上海创智学院
Shanghai Innovation Institute

PTO

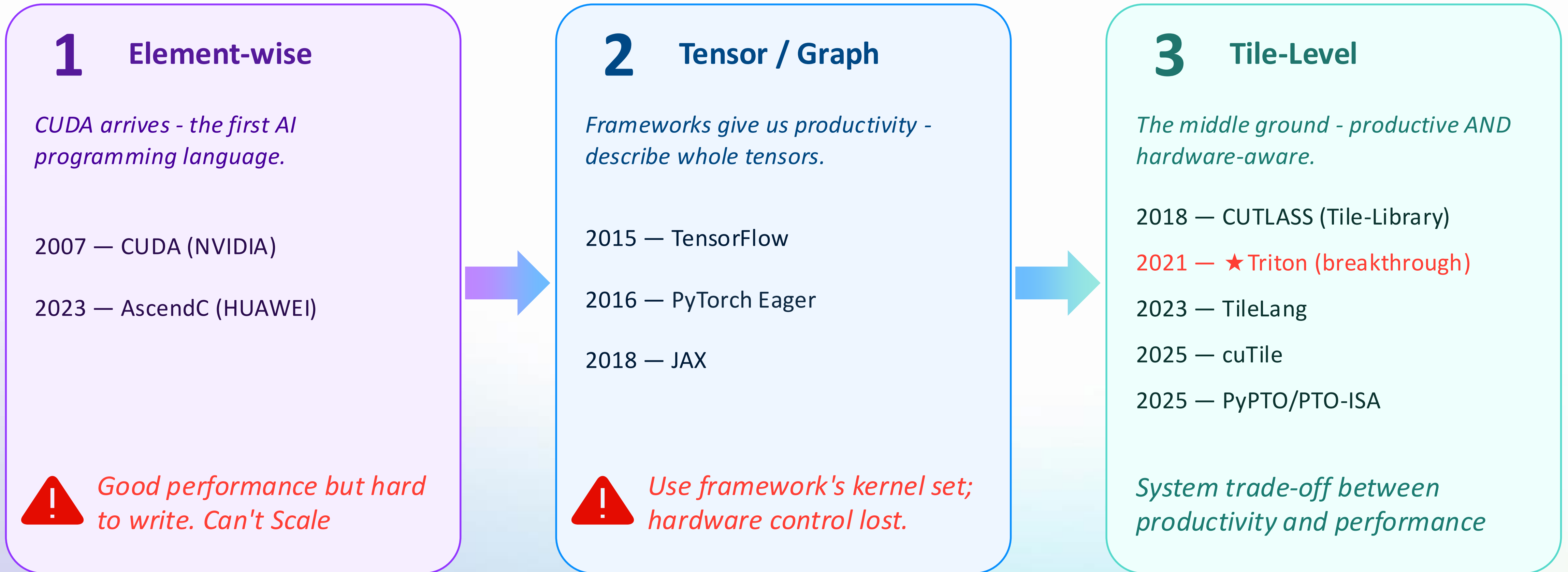
Ascend-Native Tile Programming Ecosystem

Siyuan Feng | 2026.05.12

01 Tile Programming

The History — Three Acts, Two Transitions

上海创智学院
Shanghai Innovation Institute



Tile programming wasn't invented; it emerged. Hardware made it inevitable; Triton (2019) made it approachable.

Same Operation, Three Styles — the Trade-off

上海创智学院
Shanghai Innovation Institute

Element-wise / per-thread

```
__global__ void add(float* a, float* b,
                    float* c, int N) {
  int i = blockIdx.x *
        blockDim.x + threadIdx.x;
  if (i < N) {
    c[i] = a[i] + b[i];
  }
}
```

User reasons one thread at a time.

Tile-level / per-tile

```
@triton.jit
def add(a_ptr, b_ptr, c_ptr, N,
        BLOCK: tl.constexpr):
  pid = tl.program_id(0)
  offs = pid * BLOCK + tl.arange(0, BLOCK)
  a = tl.load(a_ptr + offs)
  b = tl.load(b_ptr + offs)
  tl.store(c_ptr + offs, a + b)
```

User reasons one tile at a time.

Tensor-level / per-tensor

```
c = a + b
```

User reasons in dataflow.

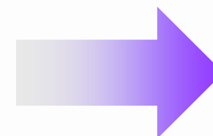
Tile = productivity of tensor-level + control of element-level — without paying either's full cost.

The Proof — Same Op, Same Hardware, 25-line gap

上海创智学院
Shanghai Innovation Institute

AscendC (element-wise) — ~50 lines, not optimal

```
class SoftmaxKernel {
  __aicore__ void Init(GM_ADDR x, GM_ADDR y) {
    xGm.SetGlobalBuffer(...);
    yGm.SetGlobalBuffer(...);
    pipe.InitBuffer(inQueueX, BUFFER_NUM, BLK * sizeof(float));
    pipe.InitBuffer(outQueueY, BUFFER_NUM, BLK * sizeof(float));
    pipe.InitBuffer(tmpBuf, BLK * sizeof(float));
  }
  __aicore__ void Process() {
    for (int i = 0; i < tileNum; i++) {
      CopyIn(i); Compute(i); CopyOut(i);
    }
  }
  __aicore__ void Compute(int p) {
    LocalTensor<float> x = inQueueX.DeQue<float>();
    LocalTensor<float> y = outQueueY.AllocTensor<float>();
    LocalTensor<float> t = tmpBuf.Get<float>();
    ReduceMax(t, x, BLK); // pass 1
    Sub(y, x, t, BLK); Exp(y, y, BLK);
    ReduceSum(t, y, BLK); // pass 2
    Div(y, y, t, BLK); // pass 3
    outQueueY.EnQue<float>(y);
    inQueueX.FreeTensor(x);
  }
  // CopyIn / CopyOut / queue + pipe — omitted
};
```



PyPTO (tile-level) — 5 lines

```
@pypto.jit
def softmax(x: Tile) -> Tile:
  m = pypto.row_max(x)
  e = pypto.exp(x - m)
  return e / pypto.row_sum(e)
```

Tile is a data conception, but not programming style —SIMT or SIMD, users don't care

PTO Stack is Tile-First by Design

上海创智学院
Shanghai Innovation Institute

Tensor level

algorithm developers — productive entry point

Tile level — load-bearing

performance experts — tile shape, memory placement, pipeline events

← this is what hardware actually does

Element level

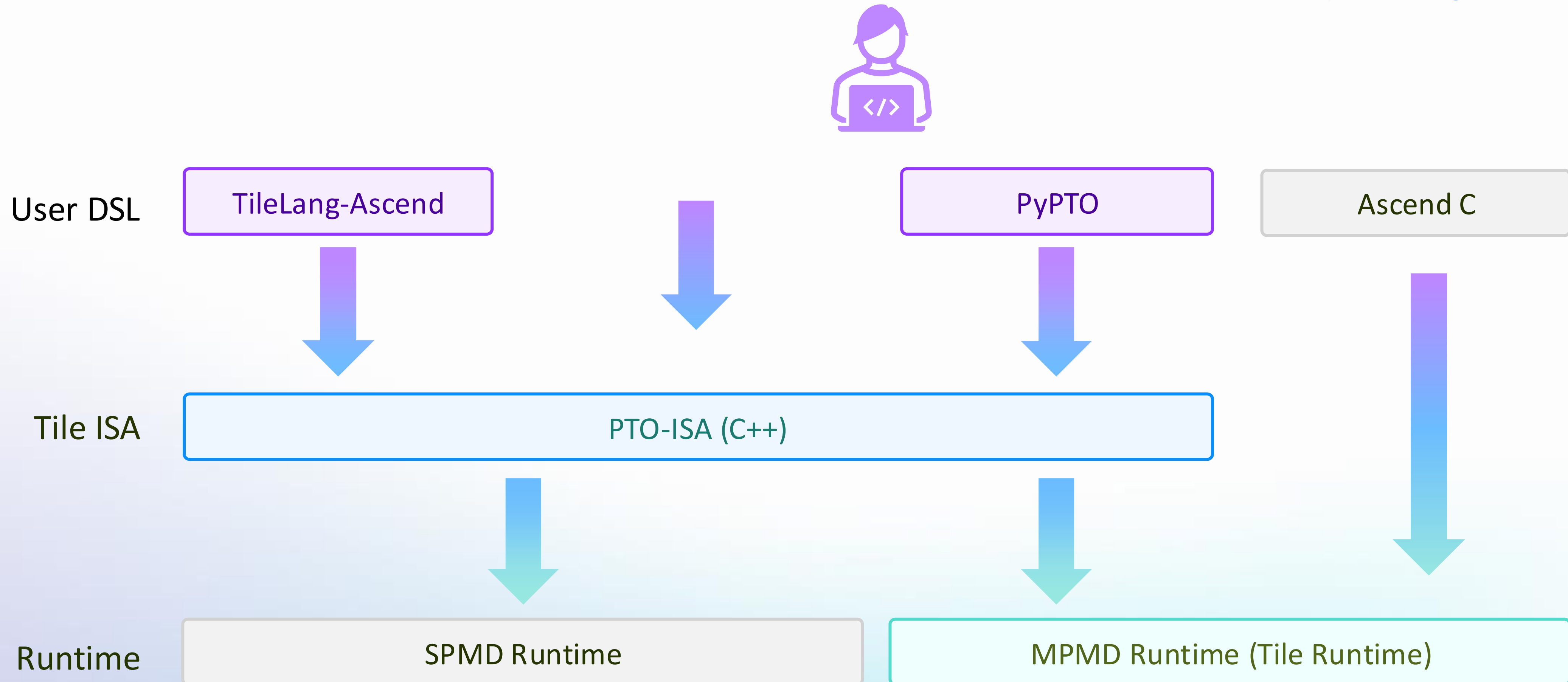
system developers — fully control the hardware behavior

- PyPTO Multi-Level Tile Programming
- PTO Tile Assembly (.pto)
- PTO-ISA (Virtual Tile ISA)
- MPMD Runtime (Tile Runtime)

Tile programming is the foundation, and tile is the first-class citizen in PTO stack

PTO is an Ecosystem — not a Block-Box Solution

上海创智学院
Shanghai Innovation Institute



Building ecosystem rather than solution — give choices to users and fit for different cases

02 Ascend-Native

Ascend-Native — Three Pillars of the PTO Design

上海创智学院
Shanghai Innovation Institute

Pillar 1

Cross-device, cross-generation

One PTO-ISA across A2 / A3 / A5 even Kirin NPU; one kernel source survives generation transitions.

Pillar 2

Native primitives exposed

AIC, AIV, GM/Vec/Mat, TPUSH/TPOP pipeline events, Auto/Manual mode — all addressable in PyPTO.

Pillar 3

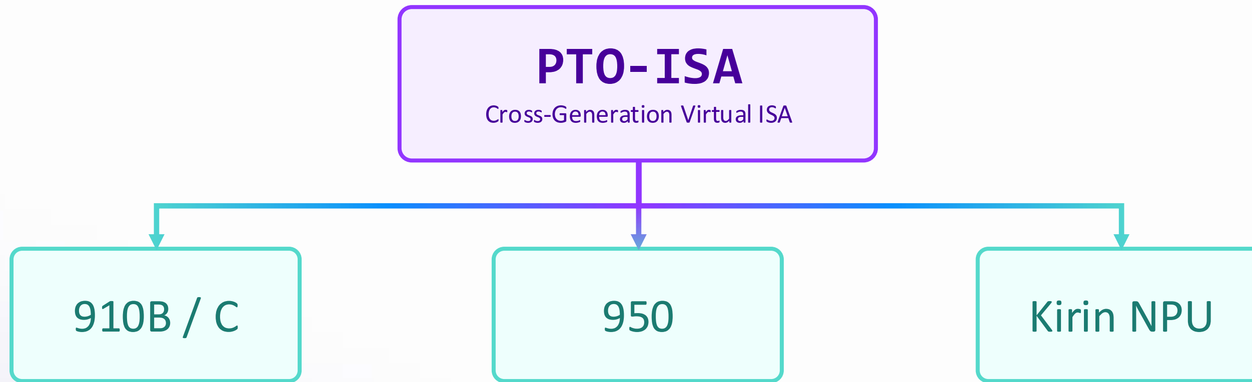
MPMD-native runtime

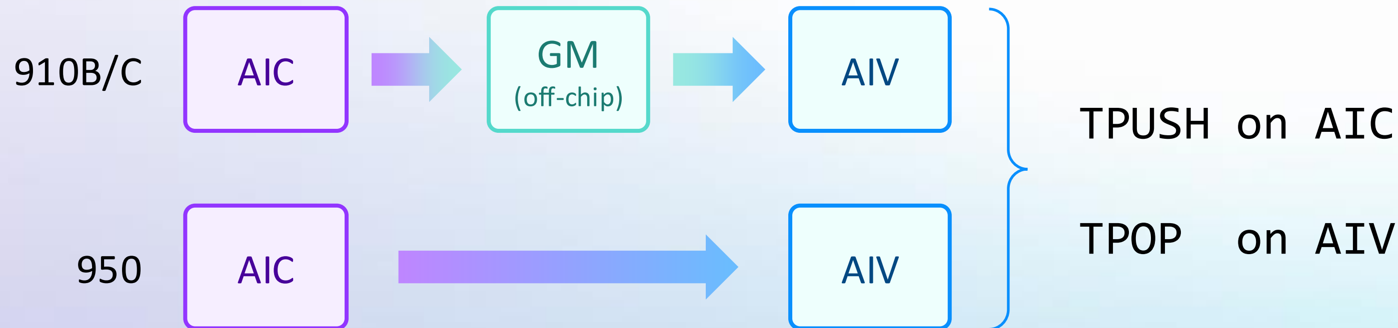
MPMD Runtime co-schedules independent programs per core; PyPTO calls this 'MPMD Execution Scheduling.'

Triton & TileLang are developed for CUDA from day one. CuTile is NV-specific. PTO ecosystem is specific to Ascend natively .

Pillar 1 — One Tile ISA, Every Ascend Hardware

上海创智学院
Shanghai Innovation Institute





Cross-hardware

The same ISA for not only Ascend NPU on cloud / car, but also for NPU in Kirin Soc

Cross-generation

One source compiles for 910B/C and 950; ISA absorbs differences in buffer placement, intra-group comm.

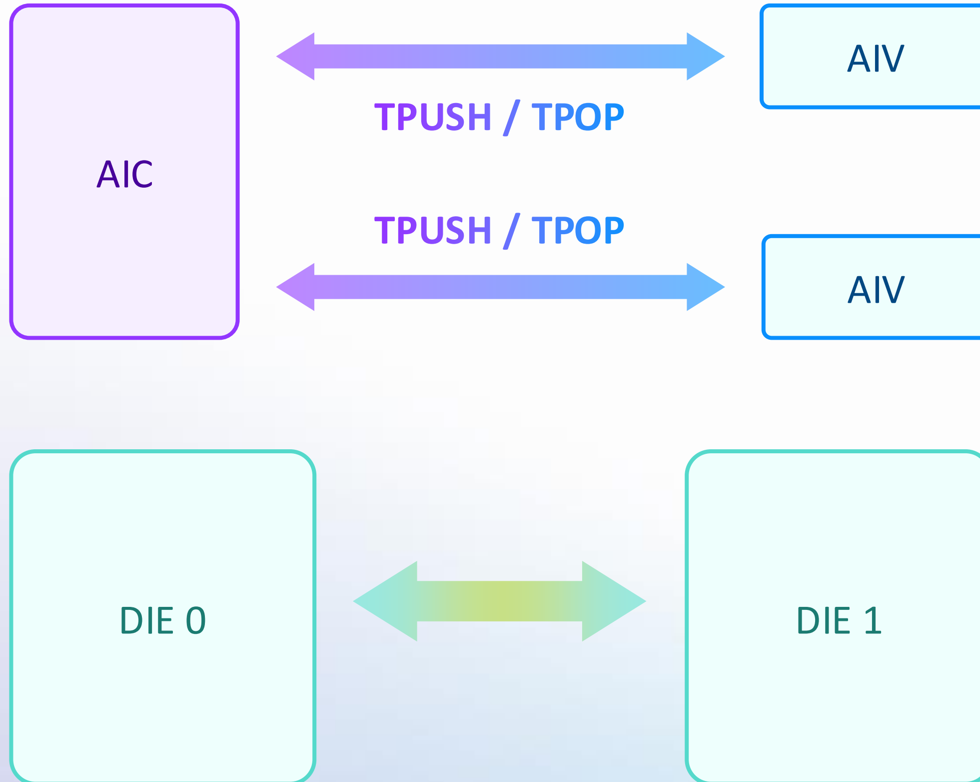
Cross-frontend interchange

PyPTO, TileLang-Ascend, PTO AS both emit the same PTO ISA via different workflow, fit in different target cases

Triton and TileLang chase per-arch intrinsics every hardware generation, while PTO unifies ISA from day one.

Pillar 2 — Leverage Hardware-Native Primitives

上海创智学院
Shanghai Innovation Institute



GPU uses homogeneous cores

GPU uses in-core heterogeneous design while NPU leverage heterogeneous cores. Hardware diff -> programming behavior diff

GPU-incubate DSL can't fit heterogeneous cores

Triton and TileLang fused kernel does not know how to split workloads into heterogeneous cores, to be specific, 2AIV

Die-to-die consideration

Cross-die program optimization is still a new topic for both NPU and GPU

Triton-Ascend and TileLang-Ascend port a GPU-shaped frontend. They cannot natively express AIC vs AIV or pipeline events.

Pillar 3 — MPMD by Design

上海创智学院
Shanghai Innovation Institute

SIMT + SPMD

— CUDA on GPU

SIMT inside a single kernel, and each kernel runs on all SM

SIMD + SPMD

— Ascend C on NPU

SIMD programming style inside a kernel. a kernel bundle 1 AIC + 2AIV, while the whole NPU AI cores run a same program

Tile + MPMD

— PyPTO on NPU

Tile programming style with PTO virtual ISA for a program. One program runs on a specific cores, different programs runs on different cores, scheduled by AI CPU

DSL Design — `@pl.function(level=AIC)` and `@pl.function(level=AIV)` are different programs, not fused kernel.

Runtime — AICPU analyze the dependency of each program and real-time dispatch programs to available cores

Ascend hardware can run divergent code paths per core efficiently.
SIMT/SPMD-shaped tools serialize the divergence. PTO doesn't pay that cost.

Performance Gain with Ascend-Native

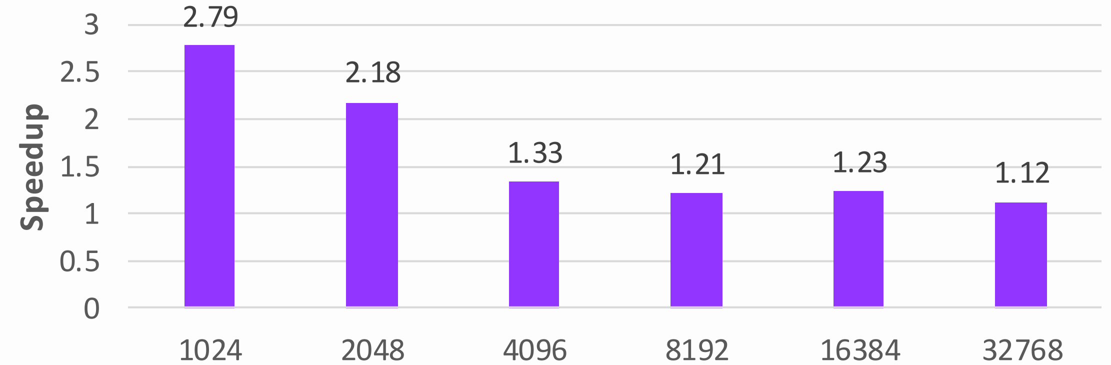
上海创智学院
Shanghai Innovation Institute

L3 **Host / Single Node**
single machine with multiple NPU connected by intra-connection

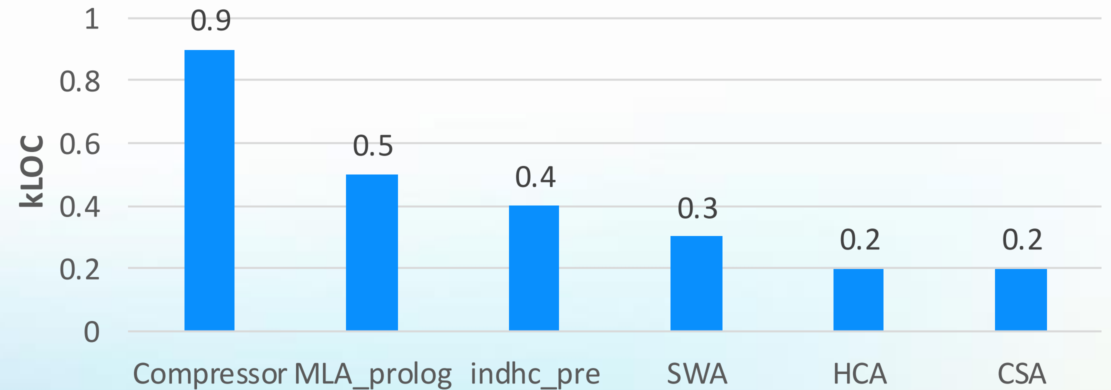
L2 **NPU Chip**
high-bandwidth UB fabric spanning many nodes — Ascend's superpod tier

L1 **NPU Chip Die**
one server / pod inside a rack; tight coupling, high intra-rack bandwidth

L0 **NPU Core / Core Group**
one OS instance managing one or more chips over a host-to-device link



FlashAttention based on PTO-ISA Speedup vs. torch_npu on 910B[1]



Lines of code for DS v4 fused kernel written by PyPTO [2]

1. https://zhuanlan.zhihu.com/p/2035084143629955119
2. https://gitcode.com/cann/cann-recipes-infer/blob/master/docs/models/deepseek-v4/deepseek_v4_pypto_operator_guide.md

03 Lingqu-Native

The Lingqu System — More than Single Machine

上海创智学院
Shanghai Innovation Institute

L7 **Global Coordinator**
top of the hierarchy; entry point for a global program

L6 **Cluster-2 / Cross-rack**
cross-rack tier; contracted bandwidth, multi-hop routing

L5 **Cluster-1 / Superpod**
high-bandwidth UB fabric spanning many nodes — Ascend's superpod tier

L4 **Cluster-0 / Pod**
one server / pod inside a rack; tight coupling, high intra-rack bandwidth

L3 **Host / Single Node**
Single machine with multiple NPU connected by intra-connection

Unified architecture design for super pod

Most stacks collapse host → datacenter into one bucket. We separate Pod (L4), Superpod (L5), and Cross-rack (L6).

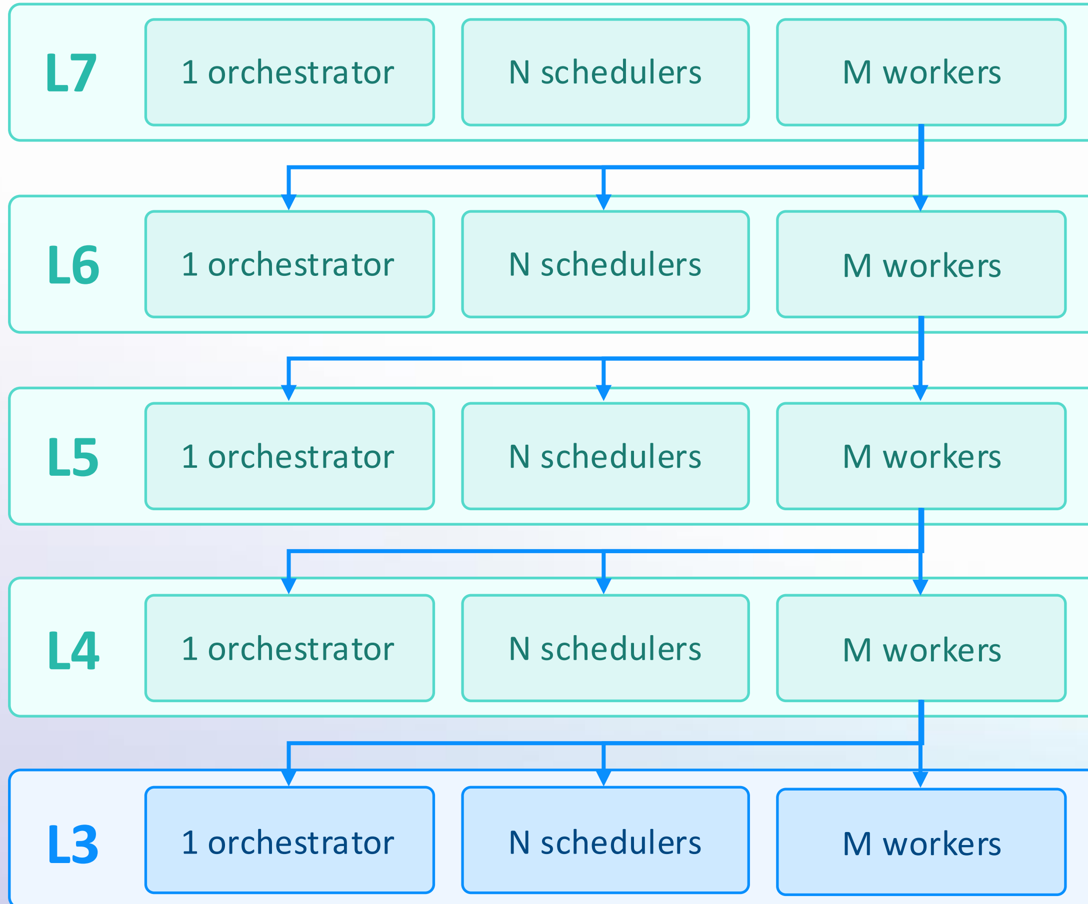
The UB fabric inside a superpod has fundamentally different bandwidth/latency from cross-rack, and the runtime exploits that.

What other distributed ecosystems miss

All current systems have very separate design below the L3 and above L3. Each host own its program and use CCL or global barrier to sync.

Hierarchy in Software — Same Shape with Hardware

上海创智学院
Shanghai Innovation Institute




```
@pl.function(level=pl.Level.CLUSTER_1) # L5 superpod
def superpod_dispatch(...): ...

@pl.function(level=pl.Level.POD,         # L4
             role=pl.Role.ORCHESTRATOR)
def pod_orchestrator(...): ...

@pl.function(level=pl.Level.HOST,        # L3
             role=pl.Role.WORKER)
def host_worker(...): ...
```

- Orchestrator: a single-thread program to run the logic
- Worker: run specific tasks on the current level
- Scheduler: find available resource for workers

UB Data Services — the Cluster's Data Plane

上海创智学院
Shanghai Innovation Institute

Collective Communication

— L0 – L6

Collective Communication across NPUs on different pod or rack. Only access to NPU but not host memory or SSD

KV Cache Pooling

— L1 – L5

Use CPU or SSD to store the prfix KV Cache, NPU can directly access the host memory or SSD within a superpod

Checkpoint Manage

— L2 – L6

Load weight or save checkpoint to SSD, including distributed checkpoint, async checkpoint saving and training resume

HCCL

Mooncake

Torch Distributed

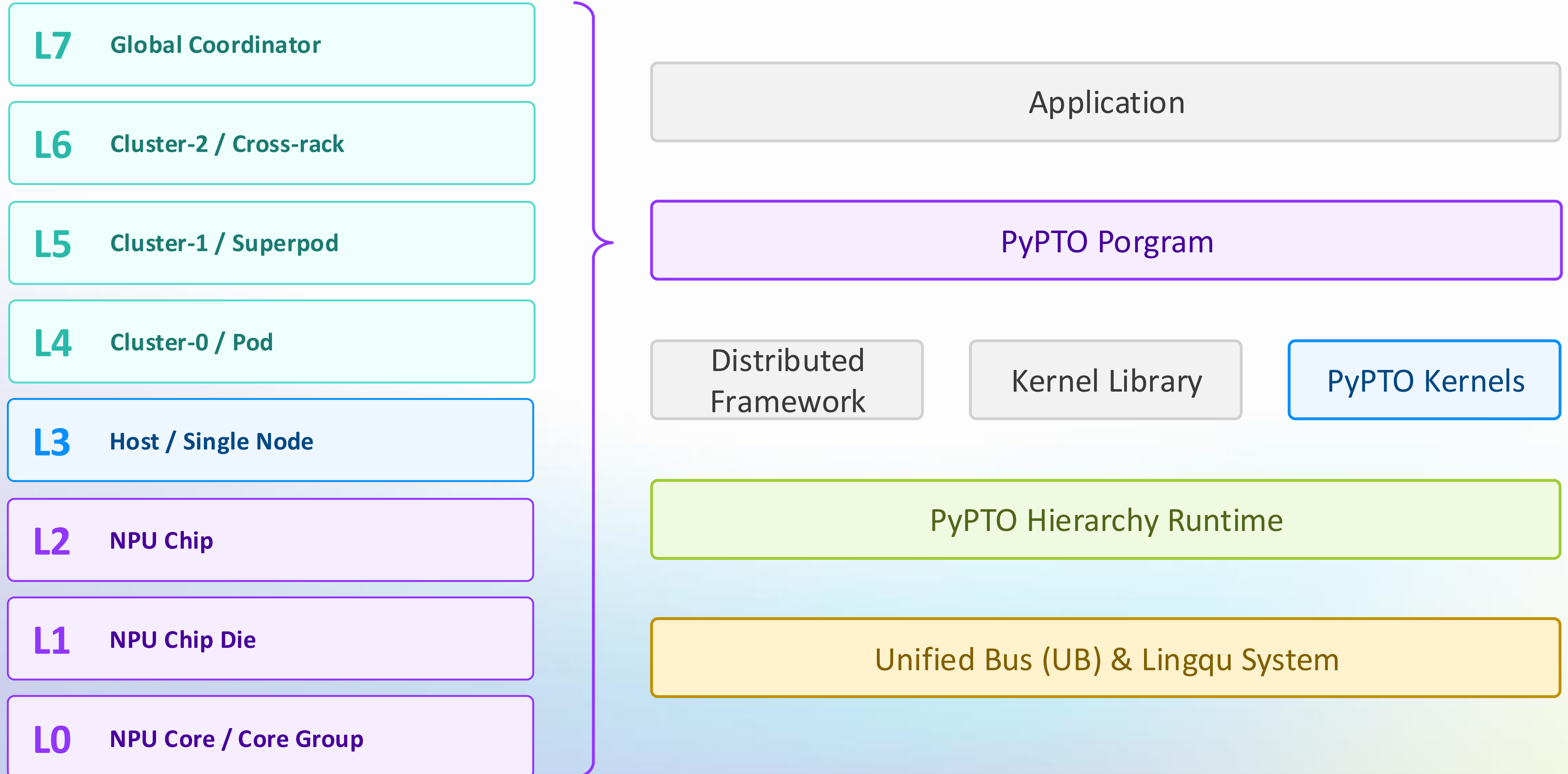
PyPTO Hierarchy Runtime

Unified Bus (UB) & Lingqu System

Higher-level capabilities — KV cache, persistence, metadata services — compose from these. No new abstractions needed.

Control the Whole Cluster with Unified Program

上海创智学院
Shanghai Innovation Institute



04 AI-Native

AI-Native —Built with AI, Built for AI

上海创智学院
Shanghai Innovation Institute

Pillar A — Built with AI

PyPTO, PTO-ISA, PTOAS, PTO runtime — agent-accelerated development.

rules · skills · reviews · testing

New op definitions, new IR passes, new backend bring-up — all faster than the pre-agent baseline.

compounds

Pillar B — Built for AI

Agents are first-class users, not just code-writers.

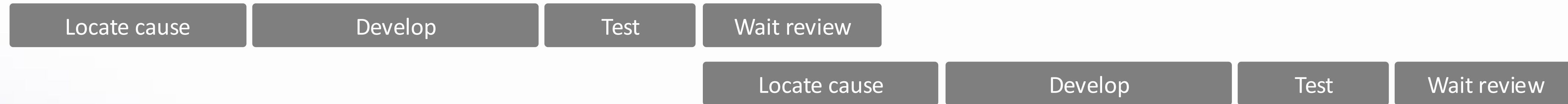
python DSL · profiling · IR dump

End-to-end debugging, performance improving, translating from other DSL — agent can do most of works that human can

Pillar A — Built with AI: AI-Accelerated Development

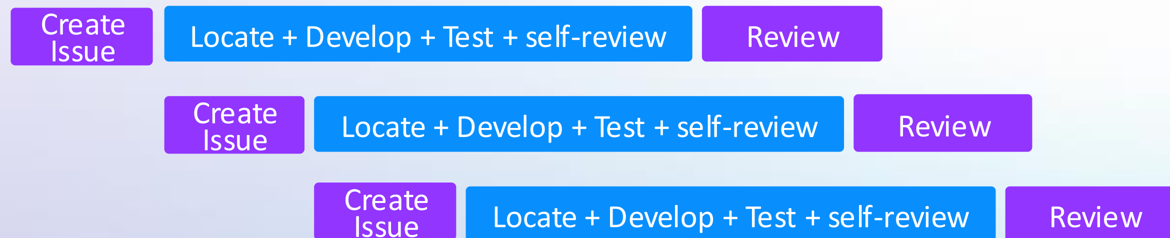
上海创智学院
Shanghai Innovation Institute

Human-Only Loop



Serial · expert-bound · weeks to a new feature

Human + Agent loop



Improve development efficiency by over **5×**

Parallel · structured · hours to a new feature

Pillar A — Built with AI: Controlled Agent Loop

上海创智学院
Shanghai Innovation Institute

```
pypto/.claude/
├── CLAUDE.md                      ← project entry point
├── rules/
│   ├── core-development.md        ← code quality baseline
│   ├── cross-layer-sync.md        ← C++ / bindings / stubs aligned
│   ├── documentation.md           ← doc-first workflow
│   ├── first-principles.md        ← IR is the source of truth
│   ├── pass-complexity.md         ← passes ≤ O(N log N)
│   ├── pass-doc-ordering.md       ← pass docs match execution order
│   ├── plans-and-proposals.md     ← plans show concrete code
│   ├── problem-handling.md        ← handling blockers and known issues
│   ├── python-style.md            ← Python style guide
│   ├── testing-and-examples.md    ← tests live in tests/
│   ├── worktree-builds.md         ← build inside the worktree
│   └── ...
└── skills/
    ├── add-op/                    ← add a new IR op
    ├── code-review/               ← review code changes
    ├── compare-codegen/           ← diff codegen outputs
    ├── create-issue/              ← file a online issue
    ├── fix-issue/                 ← implement an issue
    ├── fix-pr/                    ← address PR feedback
    ├── git-commit/                ← commit changes
    ├── create-pr/                 ← open a PR
    ├── testing/                   ← build and run tests
    └── ...
```

Well-designed agent skills and rules

```
pypto/tests/                       ← files=218   tests=4371
├── st/                            ← files= 55   tests= 211
│   ├── codegen/
│   ├── distributed/
│   ├── examples/
│   └── runtime/
└── ut/                            ← files=163   tests=4160
    ├── backend/
    ├── codegen/
    ├── core/
    ├── debug/
    ├── ir/
    ├── jit/
    ├── language/
    ├── pass/
    └── runtime/
```

More than 4000+ comprehensively tests

Codex

Gemini

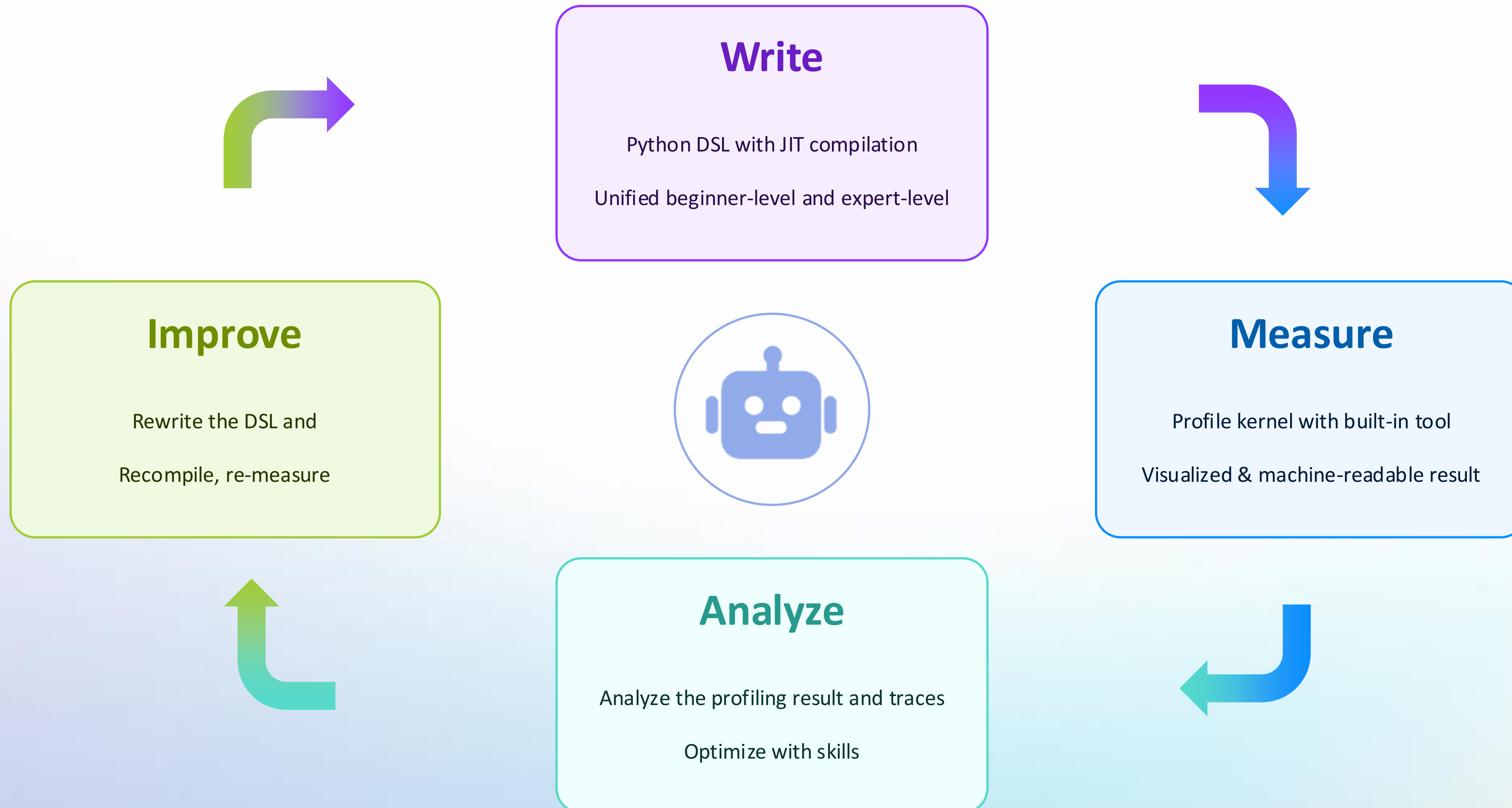
CodeRabbit

Copilot

Cross-Agent code review pipeline

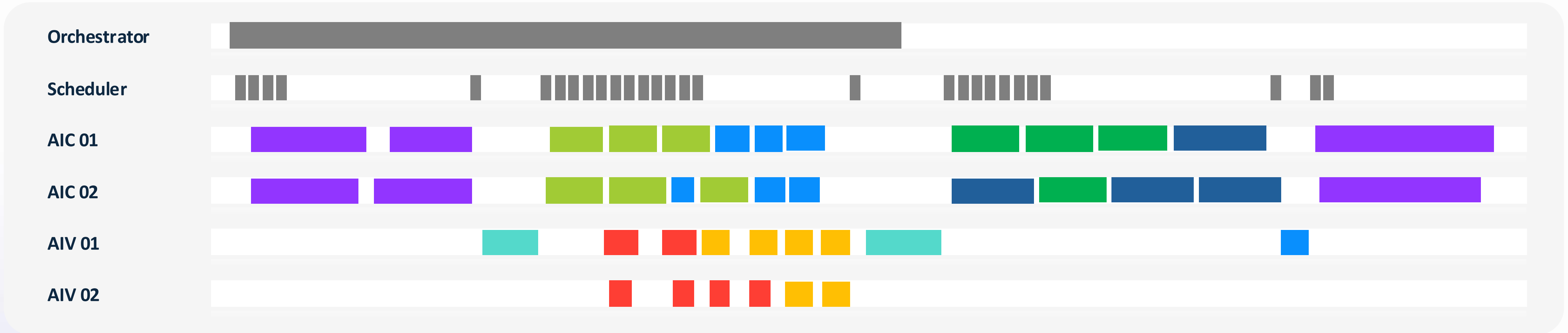
Pillar B — Built for AI: the agent's optimizing loop

上海创智学院
Shanghai Innovation Institute



Swimlane Graphs — One Trace, Two Consumers

上海创智学院
Shanghai Innovation Institute



What a human sees — Interactive visualized graph

- AIC is the bottleneck
- Attention, MLP projection takes most of the time
- Still optimization for task dispatching and organization

What an agent reads — Text-based per-core trace


```
[
  { "lane": "AIC00", "util": 0.95, "tasks": [...], ... },
  { "lane": "AIC01", "util": 0.93, "tasks": [...], ... },
  { "lane": "AIV01", "util": 0.42, "tasks": [...], ... },
]
```


One trace. Two consumers. Equal-citizen by design — not a human-only debug tool with a coincidental JSON dump.

上海创智学院
Shanghai Innovation Institute

Summary

Tile Programming – Balanced Sweet Point

上海创智学院
Shanghai Innovation Institute

Tensor level

algorithm developers — productive entry point

Tile level — load-bearing

performance experts — tile shape, memory placement, pipeline events

Element level

system developers — fully control the hardware behavior

PyPTO Tile Programming

PTO Tile Assembly (.pto)

PTO-ISA (Virtual Tile ISA)

MPMD Runtime (Tile Runtime)

Tile Programming

Ascend-Native

Lingqu-Native

AI-Native

Ascend-Native Design to Utilize the Hardware

上海创智学院
Shanghai Innovation Institute

Pillar 1

Cross-device, cross-generation

One PTO-ISA across A2 / A3 / A5 even Kirin NPU; one kernel source survives generation transitions.

Pillar 2

Native primitives exposed

AIC, AIV, GM/Vec/Mat, TPUSH/TPOP pipeline events, Auto/Manual mode — all addressable in PyPTO.

Pillar 3

MPMD-native runtime

MPMD Runtime co-schedules independent programs per core; PyPTO calls this 'MPMD Execution Scheduling.'

Tile Programming

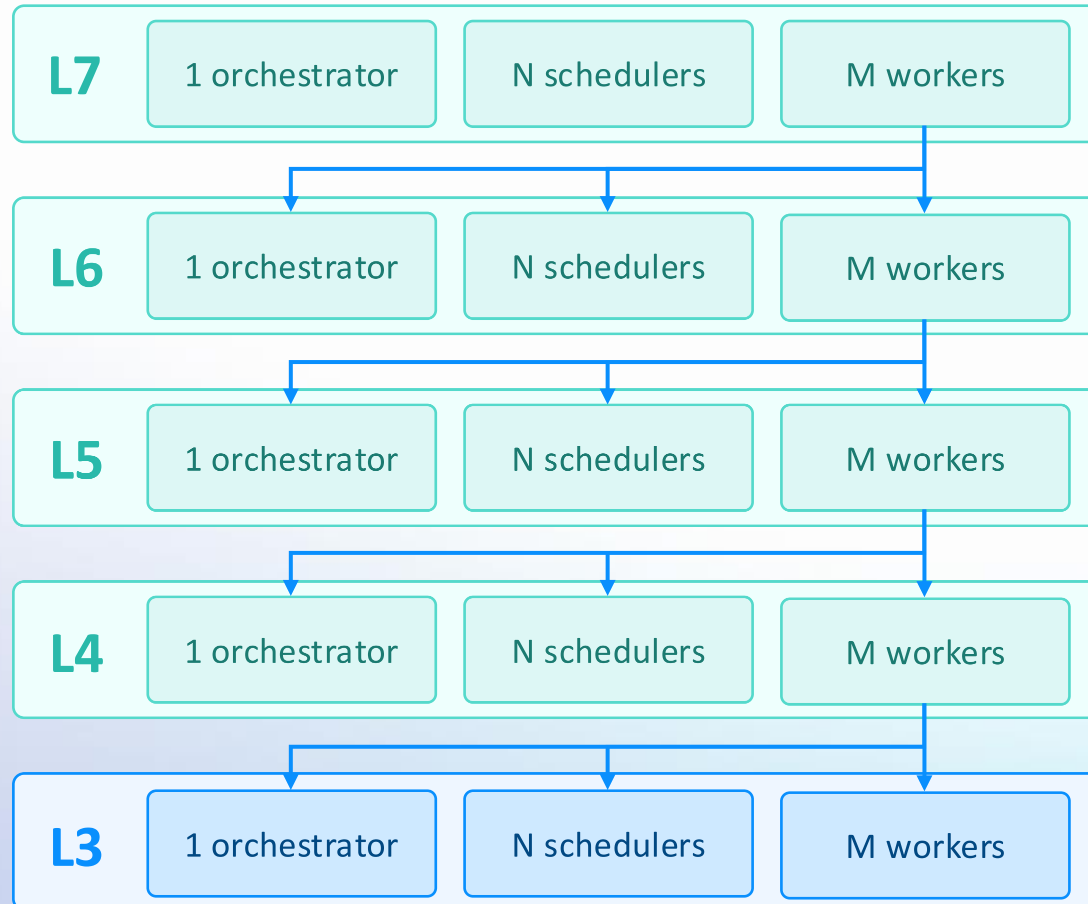
Ascend-Native

Lingqu-Native

AI-Native

Unified Program for the Whole Cluster

上海创智学院
Shanghai Innovation Institute



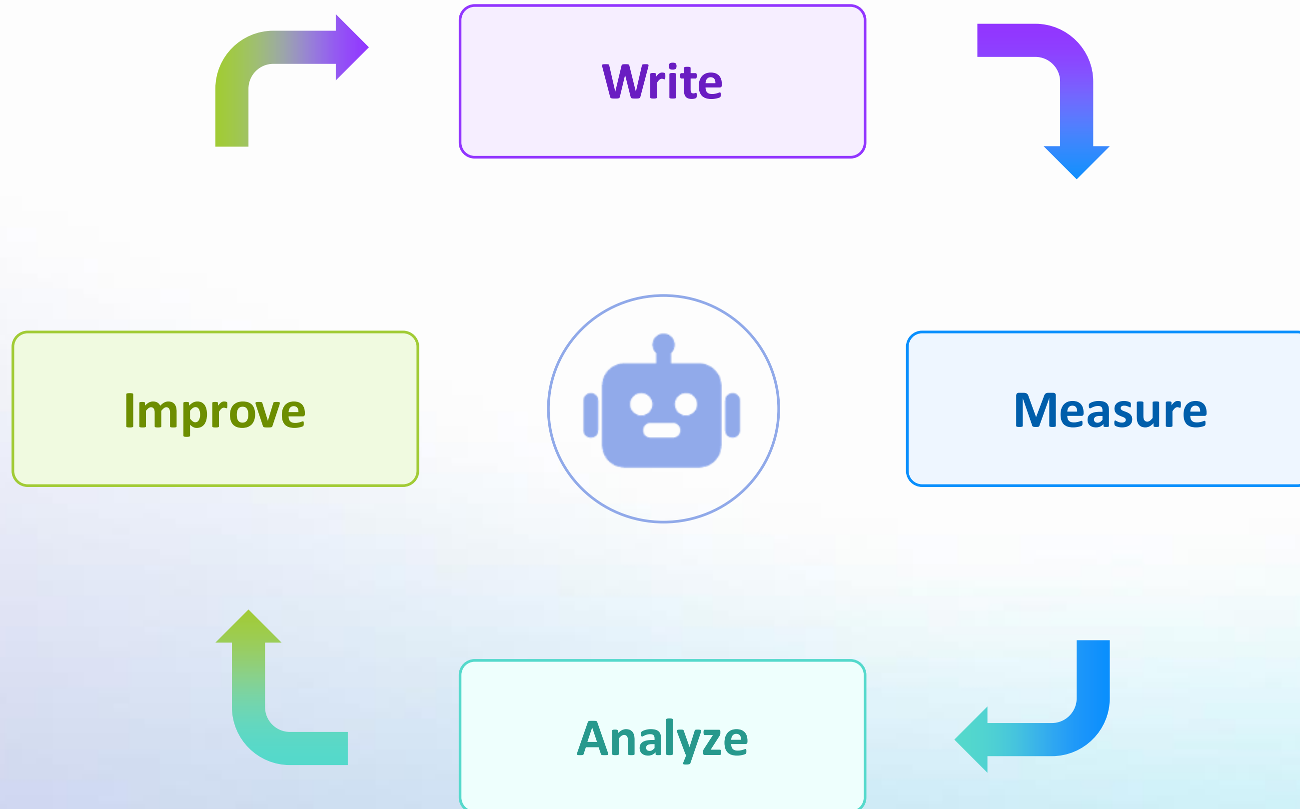
Tile Programming

Ascend-Native

Lingqu-Native

AI-Native

Built with AI and Built for AI



Tile Programming

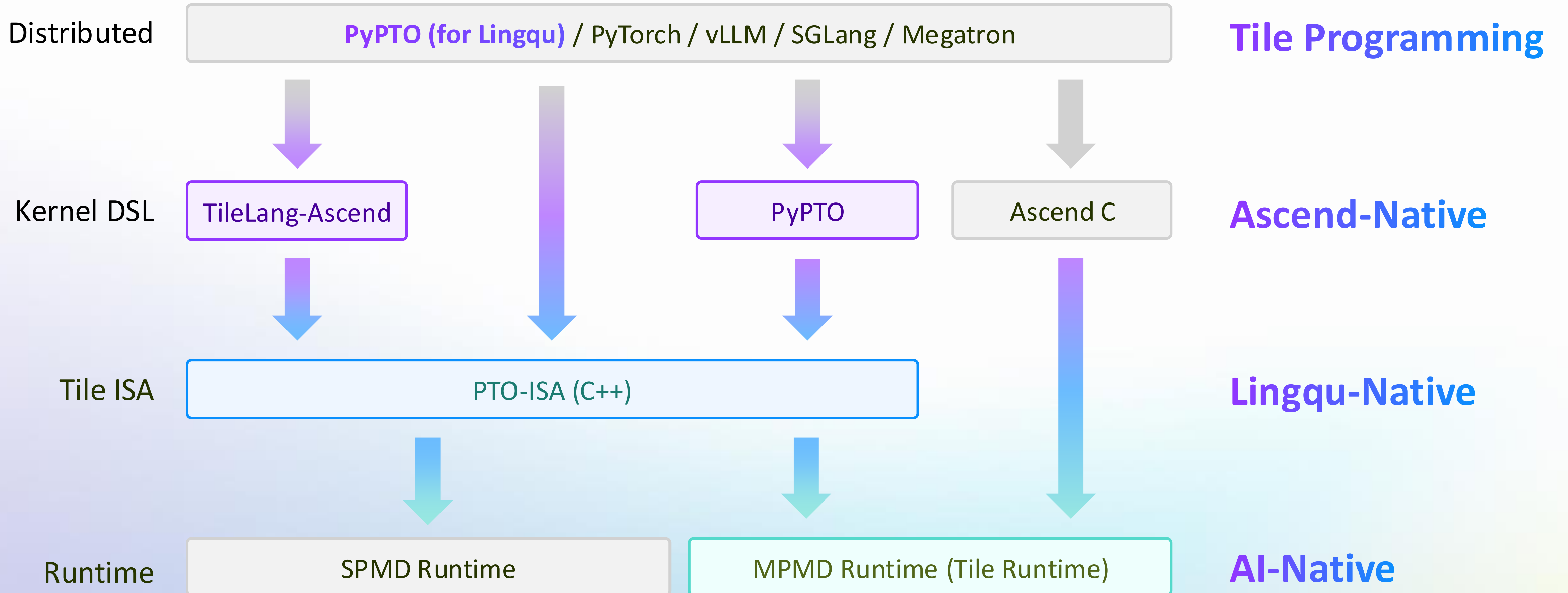
Ascend-Native

Lingqu-Native

AI-Native

Building Tile Programming Ecosystem for Ascend

上海创智学院
Shanghai Innovation Institute



上海创智学院
Shanghai Innovation Institute

Thanks for Listening

Siyuan Feng | 2026.05.12

SHANGHAI INNOVATION INSTITUTE