OmnibusCloud

Whitepaper

Distributed computing for the real world

From crowdcomputing to the ultracomputer

Let's build the ultracomputer together!

June 2026

Abstract

The world contains billions of computing devices — desktops, laptops, phones, tablets, workstations, consoles — that sit idle for most of their operational lives. At the same time, demand for computation is rising faster than datacenters can be built, across rendering, AI, simulation and data processing. These two realities have never been meaningfully connected.

OmnibusCloud connects them. It is not another datacenter; it is the coordination layer for compute capacity that already exists. Unlike earlier distributed-computing systems — which were experimental, scientific, or narrowly specialized — OmnibusCloud is built to be a production-ready, economically self-sustaining compute network, founded on three simplicities: simple to develop for, simple to install, simple to deploy.

It begins with crowdcomputing — people contributing idle machine power to projects they care about — and that stays a permanent part of the network, not a phase it leaves behind. It also grows toward something larger: a global, heterogeneous compute fabric assembled from the machines already around us. We call that destination the ultracomputer. This paper describes the problem, the platform, how work runs, the economic model, the trust model, and the path forward.

1. The Problem

1.1 Idle hardware everywhere

Consumer hardware has never been more powerful. A modern gaming PC holds a GPU capable of trillions of operations per second; a current phone outclasses the supercomputers of two decades ago. Yet most of this capability sits unused most of the time — industry estimates put consumer-device idle time at 75–90%. This is an enormous pool of capacity that has already been manufactured, purchased, powered and connected, but produces nothing during its idle hours.

1.2 Demand outgrowing datacenters

On the other side, demand for computation is rising faster than centralized capacity can be built. This is no longer only a chip-supply problem; it is a deployment problem — power, cooling, land, permitting, grid interconnection and capital intensity all slow centralized expansion, and the time between demand and usable supply can be measured in years.

A large class of workloads does not require single-digit-millisecond latency or tightly coupled supercomputer interconnects. Rendering frames, processing batches, running Monte Carlo tasks, evaluating model outputs, scanning media, generating previews, converting files and executing independent simulation shards can all tolerate distributed execution — provided the platform supplies scheduling, validation, capability matching and a trusted operating model.

1.3 The cost of building more is rising

Demand doesn't only outpace supply — it raises the cost of supply itself. The same surge in AI workloads driving the need for compute has driven up the price of the hardware compute runs on: demand for memory chips has pushed their prices sharply higher, and that increase ripples out to the cost of computers, phones and consoles alike. Each new datacenter is built from more expensive components than the one before it; as scarce inputs — chips, power, land, grid capacity — are bid up, the marginal cost of adding centralized capacity rises rather than falls.

The hardware already in people's hands tells the opposite story. It has been bought and paid for — much of it before this run-up in prices — and would cost more to replace today than it did to acquire. Its compute already exists and already sits idle. Building ever-costlier datacenters adds capacity at a rising marginal cost; coordinating machines that are already paid for adds capacity at a near-zero marginal cost in hardware. With every turn of the AI cycle, the economics tilt further toward using what already exists rather than building more.

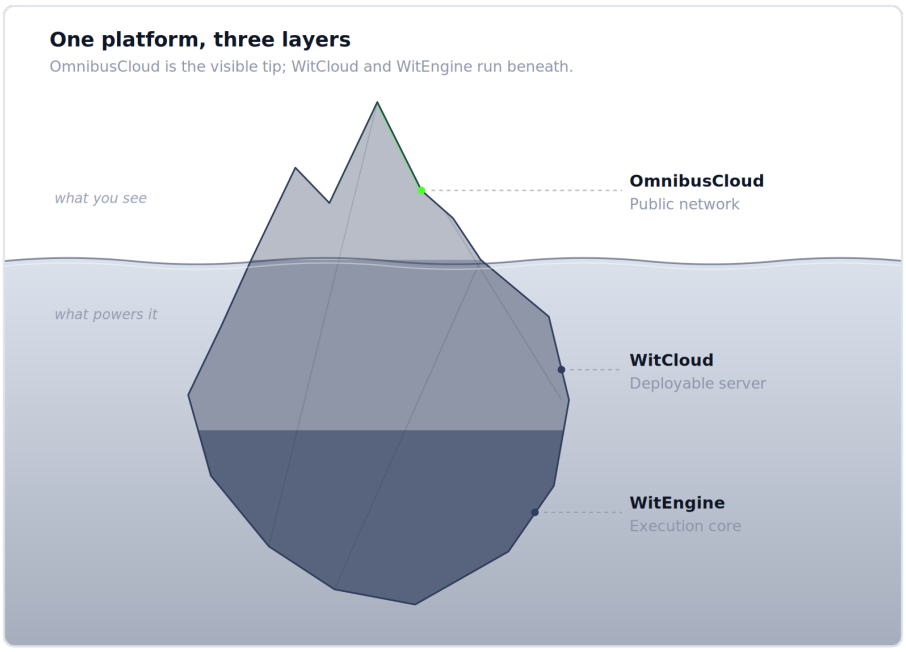
1.4 Why earlier approaches didn't scale

Several systems have tried to harness idle compute, and each hit the same kind of wall. BOINC pioneered volunteer computing, but adding a new workload meant standing up a full server stack, writing native apps per platform, and recruiting volunteers yourself — after twenty years it hosts roughly two dozen active projects, and its volunteer-only model limits participation to enthusiasts. Folding@home reached massive scale but is closed and single-purpose. Golem introduced a flexible marketplace but requires cryptocurrency, a specialized daemon and Docker expertise. Render focuses on GPU rendering, but new workload types go through a governance vote measured in months.

The common pattern: every previous platform made it hard — or impossible — for outside developers to add new kinds of computation. That bottleneck limits the diversity of work, which limits the network's usefulness, which limits participation, which limits the platform's value. OmnibusCloud is designed around the opposite premise: simple for developers to extend, simple for users to install, simple for organizations to deploy.

2. The Platform

OmnibusCloud is built as a three-layer stack, each layer a self-contained product, and each building on the one below. The same core runs everywhere — in a single office, across a global enterprise, or on the open internet. What changes between them is not the engine, but how widely it reaches.



2.1 WitEngine — the execution core

WitEngine is the distributed-computing foundation. It defines units of work, distributes tasks across heterogeneous machines, balances load by capability, handles retries and failures, and manages the controllers that carry workload logic. It is agnostic to where it runs — the same engine operates identically behind a firewall or across the public network, and can be licensed independently. This is the layer that makes many separate machines behave as one.

2.2 WitCloud — the deployable server

WitCloud is the server product. Install it on your own hardware, connect your machines, and you have a private distributed cluster. It manages client registration, controller distribution, health monitoring, capability-based matching, scheduling and result aggregation. This is what an organization deploys on-premise — its office machines become a governed compute pool, under its own policy, with nothing leaving its boundary.

2.3 OmnibusCloud — the public network

OmnibusCloud is WitCloud opened to the internet as a public platform. It adds the social layer — the portal, where projects are published and people join with their machines — and the multi-sided economy that coordinates owners, creators, developers and partners. The public layer launches as crowdcomputing — a permanent, community-driven mode — and grows into the wider ultracomputer around it, without leaving crowdcomputing behind.

2.4 Controllers: how the platform learns new work

OmnibusCloud isn't a single-purpose system. New kinds of work are added through controllers — components that define how one type of computation is prepared, split, executed and validated. Rendering is the first; simulation, asset processing and other safely distributable workloads can follow. The first controllers are open source, and developers build and test new ones locally before submitting them — but open participation never means unreviewed execution: every controller is reviewed before it can run on contributors' machines, and stays self-contained when it does — bringing its own dependencies and touching nothing outside itself. This is what makes the platform programmable rather than fixed.

2.5 Scripts: composing work across controllers

If controllers define individual capabilities, scripts are how those capabilities combine. A script describes a flow — activities wired together into one job — and those activities can come from different controllers, not just one. A developer writing a script can build on activities created by other developers: one team's rendering step, another's data-processing step, a third's encoding step, assembled into a single pipeline. A new controller doesn't just add an isolated capability; it becomes a building block everyone else can build on. That is the difference between a collection of separate tools and an ecosystem — each controller added to the network increases what every script can do.

2.6 Heterogeneous by capability

The network isn't made of identical servers — it is made of wildly different machines. OmnibusCloud measures what each can do and assigns work proportionally: a powerful GPU takes more, a modest laptop takes less, and both contribute. Tasks only reach machines that meet a controller's requirements, and only while a machine is idle and within its owner's rules. Health monitoring keeps work off machines in active use, and fault tolerance reassigns tasks when a machine drops off. This capability-aware matching is what makes a heterogeneous network productive instead of chaotic.

3. Three Simplicities

What makes a distributed platform viable isn't raw capability — it is how easily its capabilities grow. OmnibusCloud is designed around three.

Simple to develop for. Adding a new computation means writing a controller with standard tools and testing it locally — not deploying a server and recruiting a community. A new workload type takes a day, not months.

Simple to install. One cross-platform client. Install it, set your rules, and the machine is part of the network — no daemons, no wallets, no per-project setup.

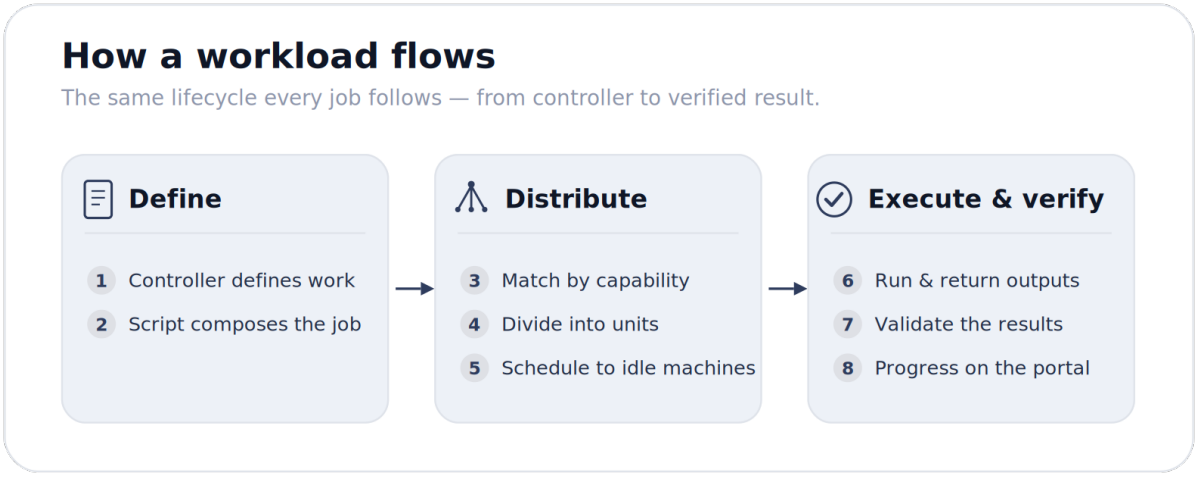
Simple to deploy. One server turns an organization's machines into a private cluster, or connects to the public network. No multi-service stack to operate.

These three are the bottleneck every earlier platform ran into. Removing them is what lets the network grow in workloads, participants and reach.

4. How Work Flows

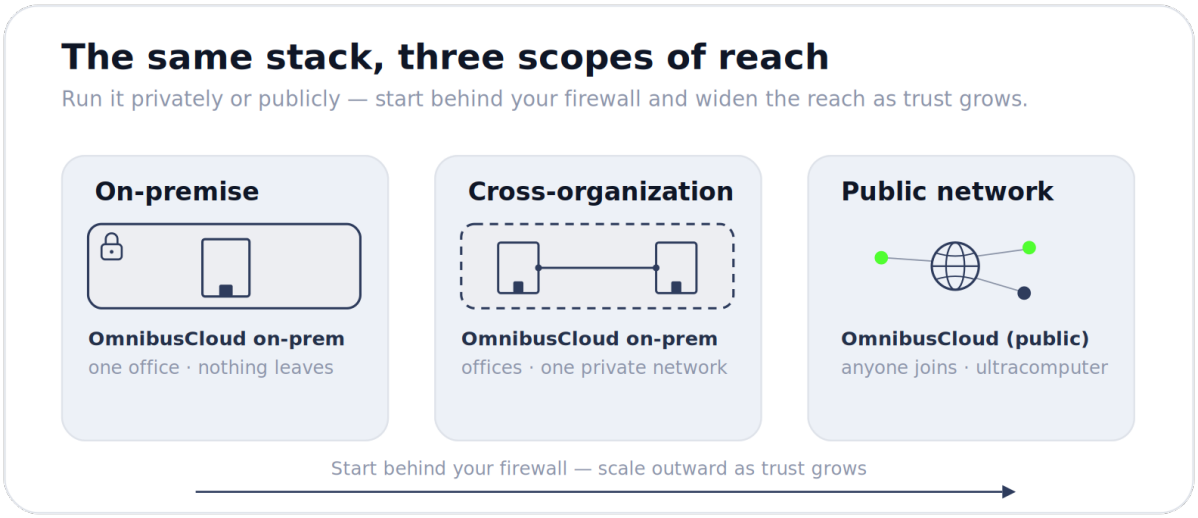
A typical workload moves through the same lifecycle:

1. **A controller defines the work** — what inputs it needs, how it splits, how results are validated.
2. **A job is defined by a script** — composing activities from one or more controllers into a flow, as a public crowdcomputing project or an internal enterprise job.
3. **Machines are matched** — by OS, CPU, GPU, memory, installed controller and availability.
4. **Work is divided** — into frames, tiles, or whatever unit the controller defines.
5. **Tasks are scheduled** — only to machines that are capable, idle, and allowed by their owner.
6. **Work runs and returns** — outputs are collected through the approved controller path.
7. **Results are validated** — useful compute must be verified compute.
8. **Progress becomes visible** — in public projects, creators and contributors follow it on the portal.



5. Deployment Modes

The same stack runs at three scopes, and they map onto the layers above.



On-premise — one office, one network, full IT control. The simplest trust model: nothing leaves the organization.

Cross-organization — offices in different time zones contributing to one private network, so compute follows the sun across a company.

Public network — the open OmnibusCloud anyone can join. This is crowdcomputing today, and the path toward the ultracomputer.

The go-to-market follows the same line: organizations can start behind their own firewall and scale outward as trust grows.

6. The Economic Model

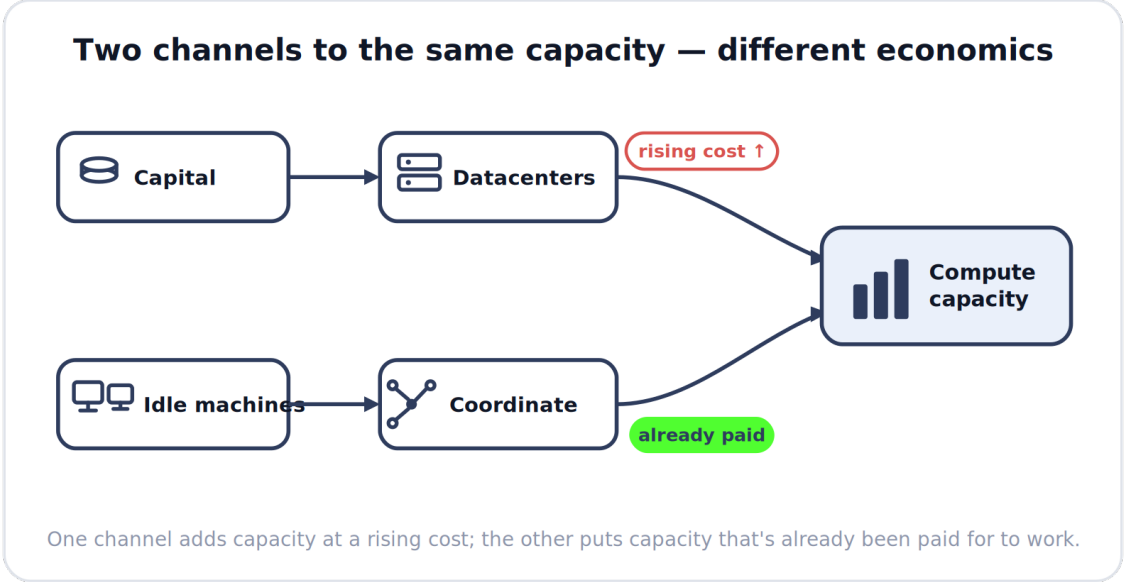
OmnibusCloud isn't trying to sell cheaper cloud. It is creating a new source of compute. Traditional cloud supply comes from capital: build datacenters, buy hardware, sell access. OmnibusCloud starts somewhere else — from the machines that already exist. Idle hardware, on its own, has no economic value; it becomes valuable only when it can be coordinated, governed, scheduled, matched to suitable work, trusted by its owner and verified after execution. That conversion — qualified availability into validated, useful compute — is the economic role of OmnibusCloud.

It is a multi-sided economy. Contributors provide qualified machine availability — for the community projects they care about, and later, once the reward model exists, in exchange for benefits too; the volunteer, community mode never closes, benefits are added alongside it, not in its place. Creators and compute consumers bring demand. Developers expand what the network can compute by building controllers, and can share in the value those controllers generate. Enterprises put their own underused machines to work as a private cluster. Partners can sponsor availability — offering a subscription, credit or other benefit instead of buying all their compute centrally.

Rewarded qualified availability. When the reward-based model arrives, its guiding principle is that rewards rest on qualified availability, not utilization. A contributor shouldn't have to wonder whether

they'll be rewarded only if the platform happened to use their machine enough that month. The promise stays simple: install the client, connect an eligible machine, follow the rules, stay available for the agreed time — and receive the benefit. Whether the machine ran for five hours or fifty is a scheduling question on the platform's side, not the contributor's problem.

Not built on cryptocurrency. OmnibusCloud is deliberately not a crypto network. Tokens and wallets add friction for mainstream users — volatility, exchange risk, compliance questions and a good deal of distrust — and tie a compute platform to speculative dynamics it doesn't need. Value here is denominated in compute and ordinary economic terms: subscriptions, credits, access, savings.



7. Trust

A network that asks people to run work on their own machines has to earn it. Trust isn't an afterthought in OmnibusCloud — it rests on mechanisms that don't change as the platform grows.

Every controller is reviewed before it runs, and on a contributor's machine it stays self-contained — bringing its own dependencies, not reading the owner's files, and writing only to a temporary folder the owner allows. The machine owner stays in control of when and whether they participate — work runs only while the machine is idle and within the owner's schedule, any machine can be paused, and participation is chosen per machine. Communication between machine and platform is authenticated and encrypted. And results are validated, because useful compute must be verified compute. The first controllers are open source, so their logic can be inspected today.

The architecture is organized around explicit trust boundaries — between the machine owner and the workload, between public projects and private data, between reviewed controllers and arbitrary code, and between platform coordination and enterprise-controlled deployments. Those boundaries are what let the system stay open enough to grow and controlled enough to trust.

And not everything belongs on crowd machines. Tightly coupled clusters, confidential data and specialized-hardware jobs stay in datacenters. OmnibusCloud is built for work that can be split, distributed across varied hardware, governed by reviewed controllers, validated after execution, and run without exposing sensitive data — it is ambitious, but not indiscriminate.

8. Use Cases

The platform starts with rendering — many frames and tiles process independently, the result is visual and immediately useful, and it is easy for a contributor to understand what they helped produce. It is the first controller and the natural proving ground.

The same model extends to other work that can be split, distributed and validated: scientific and simulation batches; enterprise idle-compute reclamation, where an organization puts its own underused machines to work; and auxiliary AI workloads — asset preparation, validation, data processing and non-sensitive batch inference that strain AI infrastructure today but don't require a datacenter. The rule is consistent: if a workload can be split, distributed, governed and verified without exposing sensitive data, it can run here.

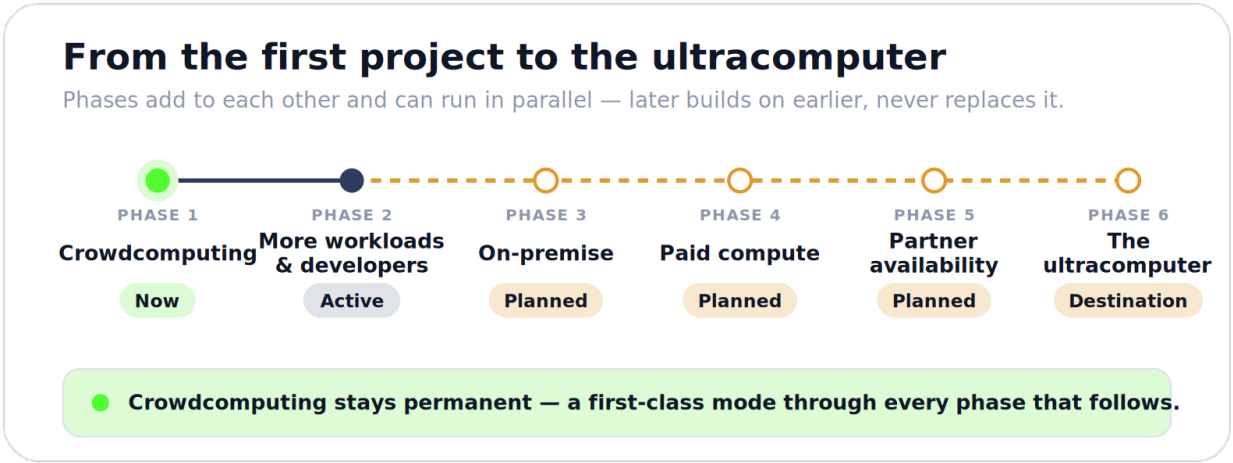
9. Vision

A supercomputer lives in one building, owned by one organization, with a fixed ceiling. When millions of machines across the world unite into one compute fabric — homes, offices, studios and pockets all contributing — the result is no longer a supercomputer. It has no single location, no single owner, and no upper bound on scale; it grows every time someone joins.

Crowdcomputing is how people enter — and it stays. Even alongside paid compute and benefits, volunteering a machine to a community project, for nothing but the project itself, remains a permanent part of the network. The ultracomputer is what the whole system grows into, with crowdcomputing always one of its modes. It will not appear all at once — it emerges as each layer is proven: participation, trust, workloads, validation, economics, developers, enterprises and partners.

10. Roadmap

The phases below add to each other — later phases build alongside earlier ones rather than replacing them — and they are not strictly sequential; crowdcomputing remains a permanent mode throughout.



- **Crowdcomputing** (now) — the public portal opens with rendering, to prove that real people will connect real machines to help real projects produce verified results.
- **More workloads & developers** — new workload types arrive as controllers, and the SDK opens the platform to outside developers, so the range of work grows without rebuilding it.

- **On-premise** — organizations turn idle office machines into a private cluster, where controlled value appears earliest, without needing a large public network first.
- **Paid compute** — once the network produces reliable, verified output, paid compute is introduced for those who need it; the principle is to monetize useful output, not activity.
- **Partner-sponsored availability** — partners offer a benefit in exchange for keeping qualified machines available, turning benefits into a compute supply channel.
- **The ultracomputer** — the coordination layer that brings every form of participation together into one global, heterogeneous compute fabric.

11. Conclusion

The compute already exists. It has been built, bought, powered and connected. The missing piece was never the hardware — it was a platform simple enough for developers to extend, simple enough for people to install, and trustworthy enough to join. Datacenters will keep growing, but they will not be the only source of compute. OmnibusCloud is building the missing layer between idle devices and real workloads.

It begins with crowdcomputing. It grows toward the ultracomputer.

Let's build the ultracomputer together!