ARCHITECTURE PROPOSAL · WORKING DRAFT

Permanent Machine Architecture

A stable whole-computer ISA for software that can remain portable, predictable and executable across generations of hardware.

CPU · MEMORY · GRAPHICS · STORAGE · NETWORK · AUDIO · I/O · ACCELERATORS

> **Core promise.** Once a PMA architecture version is ratified, a conforming future machine must continue to execute its binaries with the same architectural semantics. Hardware may change radically; the computer seen by software does not.

PMA v0.1 · Exploratory draft · 28 August 2026

NOT RATIFIED

Specify the computer, not just the CPU.

Modern systems preserve a CPU instruction set while leaving the rest of the machine - graphics, storage, audio, networking, peripherals, boot and accelerators - behind unstable interfaces. PMA extends the idea of an ISA outward until software sees a complete, durable computer.

THE PROBLEM

A stable CPU is not a stable computer

An x86 or RISC-V binary may survive the processor generation and still depend on a disappearing OS version, GPU stack, driver model, firmware interface or device implementation.

THE CONTRACT

A permanent whole-machine target

Software targets a named PMA profile containing CPU, memory, timers, device discovery and standardized machine devices. Old profiles remain implementable forever.

THE IMPLEMENTATION

Move compatibility below software

New silicon may translate old PMA operations internally, just as modern CPUs translate long-lived instruction sets into radically different micro-operations.

THE PAYOFF

Binary portability becomes normal

An application may run above an OS or directly on the machine. In either case the preserved artifact can be the actual executable environment, not merely source code that must be ported again.

Scope of this draft. This document proposes the architecture shape and a minimal executable machine. It deliberately does *not* freeze a sophisticated GPU or tensor ISA yet. Those are high-value, high-risk parts of the design and should be ratified only after emulator and FPGA experimentation.

CONTENTS

Architecture proposal + two minimal specifications

Normative language

MUST indicates a requirement for conformance. SHOULD indicates a strong recommendation with valid exceptions. MAY indicates an optional behavior.

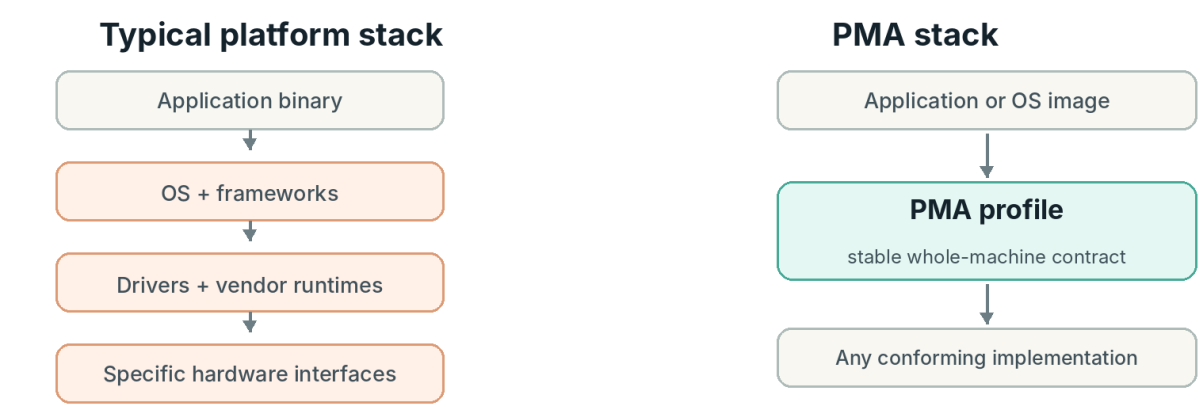
Because this is v0.1, every normative statement remains provisional. The permanence promise begins only when a profile is explicitly ratified as 1.0 or later.

This draft is intended to be small enough to implement, critique and replace.

01 · PROBLEM STATEMENT

Today, compatibility stops too early.

A program rarely targets a computer. It targets a transient combination of CPU ISA, operating system ABI, graphics API, driver stack, firmware behavior, device protocols and vendor tooling.



The goal is not fewer transistors. It is fewer moving contracts above the transistors.

1

Porting cost

The same functionality is repeatedly re-expressed for new platform APIs and device stacks.

2

Lost predictability

High-performance and real-time software depends on layers it does not control.

3

Lost software history

Old source may survive while the exact executable environment becomes difficult or impossible to reproduce.

PMA treats the machine interface itself as a long-lived public artifact.

Design for permanence without freezing implementation.

Goals

- **Whole-machine binary portability.** A binary targets a named machine profile rather than a vendor-specific device stack.
- **Stable low-level access.** Performance-critical software can use intrinsics, assembly and device command streams directly.
- **OS optionality.** A general-purpose OS is supported, but not required.
- **Implementation freedom.** Future silicon may radically change internal scheduling, widths, caches and physical devices.
- **Small trusted contract.** The normative architecture should be understandable, testable and emulatable.
- **Additive evolution.** New capabilities appear as new profiles/extensions, never silent semantic changes.

Non-goals

- **Not a new programming language.** Existing languages and toolchains should target PMA.
- **Not a required application ABI.** Operating systems remain free to define their own process and syscall ABIs.
- **Not cycle-exact portability.** Correct semantics are permanent; absolute timing and best optimization are not.
- **Not "everything as CPU instructions."** Asynchronous engines remain asynchronous devices.
- **Not a frozen 2026 GPU.** Microarchitectural details must not leak into permanent contracts prematurely.
- **Not one physical product shape.** Desktop, embedded and high-assurance systems can implement the same architecture family.

The interface is permanent. The implementation is disposable.

> **Governance principle.** Ratification should be deliberately slow. A missing feature can be added later; a bad permanent feature becomes somebody's compatibility burden for decades.

PMA should prefer reuse of mature open standards where they already match the permanence goal.

03 · MACHINE MODEL

One machine, two equally valid software models.

GENERAL PURPOSE

OS-hosted

```
Applications
    ↓
Operating system
    ↓
PMA machine profile
    ↓
Hardware
```

The OS owns protection, processes, filesystems, networking policy and resource sharing. An old OS image can itself be the preserved binary environment.

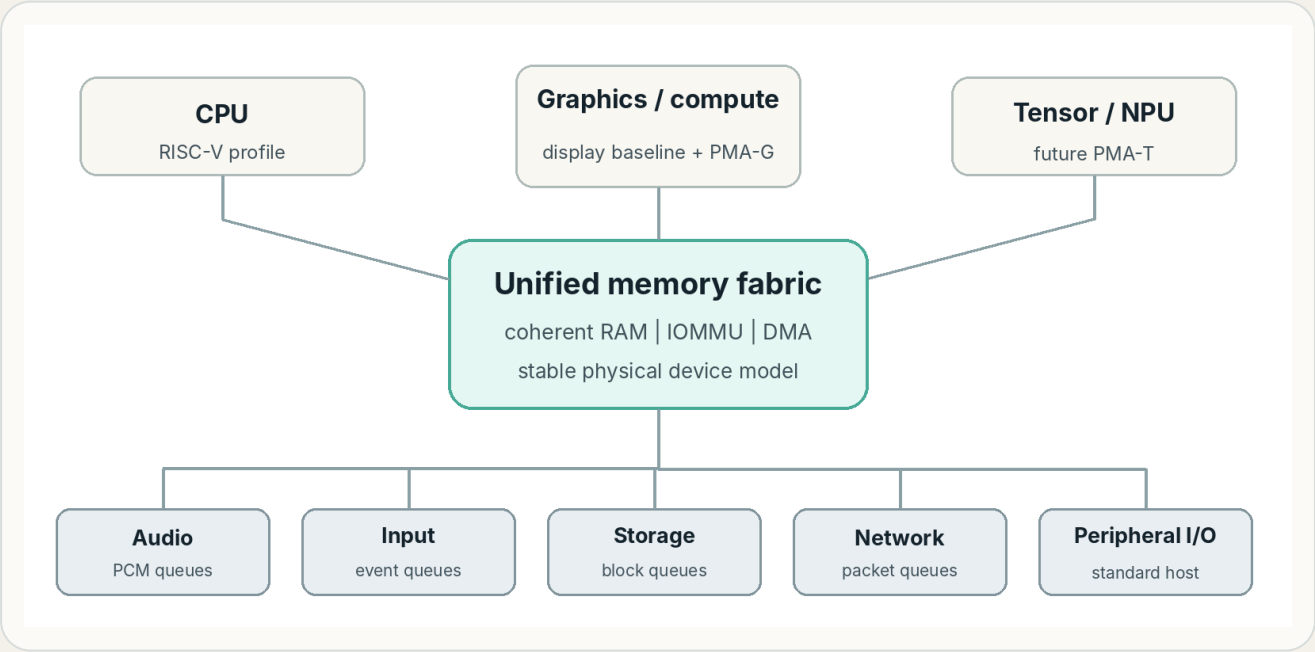
DIRECT METAL

Sovereign image

```
Application + runtime
    ↓
PMA machine profile
    ↓
Hardware
```

A game, instrument, control system or high-assurance appliance can own the machine directly and bundle only the functionality it needs.

Draft PMA-Computer-0 profile

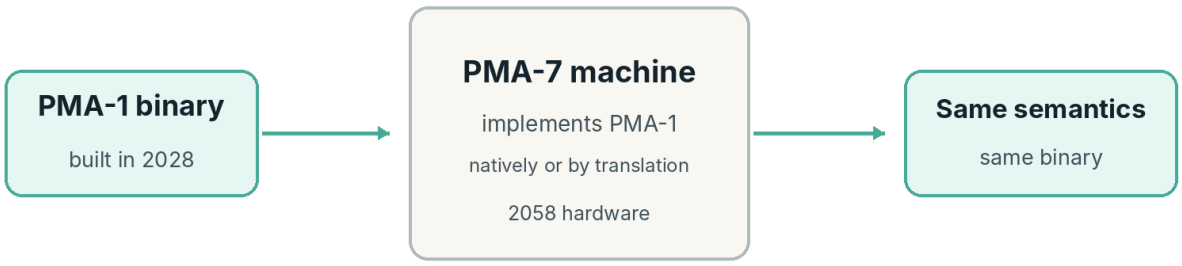


Profiles are the unit of portability: software may assume every mandatory feature of the profile exists.

04 · COMPATIBILITY AND VERSIONING

Permanence is a versioning rule, not a slogan.

PMA should copy the most valuable property of long-lived CPU ISAs: once semantics become part of a ratified target, future implementations carry them forward.



New capabilities create new targets; they do not rewrite old ones.

Draft rules

- MUST: A ratified PMA profile has immutable architectural semantics.
- MUST: Future profiles may add capabilities but cannot silently redefine behavior of an earlier profile.
- MUST: A device capability is either mandatory for a named profile or explicitly discoverable.
- SHOULD: Optional capabilities be coarse-grained. Excessive fine-grained optionality recreates portability work in application code.
- SHOULD: Conformance be validated by a public executable test suite plus architecture-level reference models.
- MAY: A future implementation use microcode, firmware translation, virtualization or emulation internally, provided guest-visible semantics remain conforming.

> **Compatibility target.** PMA promises semantic compatibility, not a permanent performance profile. Old hand-tuned assembly remains valid; new source builds may exploit newer profiles for greater performance.

PMA-0.x drafts are explicitly exempt from the permanence promise until ratification.

CPU, memory, time and interrupts

Area	v0.1 draft requirement	Rationale
CPU	64-bit little-endian RISC-V application-class baseline. Exact mandatory extension set is frozen per PMA profile; the first serious profile should be derived from an RVA23-class target rather than inventing a CPU ISA.	Reuses an open, existing binary ISA and toolchain ecosystem.
Privilege	M/S/U privilege levels for general-purpose profiles. Direct-metal software may run without a resident OS after boot.	Supports both conventional operating systems and sovereign images.
Virtual memory	A single mandatory page-table scheme for the computer profile, provisionally Sv48-class addressing.	Portable OS binaries need a predictable MMU, not merely a predictable instruction set.
Memory	Coherent system RAM visible to CPU and conforming accelerators. Cache-maintenance obligations must be explicitly specified.	Shared memory is the simplest common substrate for CPU/device queues.
DMA isolation	Standard IOMMU semantics for protected profiles; mature RISC-V IOMMU work should be adopted where suitable.	Allows devices to participate without sacrificing process/guest isolation.
Time	Stable monotonic time source plus per-hart timers. The architectural time unit is independent of CPU frequency.	Software must not infer time from a changing microarchitecture.
Interrupts	Message-signaled event model with stable interrupt-controller semantics. Existing RISC-V AIA concepts are preferred over bespoke invention.	Scales across many devices and virtualization.

> **Why not define a new CPU?** The interesting unsolved problem is the rest of the machine. PMA should spend its design budget on system-level permanence rather than rebuilding integer arithmetic, atomics, privilege and compiler backends.

CPU portability and assembly

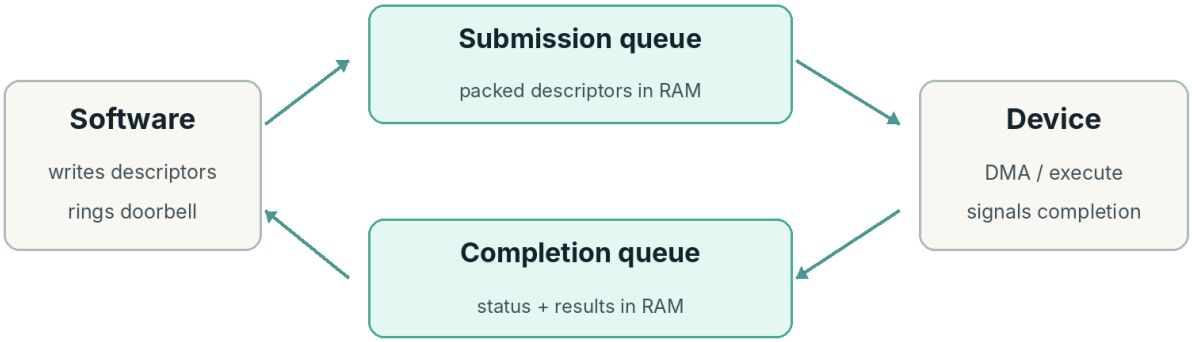
A PMA binary may contain ordinary compiler output, intrinsics or hand-written RISC-V assembly. Assembly is a first-class software artifact: it is expected to remain executable on every future implementation of the profile that defined it.

Exact CPU profile selection is a ratification decision, not a prerequisite for building the first emulator.

06 · ARCHITECTURE SPECIFICATION

A universal device contract

The device model is the heart of PMA. Devices should look like simple asynchronous machines using shared-memory command queues, completion queues, memory-mapped control blocks and interrupts.



One transport shape, different payloads: storage, network, audio, graphics, tensor and peripheral I/O.

Common device header

```
PMA_DeviceHeader
  magic             u32
  class_id          u32
  interface_version u32
  feature_bits      u64
  queue_count       u16
  event_count       u16
  queue_table_ptr   u64
  class_mmio_ptr    u64
  reserved[]        = 0
```

- MUST: Device discovery expose class, interface version and mandatory feature information before driver initialization.
- MUST: Queue formats use fixed-width packed little-endian structures and explicit memory-ordering rules.
- MUST: Devices support a polling path; interrupts are an optimization, not the only completion mechanism.
- MUST: Guest software never depend on undocumented vendor registers for conformance.
- MAY: Implementations provide vendor extensions, but portable PMA software cannot require them unless they later become a ratified profile feature.

The queue approach is intentionally similar in spirit to proven interfaces such as VirtIO, but PMA applies the concept to the physical machine contract rather than treating it primarily as virtual hardware.

The exact descriptor ABI should be designed once, fuzzed heavily, then reused across device families wherever possible.

Display, graphics and compute

Graphics is the place where a permanent architecture can most easily fail by freezing either too much hardware detail or too much software abstraction.

RATIFY EARLY

PMA Display baseline

- Enumerate displays and modes.
- Allocate/pin scanout-compatible memory.
- Present a linear framebuffer.
- V-sync / presentation event.
- Mandatory common pixel format and color-space baseline.

This is enough for a graphical OS or direct-metal application to put pixels on a screen without a vendor driver.

PROTOTYPE FIRST

PMA-G graphics/compute ISA

- Stable command stream.
- Buffer/image resources.
- Compute and graphics pipelines.
- Portable shader/compute instruction set.
- Explicit synchronization and memory semantics.

The implementation may translate PMA-G to radically different internal GPU micro-ops, just as a CPU translates a stable instruction set.

> **Deliberate omission in v0.1.** This draft does not specify shader opcodes, wave width, register files, tile models or texture units. Those choices must survive serious software workloads and at least one FPGA/RTL prototype before becoming permanent.

Rules for a future PMA-G1

- It must be low-level enough that high-performance software can submit work without a mandatory vendor runtime.
- It must not expose transient implementation choices such as a fixed physical SIMD width unless that property is essential to the programming model.
- It must define numerical behavior, memory ordering and synchronization precisely enough for independent implementations to agree.
- It should support both graphics and general compute so that one permanent accelerator contract serves many workloads.

Tensor / neural acceleration

PMA-T should follow the same principle: a stable tensor execution contract may be added later, but v0.1 should not fossilize today's model formats or accelerator microarchitectures. CPU/GPU execution remains the compatibility floor.

Audio, input and peripheral I/O

Audio output/input

- PCM buffer queues with explicit sample format, channel count, rate and frame count.
- A mandatory baseline format for every PMA-Computer profile (provisionally 48 kHz stereo float32 output).
- Timestamped completion / position information suitable for low-latency synchronization.
- No required software mixer. An OS may provide one; direct-metal software may own the device.

```
submit_audio(buffer, frames, format)
        ↓
    device DMA
        ↓
completion(frame_clock, underrun_flags)
```

Human input

- Standard event queues for keyboard, pointer and baseline game-controller input.
- Physical key/button identity is distinct from text input; operating systems may layer keyboard layouts and IMEs above it.
- Events include monotonic timestamps from the PMA time domain.
- Hot-plug device identity and capability information are discoverable.

```
PMA_InputEvent
  timestamp_ns
  device_id
  event_type
  code
  value
```

Peripheral connectivity

PMA should standardize the **host-controller contract** for common external buses rather than requiring every OS to know a vendor-specific controller. USB is an obvious first target for a desktop profile. Device-class protocols can remain layered software where appropriate.

> **Key boundary.** PMA does not promise that every future peripheral protocol is known in advance. It promises that attaching, enumerating and transferring data through the machine's standard peripheral host does not require a vendor-specific motherboard driver.

Real-time audio is a useful stress test: a stable device contract should also be a low-overhead one.

Storage and networking

PMA BLOCK

Persistent storage

The architectural device is a block store, not NVMe/SATA/UFS. Physical media and controllers may change; software sees a stable asynchronous block queue.

- Read / write / flush / discard.
- 64-bit logical block addresses.
- Mandatory 512-byte logical block compatibility.
- Device capacity and alignment discovery.
- Durability semantics explicitly specified.

Filesystems remain software and therefore can evolve independently of the machine ISA.

PMA NET

Network interface

The baseline architectural network device is a packet interface. Physical Ethernet, Wi-Fi or other media may map to the same software-visible contract.

- RX/TX packet queues.
- Baseline Ethernet-compatible framing for the computer profile.
- Checksum/offload capabilities are discoverable, never assumed.
- Hardware timestamps are optional capabilities.

IP, TCP, QUIC and application protocols live above the architecture and can be supplied by an OS or direct-metal runtime.

Why stop at blocks and packets?

Because these are low-level, stable abstractions that preserve implementation freedom. PMA should avoid freezing a fashionable filesystem, network API or cloud protocol into hardware. The machine provides durable primitives; software provides policy.

> **Open issue:** The exact boundary for wireless association, encryption and regulatory behavior needs careful treatment. A raw packet abstraction alone may be insufficient for some physical media; this should be prototyped before the desktop profile is ratified.

Boot, executable artifacts and security

Boot contract

- MUST: Every PMA-Computer implementation provide a documented immutable entry path from reset to a PMA boot image.
- MUST: The boot handoff provide a memory map, hart topology and device-discovery root using a stable packed structure.
- MUST: Direct boot of an ELF64 RISC-V image be supported by the reference emulator; physical boot-media format is defined separately.
- MAY: Implementations provide secure/measured boot, but a permanent machine must not depend on a vendor cloud service for executable authorization.

Executable preservation

```
PMA boot image
├── machine-profile requirement
├── executable image (ELF64 RISC-V initially)
├── resources / assets
├── optional signature
└── manifest
```

A direct-metal application can therefore be distributed as a self-contained machine image. An operating-system image can use the exact same mechanism and then host its own application ABI.

Exclusive / high-assurance deployment

Small trusted computing base

A machine may boot one authenticated image and transfer ownership of all architectural devices to it, with no resident general-purpose OS, shell, updater or package manager.

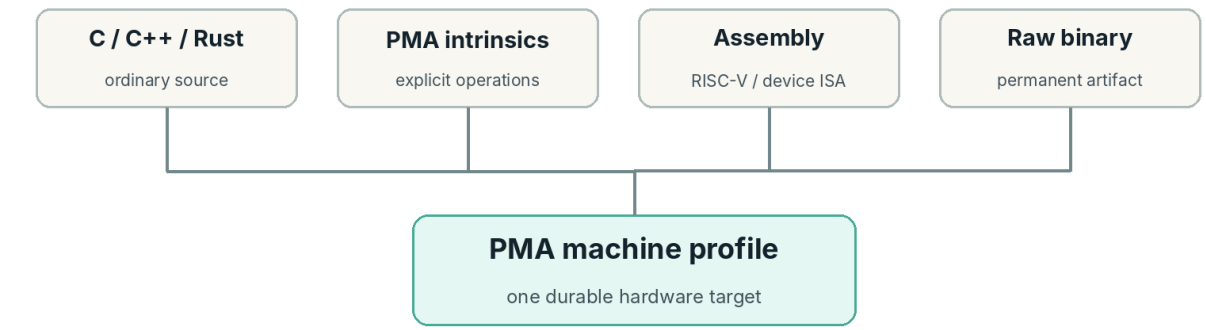
Isolation remains available

Privilege levels, page protection and IOMMU domains allow an OS or specialized runtime to build process/device isolation without changing the architecture.

Security caveat. "Only one program runs" does not automatically mean "secure." PMA reduces attack surface and hidden dependencies; application bugs, firmware, DMA, networking and supply-chain security still require explicit design.

Make low-level knowledge compound.

PMA is intentionally friendly to programmers who want to understand the machine. High-level languages remain useful, but they no longer need to hide a constantly changing hardware landscape.



Two kinds of portability

Semantic portability

The old binary still runs correctly. PMA treats this as the hard guarantee.

Performance portability

A newer compiler or hand-tuned build may exploit a newer profile better. PMA does not guarantee that 2028 assembly is still optimal in 2058.

Compiler philosophy

Compilers can remain extremely sophisticated optimizers, but their role is reduced as portability infrastructure: they target one durable machine family instead of continually translating around incompatible CPU, GPU and device ecosystems. A working compiler backend should itself have an unusually long useful life.

A hand-optimized routine can become finished software rather than a temporary platform-specific branch.

The emulator is the first real machine.

The reference emulator is not merely a demo. It is the executable definition against which the architecture can be tested before expensive hardware decisions are made.

Required modes

Mode	Purpose	Requirement
Reference	Architectural correctness	Simple, deterministic implementation favored over speed. Serves as the oracle for conformance tests.
Fast	Interactive software development	May use JIT/dynamic translation and host acceleration as long as guest-visible behavior remains PMA conforming.
Record	Capture nondeterminism	Records input, network arrival, entropy and other external events against virtual time.
Replay	Deterministic debugging	Reproduces a recorded execution without consulting live external devices.
Compliance	Architecture validation	Disables convenience backdoors and rejects behavior outside the selected PMA profile.

Emulated machine

RISC-V harts · RAM · timers · interrupts · IOMMU model · display · audio · input · block storage · network · peripheral host

- MUST: Guest software interact only through the PMA architecture in compliance mode.
- MUST: Virtual time be explicit and independent from host CPU speed.
- MUST: The same boot image execute in reference and fast modes.
- SHOULD: The reference CPU initially reuse or wrap a well-tested RISC-V execution core if doing so accelerates architectural experimentation.
- MAY: Host GPU/audio/network APIs accelerate device implementation internally; these APIs are never visible to guest software.

If the emulator needs a hidden host syscall to make a demo work, the architecture is incomplete.

Host mapping, debugging and reproducibility

PMA device	Typical host implementation	Deterministic/replay behavior
Display	Window or offscreen image; fast mode may use host GPU	Presented frames can be hashed or captured
Audio	Host audio callback or file sink	Reference mode can render exact PCM to WAV
Input	Keyboard/mouse/gamepad window events	Timestamp and record all events
Storage	Raw disk image with optional copy-on-write layer	Snapshot image at run start
Network	NAT, TAP or packet capture backend	Record incoming packets and virtual timestamps
Entropy	Host CSPRNG in live mode	Record seed/stream; deterministic fixed source in tests

Developer interface

```
pma run       demo.pma
pma run       --profile computer-0 demo.elf
pma record    --out session.pmar demo.pma
pma replay    session.pmar
pma inspect   --devices
pma snapshot  --out state.pmas
pma test      conformance-suite/

# Optional debugger integrations
pma run --gdb :1234 demo.elf
pma trace --mmio --queues --interrupts demo.elf
```

Debug facilities

CPU

Breakpoints, single-step, register/memory inspection, GDB remote protocol.

Devices

Trace queue submissions, completions, DMA ranges, interrupt delivery and protocol errors.

State

Snapshots, deterministic replay and architecture-level assertions.

> **Reference principle.** Emulator behavior that is convenient but non-architectural must be visibly marked as a debug extension and disabled by compliance mode.

Build one small program that proves the machine exists.

The first public artifact should be a single PMA boot image that exercises the complete minimal computer without Linux, Windows or a vendor driver.

Reference demo requirements

1. **Boot** from a PMA image and enumerate the machine.
2. **Display** an animated graphical scene.
3. **Read input** from keyboard and pointer.
4. **Play audio** continuously at low latency.
5. **Use storage** through the PMA block queue.
6. **Use networking** by sending/receiving packets; a tiny guest-side IP stack is enough.
7. **Run computation** using scalar, floating-point and vector CPU operations.

Success criterion

The exact same boot image runs unchanged in:

```
Reference emulator
        ↓
Fast emulator
        ↓
FPGA prototype
        ↓
First silicon
```

No source rebuild is required to move between implementations of the same profile.

Suggested implementation sequence

01

CPU + RAM + boot

ELF load, serial/debug console, timer.

02

Display + input

Make the machine immediately tangible.

03

Audio + storage

Stress DMA, queues and real-time behavior.

04

Network

Exercise asynchronous I/O and replay.

05

Compliance tests

Turn assumptions into executable contracts.

06

FPGA

Prove the model maps cleanly to real hardware.

Do not begin with a high-end SoC. Begin by making the architecture undeniable.

Questions that should remain uncomfortable.

CPU baseline

Exactly which RISC-V profile and vector features become permanent in PMA-Computer-1? How should future CPU profiles coexist without fragmenting the software base?

Graphics boundary

What is the lowest stable abstraction that can be implemented efficiently by future GPUs without exposing ephemeral wave, tile or cache details?

Unified memory

Which coherence guarantees are mandatory between CPU, GPU, NPU and DMA devices? Where are explicit flush/invalidate operations required?

Numerical determinism

Which floating-point and transcendental behaviors are bit-exact, and which are bounded but implementation-dependent?

Peripheral model

How much of USB and future external buses belongs in the architectural host contract, and how much belongs in software class drivers?

Wireless networking

Can all media cleanly present an Ethernet-like packet interface, or is a higher-level stable wireless-control device required?

Boot firmware

What is the smallest immutable boot contract? Can all policy remain in the boot image rather than permanent firmware?

GPU/NPU evolution

Should older accelerator ISAs be supported by hardware translation, firmware translation, or a protected compatibility engine? Which mechanisms are architecturally invisible?

Profiles and optionality

How few profiles can cover desktop, embedded and high-assurance uses without turning the standard into either a monolith or a combinatorial feature matrix?

Governance

Who is allowed to ratify a permanent feature? What proof - emulator, formal model, FPGA, independent implementation - is required before a 1.0 promise is made?

> **Ratification bar.** No complex feature should become permanent because it looks elegant on paper. A reasonable minimum is: two independent software implementations, a reference test suite, substantial real workloads, and at least one hardware prototype for hardware-facing semantics.

APPENDIX A · PMA V0.1 NORMATIVE SUMMARY

Minimal executable machine checklist

Component	PMA-Computer-0 status	Draft contract
CPU	Mandatory	RV64 little-endian RISC-V, application-class profile; M/S/U for OS-capable systems.
Memory	Mandatory	Coherent RAM; stable VM profile; explicit memory ordering.
Time	Mandatory	Monotonic architectural time + per-hart timer events.
Interrupts	Mandatory	Standard message-signaled interrupt/event model.
IOMMU	Mandatory for protected profile	Standard device-address translation/isolation semantics.
Discovery	Mandatory	Versioned device class registry + packed descriptors.
Device queues	Mandatory	Shared-memory submission/completion queues + MMIO doorbells.
Display	Mandatory	Mode discovery + linear framebuffer scanout + present event.
Graphics/ compute	Provisional	PMA-G to be designed after prototyping; not ratified by v0.1.
Audio	Mandatory	PCM input/output queue; common baseline output format.
Input	Mandatory	Timestamped keyboard/pointer/controller event queues.
Storage	Mandatory	Asynchronous logical block device.
Network	Mandatory	Asynchronous packet RX/TX device.
Peripheral host	Desktop profile	Standard external-bus host contract; USB first candidate.
Tensor/NPU	Optional / future	PMA-T stable tensor execution ISA after prototype evidence.
Boot	Mandatory	Stable boot handoff; emulator can directly load ELF64.
Security	Profile-defined	Memory protection, IOMMU isolation; optional authenticated boot; no cloud dependency.

One-sentence conformance test: if two independently built implementations expose the same PMA profile, the same conforming boot image must run on both without source changes, recompilation, vendor drivers or guest-visible compatibility shims.

This table is a design checkpoint, not a final register-level standard.

APPENDIX B · RELATIONSHIP TO EXISTING WORK

Build on proven ideas; extend the boundary.

PMA is not proposed in a vacuum. Several existing standards validate important pieces of the thesis, but none is intended here as a complete permanent whole-computer target.

[1] Casey Muratori - "The Thirty Million Line Problem". The motivating argument: a stable CPU ISA is insufficient when the rest of the machine requires large, changing driver and platform stacks. caseymuratori.com/blog_0031

[2] RISC-V RVA23 Profiles. Application-class profiles exist specifically so binary software ecosystems can rely on a substantial guaranteed feature set across implementations. docs.riscv.org/reference/rva23/

[3] RISC-V IOMMU Architecture. A ratified, standardized device-address translation and isolation architecture that PMA should reuse where it matches the machine model. docs.riscv.org/reference/iommu/

[4] OASIS VirtIO 1.3. Demonstrates efficient standardized devices built around descriptor rings/queues, DMA and interrupts. PMA borrows the shape of this idea while targeting a physical machine contract. docs.oasis-open.org/virtio/

[5] Arm SystemReady. Demonstrates market value in standardizing more than the CPU so old operating systems can run on new compliant hardware and vice versa. PMA pushes that objective lower and more permanently. arm.com/.../systemready-band

---

Working thesis

RISC-V can provide the permanent CPU. PMA's job is to make the rest of the computer equally boring.

This document is an exploratory design proposal. The architecture name, numeric defaults, component boundaries and profile definitions are intentionally provisional. The point of v0.1 is to create something concrete enough to implement, benchmark, criticize and replace before any permanence promise is made.