object.

It counts ten independent glyph slots over a ten-mark alphabet.

The Tractatus does not identify relational width with ten independent storage positions.

The marks

0|1|2|3|4|5|6|7|8|9

form one public reading shell.

At the break edge, the same relational order becomes readable at another depth.

Thus:

$$10 := 1|0$$

does not introduce an eleventh ground mark.

It makes the same ten-mark field readable on another relational level.

Likewise:

$$125$$

is not read in the Core as three numerical substances occupying three boxes.

Its positional reading is:

$$1@Delta_2 \mid 2@Delta_1 \mid 5@Delta_0.$$

The visible place corresponds to relational depth.

This is a different type of structure from ten flat storage slots.

The distinction can be stated directly:

$$\text{glyph position} \neq \text{relational depth.}$$

And:

$$\text{number of visible places} \neq \text{number of relational distinctions.}$$

A visible mark is readable only through its distinction within the shell.

This relational determination does not turn the shell into a collection of independent storage slots.

The argument therefore does not require an alternative cardinality.

The stronger point is typological.

The flat Cartesian object

$$M^{10}$$

is not identical with the recursively self-related address field read through the same 0-9 shell.

The shell closes and becomes readable again at another depth.

The relation therefore does not leave the 0-9 field in order to continue.

It relates that field to itself.

In the Tractatus sense, the same relational form recurs across relational depth.

The same ground marks recur.

What changes is the level of relation at which they are read.

This is relational self-similarity through the self-reference of Difference, not a claim of a separately proven set-theoretic isomorphism.

Thus the relevant contrast is not:

ten digits

versus

more than ten digits.

It is:

flat glyph slots

versus

relational depth.

A conventional storage count asks:

How many strings can be formed
from ten choices across ten slots?

UXP asks:

Which relational depth is being publicly read
at this address under the fixed profile?

Those are different questions.

The first counts inscriptions under a flat container interpretation.

The second addresses a recursively self-related relational field.

This is why the statement

10^10 possible strings

does not, by itself, establish

10^10 possible UXP relations.

To make that inference one would first have to prove:

UXP relational address field = M^10.

That identity is precisely what the architecture denies.

The public shell is:

read(Adr_P).

It is not:

the whole relational field reduced to ten glyph boxes.

Adr_P and read(Adr_P) are therefore typologically distinct.

For the same reason:

width(relation)

!=

glyph count of read(Adr_P).

A condition such as:

width(I) <= 10

is a stop boundary of the chosen reading.

It is not the declaration of a ten-slot target object into which a larger source must fit.

SELF-REFERENCE IS NOT A LOOPHOLE IN PIGEONHOLE

This point must not be weakened into the claim that fractality somehow abolishes the Pigeonhole Principle.

It does not.

If UXP were in fact reducible to nothing more than one of 10^10 flat ten-digit strings, with no further publicly determined relational address under the fixed profile, then the ordinary cardinality objection would apply.

The claim is precisely that this reduction is false.

The ten-mark shell and the recursively addressed relation are not the same type.

The technical roundtrip criterion remains:

R_P(s, READADDRESS) = s'

and:

R_P(s', COUNTERREAD) = s.

The roundtrip therefore depends on the same relation being counter-readable under the same fixed profile.

It does not depend on the combinatorial storage capacity of ten visible glyph positions.

The trace, rule, path, normal form, address, and counter-orientation are not fields packed into the numeral.

The numeral is not their container.

Therefore the number of visible positions in the .uxp inscription does not determine the validity of the roundtrip.

Roundtrip depends on the fixed relational counter-orientation.

Not on how many glyphs are visible at the public address.

The section therefore returns to the first mark with which the paper began:

|

If Difference is prior to number, then number cannot later be treated as an independent container-space without reversing the order of the argument.

And if number is a public reading of relation, then counting visible numerals does not by itself count the relation whose address they make public.

28. UXP DOES NOT COMPRESS; IT READRESSES

The difference can be stated directly:

Compression transforms a stored representation.

UXP readaddresses a relation.

Readaddressing is not a mapping from one numerical space into another. There is no second numerical domain waiting for the result. What changes is the address at which the same relation becomes publicly readable under the fixed rule.

A normal form may be shorter.

An address may be compact.

A public surface may use fewer visible positions.

But visible shortening is not the primitive operation.

The primitive movement is relational stabilization and readout.

There is therefore no foundational:

compress()

shorten()

hash_to_address()

pack_to_fixed_width()

truncate_source()

inside the Core.

If a shorter form appears, it must be grounded in the relation.

29. THE BOUNDED SURFACE IS NOT A BOX

A bounded UXP public surface is an application closure condition.

It is not a claim about storage capacity.

The strongest formula is:

bounded readout != bounded relation.

This also means:

visible positions in .uxp

!=

roundtrip capacity.

And:

relational width

!=

glyph count.

The width condition belongs to relational depth.

It says where the chosen perspective closes its current reading.

It does not count how many decimal characters a public rendering may visibly require, and it does not turn those positions into independent storage boxes.

A condition such as

width(I) <= 10

must therefore be read as a stop condition on the current relational reading.

It is not a fixed ten-position target object.

The same 0-9 shell can become readable again at another depth.

That is why the visible shell may remain numerically familiar while the relation continues recursively.

The boundary says where the chosen perspective closes.

It does not say how much source can “fit.”

And it does not define a finite storage vessel whose number of possible strings determines whether the roundtrip can succeed.

A coordinate may have a bounded public form without containing the path it addresses.

A readout can likewise be bounded without becoming a container.

The counter-reading follows the fixed relational address.

It does not recover a source by exhausting the combinatorial capacity of the visible shell.

30. PHYSICAL STORAGE DOES NOT CREATE RELATIONAL ONTOLOGY

Of course computers physically store bits.

Of course files occupy physical storage.

Of course C programs allocate arrays.

UXP does not deny hardware.

It denies the inference:

because bits are physically stored,

therefore the number is ontologically a container of the relation.

A machine carrier is still a carrier.

A stored byte still requires a rule.

A stored number still requires a rule.

A stored file still requires a rule.

Storage does not eliminate relation.

It presupposes it.

MOVEMENT VII — ONE RELATION, MANY EXECUTION WINDOWS

Parallel work without parallel ontology

Worker execution is the strongest practical place for the old model to return unnoticed.

A machine naturally divides work.

The Tractatus does not therefore divide the relation.

This section keeps that distinction strict: worker boundaries are execution boundaries, not new relational origins.

31. WHY WORKERS MATTER

The final worker extension adds an important practical result.

Parallel execution does not require parallel ontologies.

A naive worker design would do this:

file
-> chunk 0 -> local relation 0
-> chunk 1 -> local relation 1
-> chunk 2 -> local relation 2
-> chunk 3 -> local relation 3
-> reconstruction table
-> rebuilt whole.

That is precisely the architecture UXP must not smuggle back in.

It creates:

child relations,
local origins,
chunk identities,
reconstruction order,
metadata.

The final worker design does something else.

There is still only one file.

There is still only one relation.

There is still one global positional order.

The workers receive execution windows of that already-global relation.

32. ONE RELATION, MANY EXECUTION WINDOWS

Let a public file have length:

N

and let public file offset be:

p.

The global relational position is:

Delta_x = N - 1 - p

This global relation exists independently of which CPU thread performs the machine work for that position.

A worker receives an interval:

[Delta_begin, Delta_end)

of those global positions.

It does not create:

local Delta_0.

It does not create:

Worker_2_Relation.

It does not create:

Chunk_Address_2.

It merely executes a disjoint window of the same globally addressed inscription.

Thus the architecture is:

one file
-> one TractatusCore relation
-> N global Delta_x positions
-> disjoint worker execution windows
-> one shared relation.

Never:

file
-> chunks
-> child relations
-> local Delta_0
-> reconstruction table.

SECOND READING

Several workers do not mean several smaller UXP relations.

They mean several execution associations with already-global positions of one relation.

The worker may begin its machine task at Delta_k.

That does not make Delta_k a new Delta_0.

This single distinction prevents parallelization from collapsing back into a block tree.

33. THE WORKER BOUNDARY IS NOT A RELATIONAL BOUNDARY

This is the decisive invariant.

Suppose four workers are used.

The machine may divide the work into:

W0

W1

W2

W3.

But no new relational zero appears at the beginning of W1, W2, or W3.

If W1 begins at global Delta_k, then its first mark is still:

mark@Delta_k

not:

mark@Delta_0.

This matters because restarting each worker locally would create a different trace.

The final tests demonstrate exactly this.

When workers retain global Delta_x positions, the parallel final relation is strictly trace-identical to serial execution.

When each chunk is deliberately restarted at local Delta_0, the trace diverges exactly at the first worker boundary.

The experiment therefore says:

Chunking is not the problem.

Creating a new relational origin is the problem.

34. POSITION DETERMINES ORDER

The final worker architecture does not need a hidden chunk-order table.

No worker writes:

I am chunk 3.

No worker stores:

previous chunk = 2.

No reconstruction metadata says:

concatenate chunks in the sequence 4,1,3,2.

The global Delta_x relation already determines the order.

Thus:

position determines order.

The worker schedule may change.

The number of threads may change.

The physical order in which CPU cores finish their assigned windows may change.

The relational order does not.

This is a powerful separation between:

execution association

and:

relational address.

35. WORKERS DO NOT STORE HISTORY

The worker layer adds parallel work.

It does not add historical state.

No worker stores previous public readouts.

No worker records the sequence of other workers.

No worker maintains a reconstruction log.

No worker produces a private address that must later be resolved.

All workers write disjoint positions of one current whole relation.

Therefore:

worker parallelism != historical decomposition.

The scheduling layer remains external to the relational identity.

36. AUTO WORKER COUNT IS NOT A NEW RULE

The final implementation permits:

workers = 0

as an automatic scheduling mode.

The current heuristic is:

max(1, online logical CPUs - 1)

This is not P_UXP.

It is not part of the trace.

It is not stored in the file.

It is not required for counter-reading.

It is ordinary execution convenience.

The same relation may be materialized with:

1 worker,

2 workers,

4 workers,

8 workers,

another count,

and the required invariant is:

same trace.

The CPU topology changes only the scheduling of the Vollzug carrier.

It does not change the relation.

37. THE STRICT WORKER TEST

The correct worker test is not:

“Did the program finish?”

It is:

serial trace == parallel trace.

The executable check is strict.

For the same inscription:

Sigma_serial

≡_Sigma

Sigma_parallel.

The UXP reading must also remain the same.

The final tests were designed to compare:

every mark,

every global Delta_x position,

the whole trace,

the UXP reading.

The result of the worker distribution is accepted only if all of these bind.

This is important because a fast parallel implementation that changes the trace is not an optimization.

It is a different relation.

38. WORKER PERFORMANCE

The worker architecture exists for a practical reason:

one relation can be materialized much faster on a multicore CPU.

On the benchmark environment, four workers repeatedly emerged as the most useful operating point.

The exact number is machine-dependent.

That is why the scheduling policy is not frozen into the Core.

A machine with more logical CPUs may profit from more workers.

A machine with fewer CPUs may not.

The relevant invariant is not:

always use four workers.

It is:

use any worker count that preserves the same globally addressed relation.

Performance is downstream of relation.

39. WORKERS AND MEMORY

The worker design also avoids multiplying the whole relation.

Four workers do not allocate four full TractatusCore relations.

They operate on one shared whole relation.

Therefore the conceptual form is:

one Sigma

+ several execution associations.

Not:

Sigma_0 + Sigma_1 + Sigma_2 + Sigma_3.

This distinction matters both semantically and practically.

It avoids turning worker parallelism into object-tree architecture.

It also avoids multiplying the full relational carrier merely because more CPU cores are available.

40. FINAL-RELATION PARALLELISM AND LIVE-EVERY-TAKT OBSERVATION

One boundary must remain explicit.

The worker implementation parallelizes the materialization of the same final current relation.

It does not claim that a public observer performing a read after every single serial Takt may observe future worker-completed positions before their ordered Takt becomes public.

For file-oriented final-relation execution, this is not a problem.

The final trace binds strictly.

For a live-every-Takt public observer, an additional ordered commit frontier would be required:

Delta_0
-> Delta_1
-> Delta_2
-> ...

Workers may prepare future positions.

Public visibility must still respect the chosen observation order.

This is an execution issue.

It does not require child relations or chunk metadata.

41. THE FINAL FILE ARCHITECTURE

The complete file-facing architecture can now be written:

file inscription
-> global Delta_x relation
-> worker execution windows
-> one whole current Sigma
-> fixed P_UXP
-> Lock / NF / Adr
-> public readout
-> optional .uxp file carrier.

For a hexadecimal file:

hex inscription
-> same global relational positions
-> parallel worker materialization
-> UXP reading
-> hex-facing public artifact

without continuous forced decimalization.

The worker layer does not modify the Core.

The medium layer does not modify the Core.

The application perspective does not rewrite the Core.

The dependency remains one-way.

42. THE STRONGEST FAILURE CONDITIONS

The architecture can now state its own failure conditions clearly.

If counter-reading requires the previous forward session, UXP has failed.

If a source-length field must be secretly stored, UXP has failed.

If a lookup table must survive, UXP has failed.

If a hidden source copy accompanies the public address, UXP has failed.

If a filename determines reconstruction, UXP has failed.

If worker IDs are needed to reconstruct order, the worker architecture has failed.

If local chunk origins must be stored, the worker architecture has failed.

If a block tree tells counter-reading what the number “really contains,” UXP has failed.

If forward and counter-reading require different private knowledge, UXP has failed.

These are not cosmetic implementation details.

They would restore the storage ontology.

43. WHAT UXP POSITIVELY CLAIMS

UXP claims that number is a public readout of relation.

It claims that multi-position inscription is relational depth rather than an aggregate of numerical substances.

It claims that a fixed rule determines a way of reading.

It claims that the same relation may become visible through different public media.

It claims that the path remains effective as present relation rather than requiring an archive of snapshots.

It claims that valid public readout is counter-readable to the same trace class.

It claims that a normal form is another address of the same relation, not a compressed hidden copy.

It claims that streaming values may be tapped without becoming historical reconstruction state.

It claims that worker parallelism can distribute execution without dividing the relation, provided all workers retain global relational positions.

And it claims that compression becomes a dead end when it insists on asking where the relation has been packed.

44. WHAT UXP DOES NOT NEED TO CLAIM

UXP does not need to claim that conventional compression is useless.

It is useful.

UXP does not need to claim that physical storage does not exist.

It exists.

UXP does not need to claim that programs use no runtime memory.

They do.

UXP does not need to claim that specialized pipelines are useless.

They are often essential.

UXP does not need to claim that notation is irrelevant.

Notation changes public execution surfaces.

What UXP rejects is the elevation of these implementation facts into the ontology of number.

A shader pipeline may be specialized without shader-ness being stored inside a number.

A pixel pipeline may be specialized without pixel-ness being a substance in a bit pattern.

A worker schedule may be specialized without worker IDs becoming part of the relation.

A file may be stored without the public readout becoming a container of the source.

45. THE RULE IS THE WAY

The deepest computational consequence can be stated simply:

The rule is the way.

A rule is not something added after the path.

The path is what the relation does under the rule.

Different rules expose different readings.

A fixed rule produces a determinate Vollzug.

Changing the rule creates another execution.

Conventional programs already rely on this every time a parser, shader stage, pixel pipeline, instruction set, protocol, or file reader turns marks into determinate behavior.

UXP makes the dependency explicit.

The program is not merely opening a box.

It is enacting a way.

46. THE NUMBER IS WHERE THE RELATION BECOMES PUBLIC

The weak formulation says:

“The number represents the data.”

The stronger formulation is:

The number is where the relation becomes publicly readable.

This removes the container.

It removes the hidden payload.

It removes the false demand that every aspect of the path be visible as literal symbol content inside the numeral.

What remains is:

relation,

fixed rule,

enacted way,

public readout,

counter-reading.

47. WHY COMPRESSION IS A DEAD END

Compression asks:

How do we store the same data object in fewer symbols?

UXP asks:

Under which relation and fixed perspective does this Vollzug become publicly readable?

The first question leads to storage capacity.

The second leads to addressability.

The first leads to codewords.

The second leads to readout.

The first leads to metadata.

The second leads to rule.

The first leads to decoder state.

The second leads to counter-orientation.

The first leads to chunk identity.

The second permits worker windows without child relations.

The first leads to containers.

The second begins before the container exists.

That is the inversion.

SECOND READING

Visible shortening may occur.

That is not denied.

What is denied is the inference that shortening must therefore mean that a larger payload has been packed into a smaller numerical container.

UXP names another operation.

The relation is readaddressed.

Compression becomes a dead end precisely when every shorter public form is forced back into the language of storage, packing, codewords, and reconstruction containers.

MOVEMENT VIII — EXECUTABLE CHECK

Example, benchmark and failure

The paper now reaches the level at which the written relation can be tested against an implementation.

The authority direction remains unchanged:

Tractatus relation

→ paper formulation

→ executable relation

→ tests

→ benchmark.

The benchmark does not establish the theory.

It checks whether the implementation can carry the stated relation without silently replacing it with a more familiar computational architecture.

48. MINIMAL FILE EXAMPLE

A minimal file example makes the architecture concrete.

Assume a file:

example.hex

with the visible hexadecimal inscription:

A1F0

The file contains four public marks.

The global positional relation is not created by worker-local numbering. It follows directly from the whole inscription:

A@Delta_3 | 1@Delta_2 | F@Delta_1 | 0@Delta_0

The rightmost public mark is Delta_0.

The next position is Delta_1.

The relation continues globally from there.

Now suppose the machine uses two workers.

For execution convenience, the physical file may be divided into two non-overlapping windows:

Worker 0 reads: A1

Worker 1 reads: F0

This does NOT mean:

Worker 0 relation = A@Delta_1 | 1@Delta_0

Worker 1 relation = F@Delta_1 | 0@Delta_0

That would create two local origins and therefore two different relations.

Instead both workers retain the positions of the one whole file:

Worker 0:

A@Delta_3

1@Delta_2

Worker 1:

F@Delta_1

0@Delta_0

Both workers write into disjoint global positions of one shared current relation.

The resulting relation is therefore exactly:

A@Delta_3 | 1@Delta_2 | F@Delta_1 | 0@Delta_0

No chunk identifier is required.

No worker-order table is required.

No separator is required.

No local Delta_0 is created.

No previous worker result has to be stored as reconstruction state.

The worker schedule may be different on another CPU.

The relation may not.

After the global relation has been materialized, UXP applies the same fixed perspective to that relation.

If the active public file shell is hexadecimal, the hexadecimal inscription may remain hexadecimal throughout the file stream.

A file-facing frontend may then write the public UXP readout to a user-chosen artifact such as:

example.uxp

The filename does not participate in the relation.

It is only an external name for the public artifact.

In counter-orientation, the same fixed rule readaddresses the public result back to the same trace class.

The minimal movement is therefore:

example.hex

-> one global relation

-> parallel execution windows

-> one Sigma

-> P_UXP

-> public readout

-> example.uxp

and not:

example.hex

-> chunks

-> chunk codes

-> metadata

-> container

-> decoder.

49. BENCHMARK: ONE FILE, ONE RELATION, MULTIPLE WORKERS

The worker architecture was tested on an 8 MiB hexadecimal inscription.

The benchmark did not process multiple independent files.

It processed one file as one relation and distributed disjoint globally addressed execution windows across several workers.

Every benchmark configuration was required to satisfy:

$$\text{strict_same_trace} = \text{true}$$

and:

$$\text{same_uxp_reading} = \text{true}$$

against the corresponding serial reference.

The benchmark was repeated five times for each worker count.

Measured mean results in the test environment were:

Workers	Mean time	Throughput	Speedup
1	0.06844 s	116.90 MiB/s	1.00x
2	0.03311 s	241.61 MiB/s	2.07x
4	0.01894 s	422.32 MiB/s	3.61x
5	0.02901 s	275.77 MiB/s	2.36x
8	0.02209 s	362.18 MiB/s	3.10x

On this particular CPU environment, four workers produced the best mean result:

approximately 422 MiB/s

for the 8 MiB hexadecimal test relation.

This was approximately:

```
3.61x
```

the throughput of the one-worker run using the same globally addressed worker algorithm.

The exact optimum is hardware-dependent.

The benchmark therefore does not define:

$$\text{workers} = 4$$

as a UXP rule.

Worker count remains scheduling.

The invariant is:

same relation under every valid worker count.

The semantic stress tests included:

8 MiB decimal inscriptions,

8 MiB hexadecimal inscriptions,

an 8 MiB + 13 mixed-mark inscription,

awkward worker boundaries,

68 deterministic random and boundary cases.

The tested worker counts were:

1, 2, 4, 5, 8.

The result was:

strict_same_trace = true

same_uxp_reading = true

random/boundary failures = 0

The adversarial test deliberately restarted every worker chunk at a private Delta_0.

That version failed.

For a 20-position inscription, the first trace mismatch appeared exactly at the first worker boundary:

2 workers -> mismatch at global position 10

4 workers -> mismatch at global position 5

5 workers -> mismatch at global position 4

8 workers -> mismatch at global position 2

Restoring global Delta_x positions restored strict trace identity.

The benchmark therefore supports the executable distinction:

chunking itself is not the failure.

a new local relational origin is the failure.

50. MEMORY AND WORKER SCALING

The final worker implementation does not allocate one whole relation per worker.

All workers operate on one shared globally addressed relation.

Measured maximum resident memory for the complete 8 MiB benchmark process was approximately:

1 worker -> 410,996 KiB

4 workers -> 410,760 KiB

8 workers -> 410,748 KiB

The near-constant measured process footprint is important.

More workers did not multiply the complete relational carrier.

The architecture remained:

one Sigma

+ several execution associations.

Not:

Sigma_0 + Sigma_1 + ... + Sigma_n.

The performance gain therefore came from distributing executable work across CPU cores, not from duplicating the relation.

51. WHAT THE BENCHMARK ACTUALLY PROVES

The benchmark should not be described more broadly than it demonstrates.

For the tested final/file mode it establishes executable equivalence between:

serial final trace

and:

parallel globally addressed final trace.

In the tested fixtures:

Sigma_serial ≡_Sigma Sigma_parallel.

It does not yet claim that a public observer reading after every exact serial Takt will see an identical temporal sequence while workers are allowed to prepare future global positions concurrently.

Such a live-every-Takt mode requires an ordered public commit frontier.

That distinction does not reopen the Core.

It distinguishes:

final-relation parallel materialization

from:

exact serial observation order.

For file processing, the first mode is the relevant fast path.

52. PUBLIC RELEASE AND REPRODUCIBILITY

The public project home is:

The Future Foundation

https://the-future-foundation.com

The Tractatus_Core is available there for download.

The Tractatus Principio can be read there in full.

This matters for reproducibility.

The claims in this paper are not descriptions of an inaccessible implementation. The foundational text, the executable Core, the tests, and the benchmarked application can be inspected together.

The intended authority order remains:

Tractatus Principio

-> paper formulation

-> executable relation

-> tests

-> benchmarks.

That order matters.

The benchmark does not establish the theory.

The code does not retroactively define the Tractatus.

The executable implementation is a test of whether the written relation can be carried without replacing it with a more familiar computational model.

Readers can therefore compare:

the relational formulation,

the public UXP paper,

the Tractatus_Core,

the worker implementation,

and the measured benchmark behavior

against one another.

The public availability of both text and Core is part of the architecture's claim to reproducibility: the relation should not have to be trusted as a description when it can also be inspected in executable form.

53. FINAL STATEMENT

UXP is not a compressor built on the Tractatus Core.

The Tractatus Core enacts relation.

UXP fixes the perspective under which that relation becomes publicly readable.

The input medium is a reading surface.

The file is a reading carrier.

The number is a reading address.

There is no independent number space behind these readings.

There is only the readaddressing network of the relation.

The rule is the way.

Streaming makes the current relation observable before the source has ended.

Those current values may be tapped.

They do not have to be archived.

Worker execution can distribute the machine work across several CPU cores.

The relation is still one.

No worker receives a new relational origin.

No worker creates a child relation.

No worker needs a reconstruction identity.

Global position determines order.

The machine carries the present relation, not a list of everything that was previously displayed.

Counter-reading does not retrieve a payload hidden inside the number.

It traverses the same relation under the same fixed rule in counter-orientation.

There is no hidden shader inside a number.

There is no hidden pixel inside a number.

There is no hidden source inside a UXP readout.

There is no “rest of the data” waiting behind the public surface.

Nothing is stored in the number.

The number contains none of the relation, the rule, the path, or the meaning as hidden objects.

It is where they become readable.

That is why compression is the dead end.

Compression keeps asking where the information has been put.

UXP begins by recognizing that the number was never the place where the relation had to be put.

IMPLEMENTATION GUARDLINE

For avoidance of ambiguity:

“Nothing is stored in the number” does not mean that a running implementation uses no physical memory.

A program necessarily carries the current relation and temporary execution structures.

A file necessarily occupies physical storage.

A worker necessarily uses machine resources.

These physical carriers do not become hidden UXP payload.

A faithful implementation must not require:

a retained archive of previous streaming readouts,

a hidden copy of the source,

source-length reconstruction metadata,

path-history metadata,

a lookup table,

a block tree,

private forward-session state,

worker-order metadata,

local worker Delta_0 origins,

child-relation identities

in order to counter-read the public readout.

For worker execution:

one file

-> one relation

-> global Delta_x

-> disjoint execution windows

-> one shared current relation.

Worker count is scheduling.

Worker boundaries are execution boundaries.

They are not relational boundaries.

Current relation is not stored history.

Physical carrier is not relational ontology.

Public readout is not information container.

The rule remains fixed.

The way is enacted.

The relation is read.

ADDENDUM

THE BYTE AS A CLOCKED RELATION OF DIFFERENCE

A work-immanent extension to the file-facing medium reading of Publication III

Status of this addendum. The preceding 56 pages remain unchanged. This addendum extends their medium and file discussion by making explicit how a physical byte can be read without turning bits, bytes, or machine integers into the ontology of UXP.

The extension remains under the same authority direction already fixed by the paper: Difference before number, relation before value, the rule as the way, and public readout as a reading address rather than a container.

A.1 THE BYTE BEFORE THE VALUE

A byte should not enter the Tractatus relation first as the finished scalar value 0 through 255. That scalar is already one stabilized machine reading of an eight-position physical carrier.

At the machine boundary, the byte can instead be read as eight binary-facing positions:

B = b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0

bj in {0,1}

The notation {0,1} must not be reified into two primitive substances. The two marks are readable positions of one distinction. The 1 is the marked position. The 0 is not the absence of relation. It is the closed or unmarked counter-position within the same relational field.

0 != nothing

Therefore 00000000 is not an empty non-state. It is a complete eight-position configuration in which every current position is read under the closed side of the distinction.

A.2 256 CONFIGURATIONS, 255 AS THE HIGHEST READOUT

Eight binary-facing positions have a finite configuration space. The familiar count is useful here, provided it is not turned into a storage ontology:

2^8 = 256 configurations

The conventional unsigned numerical surface then reads those 256 configurations as the sequence:

0, 1, 2, ..., 255

The difference between 256 and 255 is therefore typological, not a missing state. 256 names the number of available eight-position configurations. 255 is the highest value on the conventional zero-based scalar reading of that same finite carrier.

The all-zero configuration counts. Its numerical readout is 0. The all-one configuration also counts. Its conventional unsigned readout is 255. Neither configuration is more ontologically complete than the other.

256 = configuration count

255 = highest conventional unsigned readout

This distinction matters for UXP because the combinatorial surface must not be mistaken for a set of little information objects stored inside the byte. The byte is a machine carrier on which a finite relational configuration becomes publicly readable.

A.3 THE BYTE AS SELF-RELATION OF THE DISTINCTION

The eight positions are not eight isolated bit-things that are later assembled into a byte-object. Each new position enters a relation in which the preceding distinctions are already effective.

Sigma_j | next position -> Sigma_(j+1)

Thus 1|1 is not merely two copies of the same object. The second marked position is distinguished relative to an already effective marked position. The next Takt therefore acts on a relation.

1 -> 1|1 -> (1|1)|1 -> ...

The eight-position carrier can accordingly be read as a finite stabilization of repeated self-reference of the distinction. The self-reference is not added after the bits have been collected. It is what makes their ordered relation effective in the first place.

Delta := |(|)

In this sense the byte is a small, finite machine-facing reading of the same movement already governing the Tractatus: Difference distinguishes its own already effective distinctions.

A.4 STROKE, POINT AND ORTHOGONAL READING

The binary-facing position must likewise not be reduced to the naive pair "stroke" versus "no stroke". The zero position is not nothing, and the stroke itself admits more than one public orientation.

A stroke read transversely can be read frontally as a point when the perspective turns orthogonally. The point is not a newly created object and the stroke has not disappeared. The same relation is being read from another Face.

| <-> point under i = R_90

The imaginary axis therefore supplies the operative reading by which visible extension can turn into depth without changing the carrier relation. The orthogonal reading is not a second substance and not an auxiliary information channel.

For the byte this prevents another common reification. A physical bit position is not an ontological miniature switch whose truth is contained in a scalar. It is a machine-visible position within a relation that also admits orthogonal reading.

A.5 FROM EIGHT PHYSICAL BIT POSITIONS TO TWO HEXADECIMAL PUBLIC MARKS

The current executable P_UXP profile does not need to reinstate binary as an active public UXP shell. Physical bit storage can remain a property of the machine carrier while the file-facing public shell is hexadecimal or decimal.

For arbitrary byte-oriented computer data, hexadecimal supplies a direct public reading surface: one physical byte is publicly readable as two hexadecimal marks.

8 physical bit positions <-> 2 hexadecimal public positions

For example, the machine pattern 10110010 may be read on the hexadecimal public surface as:

B | 2

This must not be translated into the claim that B2 contains 10110010, or that 10110010 is the hidden truer object behind B2. They are different public readings of the same physical carrier under different reading conventions.

10110010 ~= B|2 as public readings of one carrier

Hexadecimal is therefore sufficient as the file-facing machine shell for arbitrary byte patterns without converting the byte into a decimal scalar object first.

A.6 FILE-FACING CONSEQUENCE

At the operating-system boundary a program may receive a byte in an ordinary machine type such as unsigned char. That implementation fact must remain downstream of the relation.

The value returned by the machine type - for example 178 for the carrier whose hexadecimal surface is B2 - must not silently become the mathematical meaning of the byte inside UXP.

The file-facing reading should therefore be understood as:

physical byte carrier

-> two current hexadecimal public positions

-> Takt

-> current relation

-> P_UXP

-> read(Adr)

The two hexadecimal positions are not a two-slot package containing eight hidden bits. They are the chosen public shell reading of the same current carrier.

Likewise, a bounded I/O buffer does not become a byte block relation. The buffer only exposes the next physical carriers to the same ordered Vollzug. The live relation remains

one.

This extension therefore strengthens the streaming rule already stated in Publication III: the next machine-visible carrier is read into the present relation and need not be archived as a previous byte-object, bit-object, or scalar history.

A.7 PROFILE SPECIALIZATION

This addendum distinguishes a physical machine carrier from the active public UXP shell profile.

Physical machine fact: contemporary byte-addressed machines physically carry bytes whose state can be described through eight binary-facing positions.

Current executable UXP profile: hexadecimal and decimal are the active public shells. Binary is not reintroduced here as an additional public UXP shell.

The role of the physical bit description is therefore explanatory and file-facing: it shows why arbitrary machine bytes can be brought into the hexadecimal public shell without treating a byte as a scalar information container.

A.8 IMPLEMENTATION GUARDLINE FOR BYTES

The byte extension is faithful only if the following distinctions remain explicit:

physical bit carrier != binary ontology

byte != scalar object 0..255

00000000 != absence of relation

256 configurations != highest readout 255

two hexadecimal marks != container for eight hidden bits

unsigned char != UXP mathematical primitive

hexadecimal public reading != compression code

buffer boundary != relational boundary

A faithful implementation may use machine bytes, nibble extraction, bounded buffers, integer registers, and ordinary file APIs. Those carriers are permitted because implementation convenience remains downstream of the relational reading.

What is forbidden is the reversal of authority in which the machine scalar 0..255 becomes the ontology from which the UXP relation is subsequently reconstructed.

A.9 FINAL STATEMENT

The byte does not contain eight primitive information things. It is a finite physical carrier on which one distinction is readable across eight ordered positions.

The zero position is not nothing. The one position is not an isolated thing. The ordered configuration is a clocked self-relation of Difference.

bit position = binary-facing reading of a distinction

byte = eight-position stabilization of that clocked relation

hex = public shell reading of the same byte carrier

The machine may physically store the carrier. UXP does not therefore make storage the ontology of the relation.

The byte is where a finite machine relation becomes readable. The hexadecimal marks are where that carrier becomes publicly readable under the current file-facing shell.

The rule remains fixed. The way is enacted. The relation is read.