

The Capability-Centric Ontology

What an enterprise ontology actually is, why the data-centric and object-centric answers index the wrong thing, and the authored capability core that neither one can build.

Ontology is a definitions game, and the market has two serious answers. Both index the object. Only one indexes the capability.

Ontology is the word every enterprise now uses and almost no one defines. Strip away the data catalogs wearing the name and two serious answers remain. The first is accumulation: ingest every system, record, and document, relate the nouns to one another, and call the graph an ontology. The second is executable: model the business as objects with properties and links, attach actions that write back to them, and run a live operational twin of the enterprise. Palantir built the best version of the second, and it is genuinely more than a data model.

But the lake and the twin share one commitment, and it is the one that matters. Both index the enterprise by the object. The noun is the spine, and the verb is an operation performed on a noun. However sophisticated the engine sitting on top, the organizing principle underneath is the object. That is the floor of the idea, however high the ceiling of the tooling.

The capability-centric ontology indexes by the verb. The capability, the governed statement of what the enterprise can do, is the invariant, and objects, data, and applications hang off it as attributes and relationships. The schema runs across all four domains, chip to experience, with the capability at the center. That is a different invariant, not a faster engine, and it is the ceiling. A buyer who grades every ontology as an object model will never see it, because it is not competing on the object.

THE CLAIM

The lake relates nouns. The twin operationalizes them. Both index the object. The capability-centric ontology indexes the governed verb, and that is the only invariant that survives when intelligence is general and data is everywhere.

BCM does not claim the whole ceiling. It brings the one layer that is most valuable and most impossible to build on demand.

The full ceiling is real, and it is honest to admit it is brutally expensive. A complete cross-domain ontology, every edge made explicit across business, information, application, and technology, at every altitude from the value chain down to the individual attribute and the individual device, is an enormous construction. It can be built. It is rarely built whole, because the cost of mapping every edge by hand is more than almost any program will bear.

WHAT BCM BRINGS

Carlson BCM brings the one layer that is at once the most valuable and the most impossible to produce on demand: the attributed capability core. Two things sit inside it. First, the capability model itself, roughly three thousand insurance capabilities, canonical, deduplicated, and federated from the source libraries of the enterprise, structured through the BCM hierarchy from value chain to functional how. Second, and this is the part that is routinely underpriced, the attribution of those capabilities to the operational business architecture, their users internal and external, their products and coverages, their geographies, their markets, their strategies, and their value, together with the executable attribution of value drivers, AI tier, and the governed verb that names the work.

The method is how this was built and how it extends to any new population. The asset is what the enterprise gets on day one, and it is precisely the thing no object model supplies: the authored capability invariant. The next page draws it.

The capability-centric ontology.

One capability object, attributed across the enterprise, sitting above the noun substrate the rest of the market calls an ontology.

THE CAPABILITY ONTOLOGY

STRATEGIC ATTRIBUTES

revenue · cost · CX · risk · agility

OPERATIONAL ATTRIBUTES

products & coverages · geographies
channels · roles · users



TECHNOLOGY ATTRIBUTES

core industry suite · packages
AI-native platforms

L3 governed verbs · bind · adjudicate · endorse · underwrite

JOIN

L3 governed capabilities above · L4 actions on objects below

SEMANTIC · L4 OBJECTS

Policy · Claim · Coverage
Insured Party · Premium · Agent

KINETIC · L4 ACTIONS

update status · assign
create · link · approve

DYNAMIC · GOVERNANCE

security · roles
audit · agents

THE OBJECT & DATA ONTOLOGY · THE NOUN SUBSTRATE · objects and their L4 actions over integrated data · the master data every carrier already runs

The capability ontology sits above the object and data ontology.

*The substrate is replaceable. **The invariant is not.***

CAPABILITY ABOVE · NOUN SUBSTRATE BELOW · ~3,000 CAPABILITIES MAPPED ~50,000 WAYS

Everything reachable in the ontology is reachable only through the capability. The kernel is the join, or there is no join.

Take the hardest version of the question. How does a data asset, at any altitude, a datamart, a database, an entity, a tuple, an attribute, relate to a coverage? It never relates directly. The honest path is a composition of two many-to-many joins: the asset binds to the capabilities it serves, and the capabilities map to the coverages they touch. Composing two many-to-manys does not return a clean answer. It returns a bounded possibility space, the set of coverages reachable through the capabilities the asset serves. Anyone who draws a straight line from a data element to a coverage is overstating what they have.

What the kernel buys is two things, and the second is the sharper one. The positive: the universe of what an element could relate to, bounded by the capabilities it touches. The negative: what it cannot relate to. If none of the capabilities an element serves relate to a coverage, the element does not relate to that coverage. Exclusion is the certain inference, and it is precisely what pure data modeling can never give you, because data modeling has no capability layer to rule anything out with. The certainty scales with cardinality. An attribute bound to one capability tells you nearly everything. A datamart bound to forty tells you only what holds across the union.

An object graph can traverse the links between nouns all day. It still cannot tell you which capability a datamart serves, because the capability was never one of its objects. The same substrate, indexed by capability, answers questions that look unrelated. Rationalize data assets. Trace lineage. Rationalize an application portfolio during a cloud migration. Translate a growth strategy in a geography into the capabilities, data, and authority behind it. Run the dissection that lights up a merger. One ontology, many queries. The starting node and the population change. The invariant holds.

THE LAW

Ingestion scales the data. An object twin operationalizes it. Neither authors the capability the question is about. Exclusion, and meaning, come only from the invariant they do not hold.

Two builds are stacked here, and buyers price them as one. The second is the one no enterprise, and no platform, will author for you.

The first build is the capability model. Roughly three thousand capabilities, built from scratch, is a year of work for ten disciplined people, and the discipline is the hard part. Depersonalization, verb discipline, and temporal layering have to be held across the entire model without drift. Most enterprises cannot produce it, because by the third month the model has slid back into org charts and house vocabulary.

The second build is the attribution, and it is far more onerous than the model itself. Mapping three thousand capabilities to their users, products, coverages, geographies, markets, strategy, and value is thousands of additional hours and a small army of subject-matter experts, because every edge is a judgment that has to be made and held consistent across the whole corpus. No enterprise does this on its own, and there is no reason it would, unless it is already committed to a nine-figure transformation and someone walks in with the will to hold the line.

So the honest math is blunt, and it does not spare the best tools in the market. Low-code builders, SDKs, an ontology language, forward-deployed engineers: every one of them hands you an empty model and makes you author what it means. Palantir, McKinsey, Deloitte, or the carrier itself could spend years and still not produce the canonical, attributed capability invariant, because the blocker is not compute, headcount, or tooling. It is sustained architectural discipline over a multi-year build. Carlson BCM hands over the finished asset. Sixty to eighty percent of the ontology's value, attributed and machine-actionable, on day one.

The strongest case against this paper has a name. It is Palantir, and answering it sharpens the thesis rather than softening it.

Be honest about what Palantir is, because overclaiming loses the argument. Its Ontology is the best executable object model in the market: a semantic layer of object types, properties, and links; a kinetic layer of actions that write back and trigger work; a dynamic layer of security, roles, and governance, with agents reasoning over a live twin of the enterprise. It is real, operational, and better at execution than anything in this method. None of what follows diminishes it.

But it is object-centric. Its invariant is the business object, and its verb is an action, a governed transaction performed on an object at runtime. BCM is capability-centric. Its invariant is the capability, and its verb is the governing verb that classifies the work and from which AI tier, value, and authority are derived. Same executable posture, opposite invariant. Palantir's verb is downstream of the object. The BCM verb is the spine.

They run in opposite directions. Palantir starts in the data, models it as objects, and reaches up to action: bottom-up operationalization. BCM starts at the capability and binds data and objects upward to it: top-down canonicalization. They can meet in the middle, but they are not the same layer. One is the executable substrate; the other is the authored invariant that rides on it, and not as an add-on, because the capability invariant rides on Palantir, on a data lake, or on plain SQL equally. The substrate is replaceable. The invariant is not.

Palantir's strongest answer is that its forward-deployed engineers author capability-like objects per client in weeks, and that its agents reason from objects up to capability. True, and beside the point. Object-derived capabilities authored from one client's data are bespoke and per-engagement, the exact thing that does not port. The moat was never modeling speed on a single account, where Palantir wins outright. It is canonicalization across the industry: three thousand capabilities, deduplicated and federated, so the next carrier inherits them on day one. Palantir authors meaning fast, per client; BCM arrives with it already authored for the industry.

Palantir and Carlson BCM.

Same executable posture. One sits above the other.

	Palantir Ontology OBJECT-CENTRIC	Carlson BCM™ CAPABILITY-CENTRIC
INVARIANT	the business object · a noun	the capability · a governed verb
SPINE	object types · properties · links	canonical statement · L1 → L4 · class
IN INSURANCE	L4 objects · Policy · Claim · Coverage	L3 verbs · bind · adjudicate · endorse
VERB	L4 action · a runtime transaction on an object	L3 governing verb · the classifier the work is scored from
ALTITUDE	L4 · atomic actions on live object instances	L3 · governed capabilities that compose them
DIRECTION	data → objects → actions	capabilities → data & objects
AUTHORSHIP	an engine · authored per client	pre-authored · ≈3,000 capabilities
NORMALIZATION	object graph · denormalized for ops	one invariant · Codd at the capability layer
STRENGTH	real-time execution, integration, governance, agents over live data	canonical meaning that ports across the industry

*Palantir is the executable substrate. BCM is the authored capability invariant it does not supply.
The substrate is replaceable. The invariant is not.*

SAME EXECUTABLE POSTURE · OPPOSITE INVARIANT · THE SUBSTRATE IS REPLACEABLE, THE INVARIANT IS NOT

So it comes down to this. What Carlson BCM brings was never a better way to model the enterprise. **It is a different altitude, and the one that stays scarce.**

Point all of Palantir's integration and intelligence at the enterprise and it still models objects from data and still leaves you to author what those objects mean as capability. The best execution engine in the market does not erase the scarce thing. It sharpens it. The substrate gets stronger every year, and the one component it cannot manufacture only becomes more valuable for the contrast.

As data becomes ubiquitous and intelligence becomes general, the scarce asset is neither the data, nor the reasoning, nor the substrate that runs them. It is the capability invariant that makes all of it addressable, and the invariant is authored, not bought, not ingested, not generated.

An object-centric platform can model every noun in the enterprise and act on it at runtime, and it will still sit at the level of the atomic action on the object. The capability that composes those actions, and the governed verb that names it, is a level above, and it is the level that has to be authored. That is the whole of the matter. The substrate is replaceable. The invariant is not.