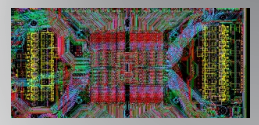
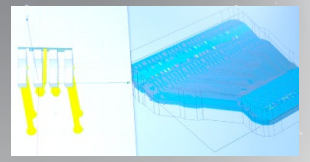
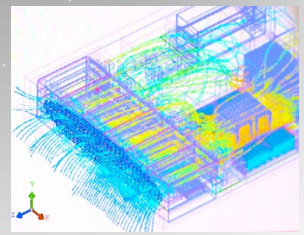
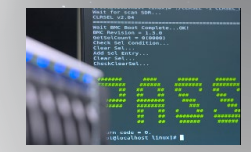
Scaling AI Inference Infrastructure from Silicon to System



From System Design to Data Center

Design	Validation	Production	Service
• Board • Mechanical • Signal integrity simulation • Thermal simulation • BIOS • OpenBMC	• HW and FW function • EA • Thermal • Power • EMI and Safety • Reliability	• System assembly • L10/L11 validation • Rack integration	• Firmware customization • Firmware update • On-site support





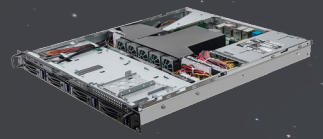


Built for AI Infrastructure — GPU to CPU

Mainstream


4U36L6E


2U12L


1U4L4E


1U2S

All-Flash

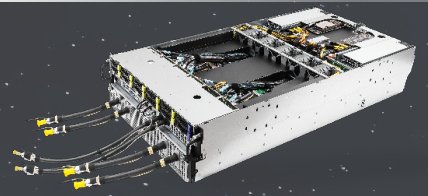

2U16E3S

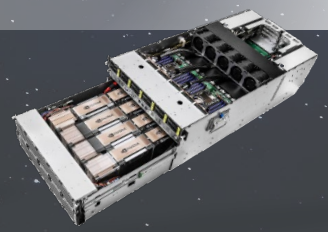

2U24E


1U24E1S


1U12E

AI GPU

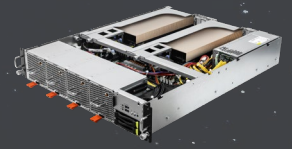

4U8X/4U8M (SXM/OAM)


6U8X/6U8M (SXM/OAM)


MGX Rack


4U8G/4U10G


4UMGX


2U2G

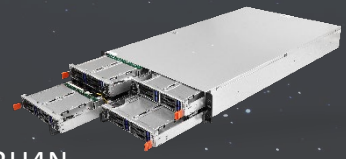

2UMGX


1U1G


2U1G

Multi-Node

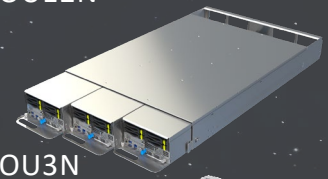

4U18N


2U4N

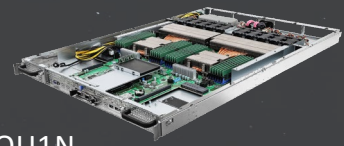

1U2N2G

OCP


4OU12N


2OU3N


1OU3N

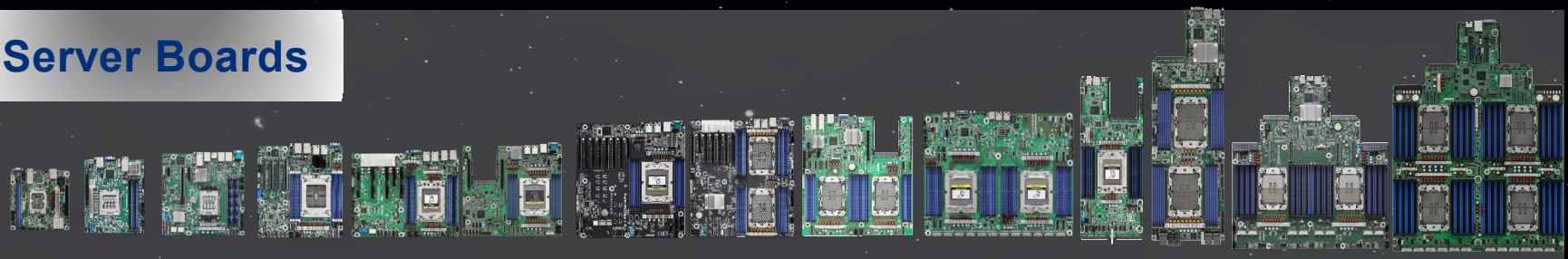

1OU1N

Storage

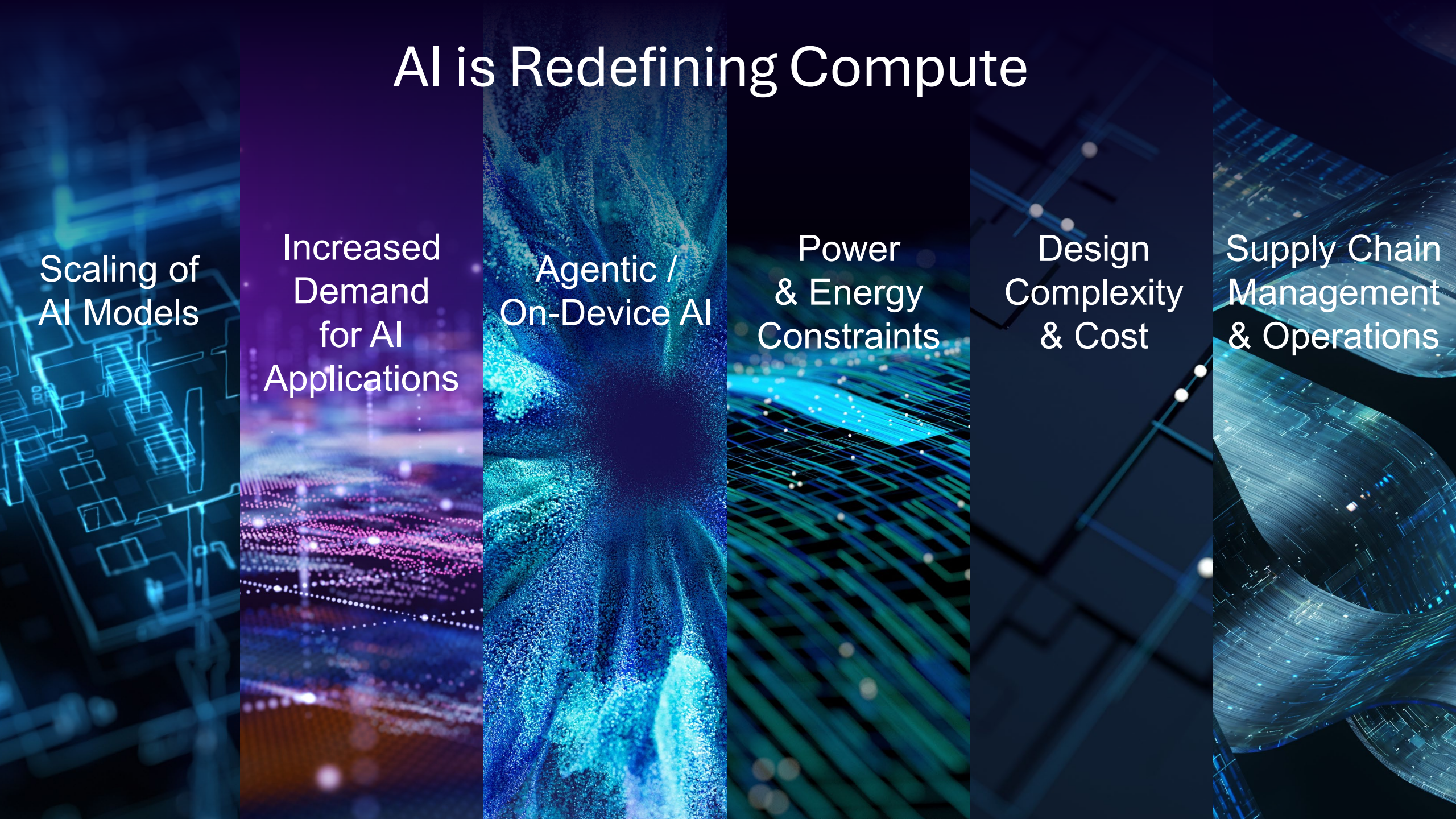

1U12L


4U60L

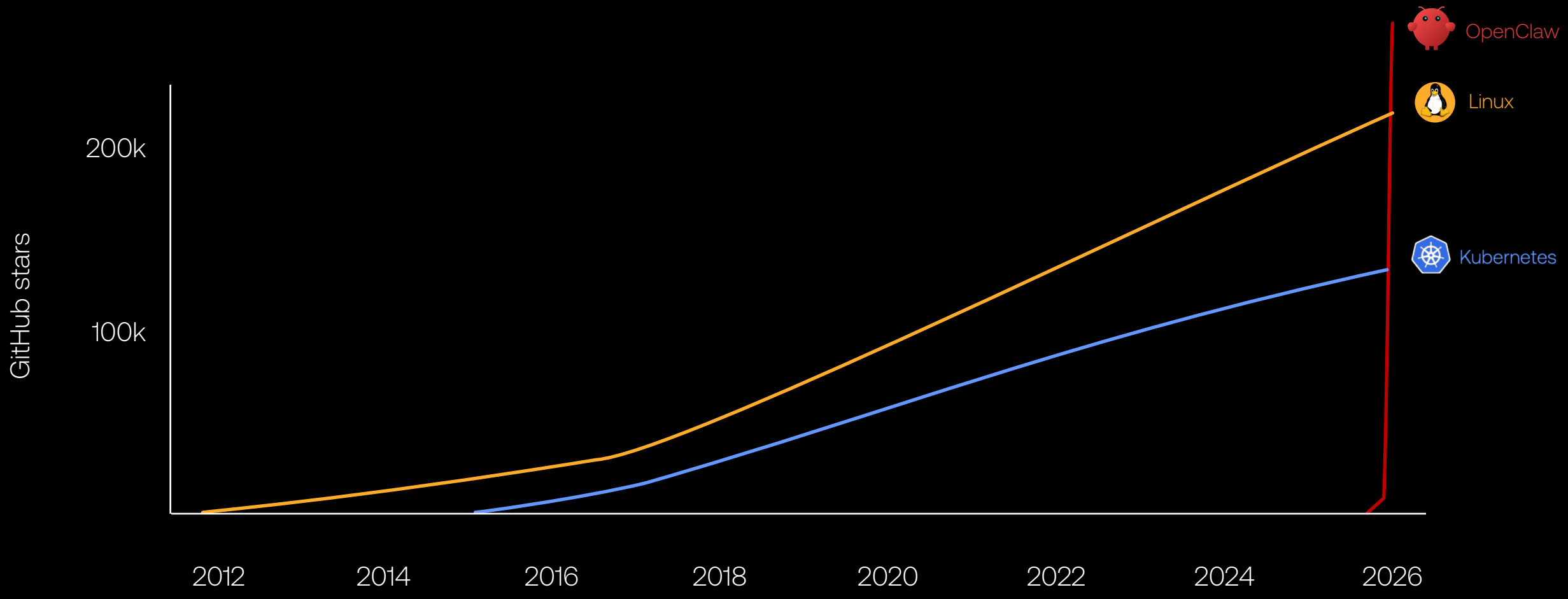
Server Boards



AI is Redefining Compute

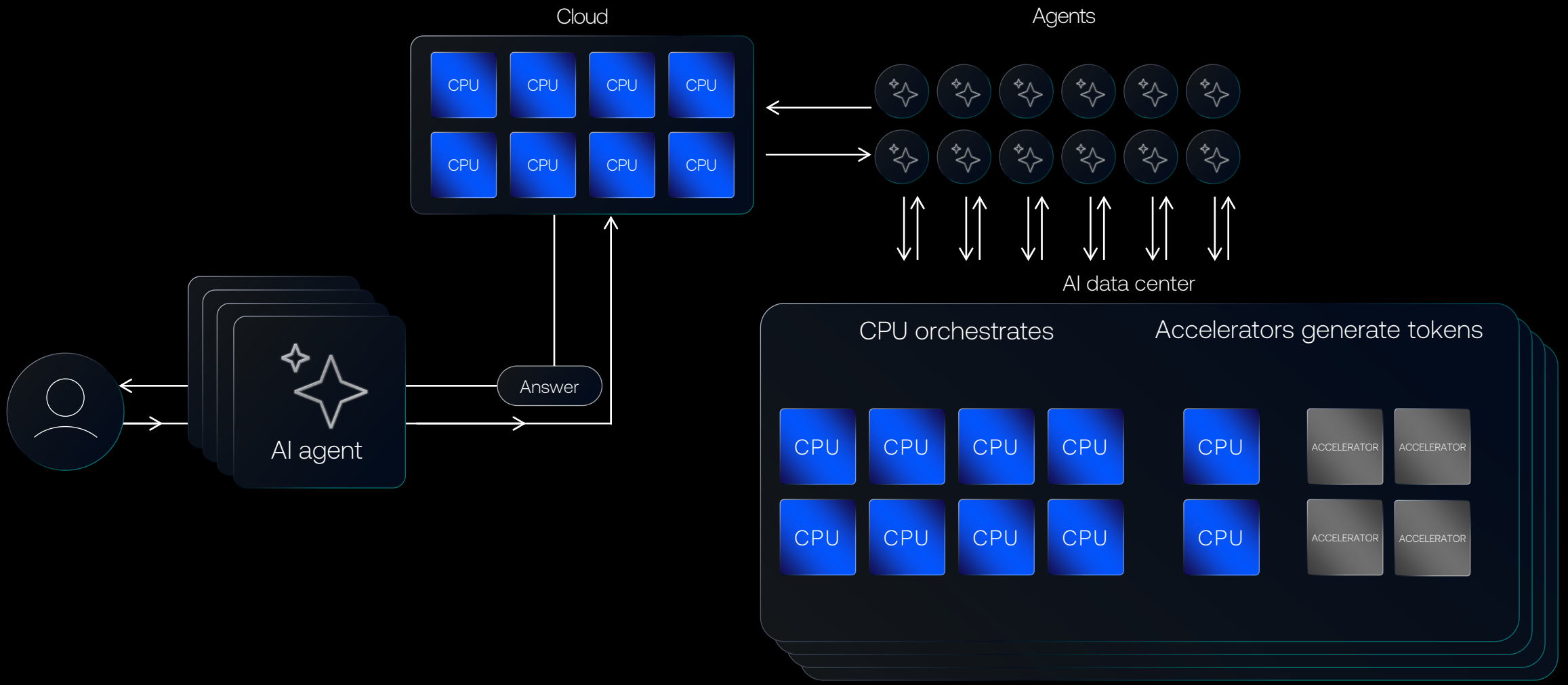


Agentic explosion



*Based on GitHub adoption trends

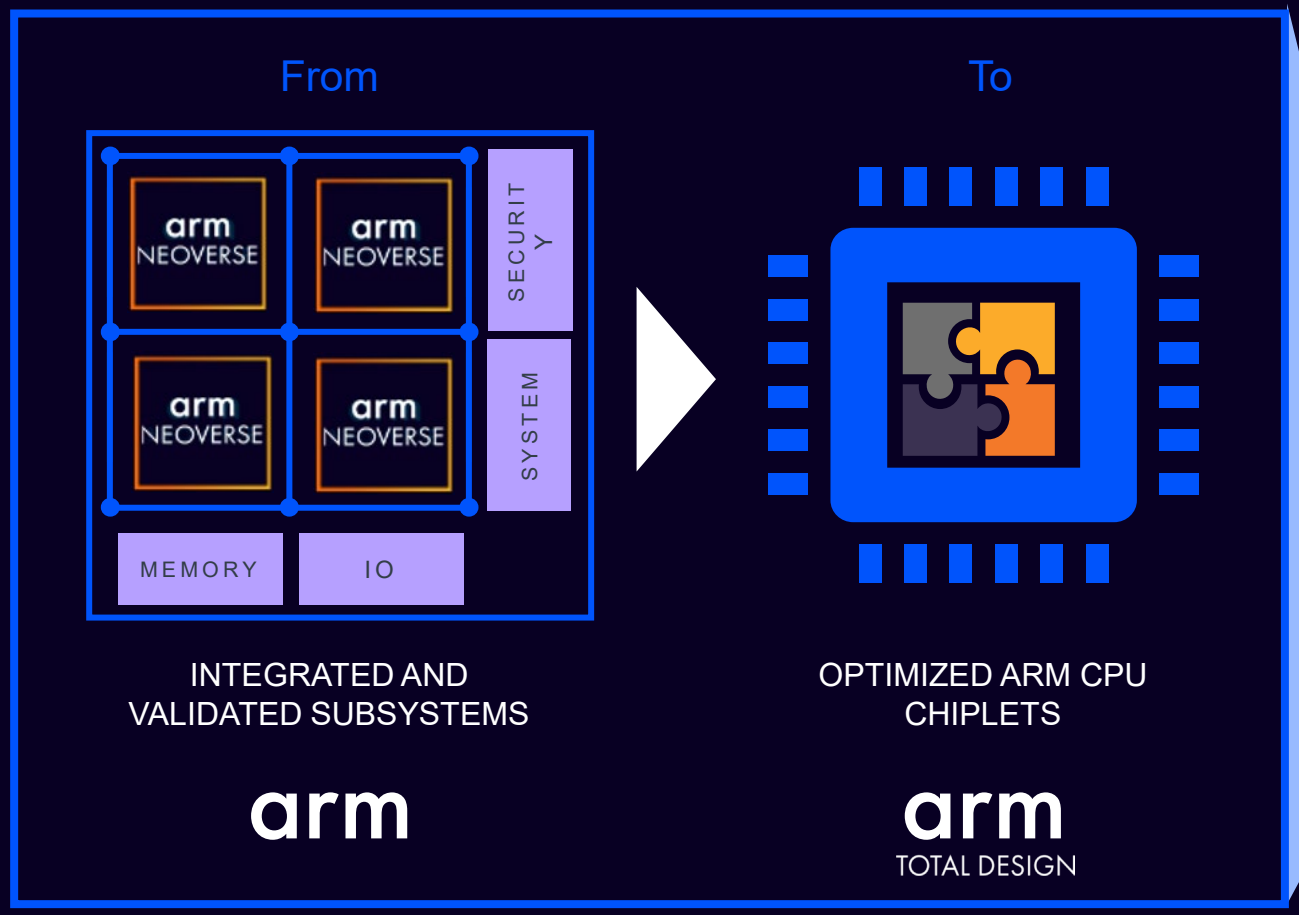
3-5X more CPUs needed to balance agent flow

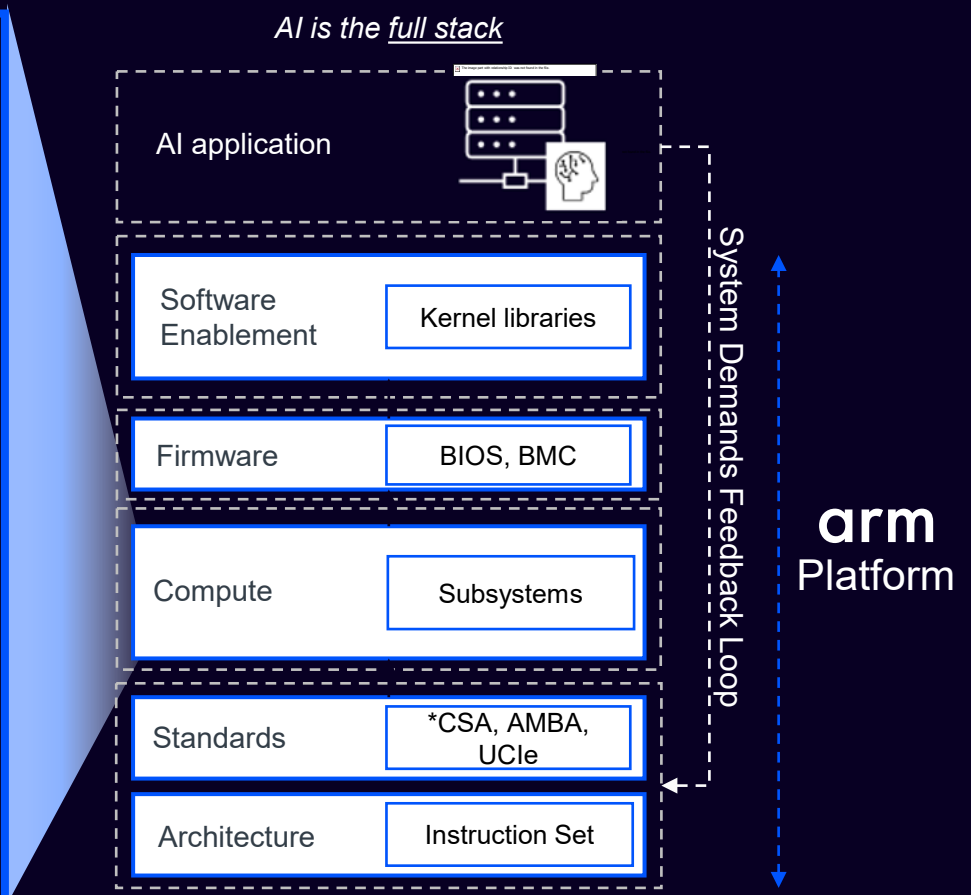




Public © 2025 Arm

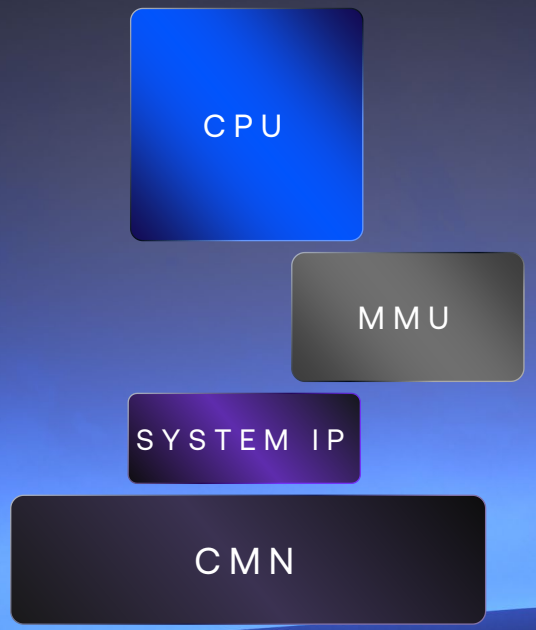
Arm Platform for AI System Delivery





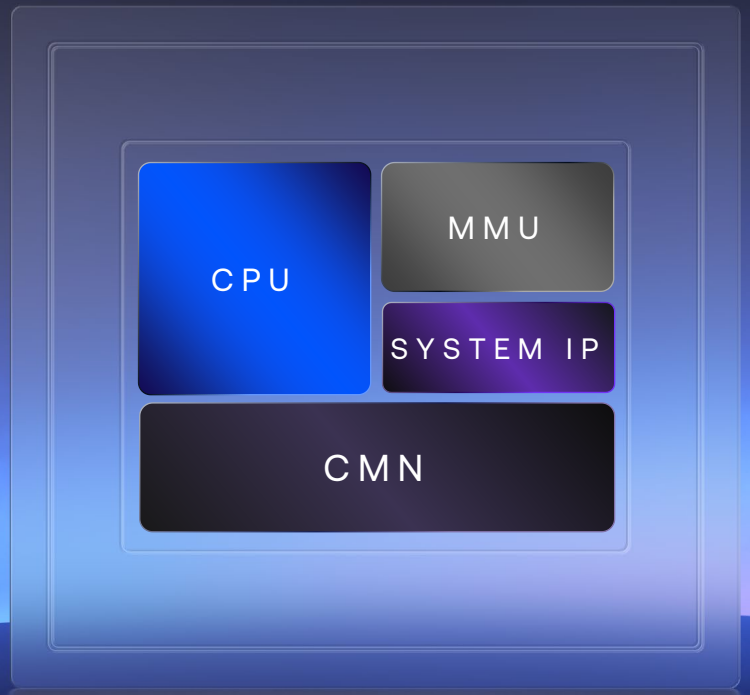
One platform: multiple entry points

Build your own



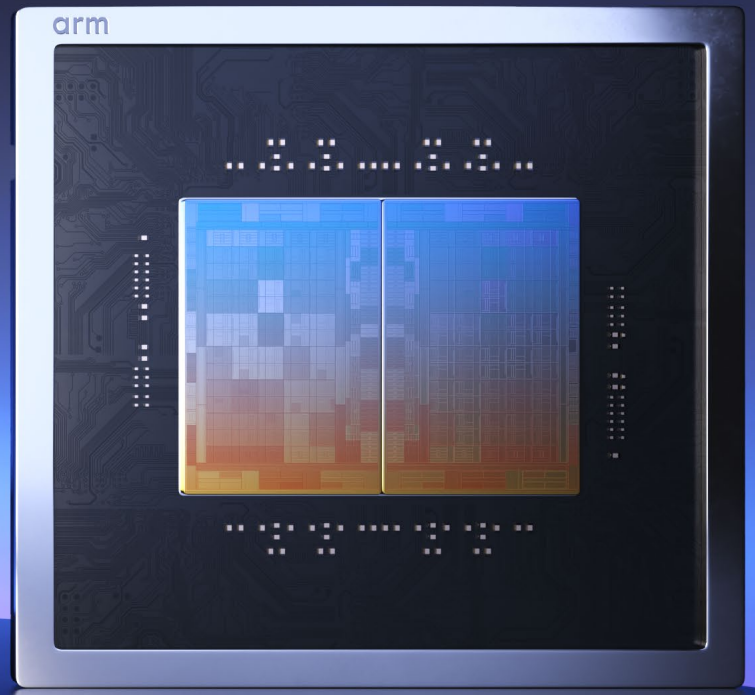
IP

Accelerate your design



Compute Subsystems (CSS)

Deploy faster

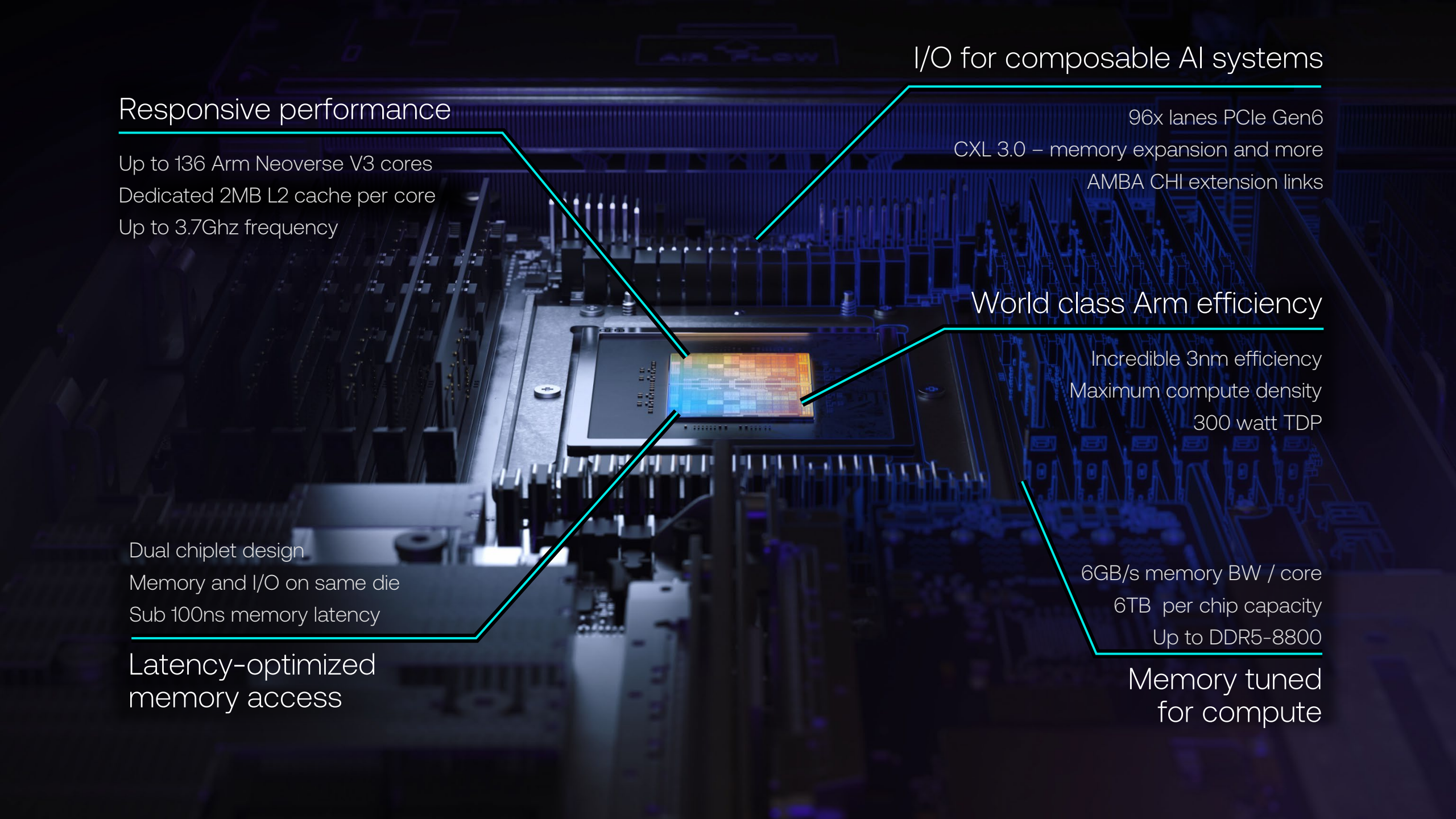


Silicon

arm AGI CPU

Public © 2025 Arm


arm
arm AGI CPU
World's most efficient agentic CPU
Performance
Scale
Efficiency



Responsive performance

Up to 136 Arm Neoverse V3 cores
Dedicated 2MB L2 cache per core
Up to 3.7Ghz frequency

I/O for composable AI systems

96x lanes PCIe Gen6
CXL 3.0 – memory expansion and more
AMBA CHI extension links

World class Arm efficiency

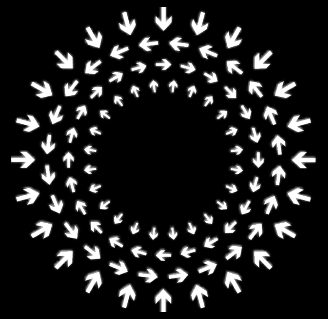
Incredible 3nm efficiency
Maximum compute density
300 watt TDP

Latency-optimized memory access

Dual chiplet design
Memory and I/O on same die
Sub 100ns memory latency

Memory tuned for compute

6GB/s memory BW / core
6TB per chip capacity
Up to DDR5-8800



OPEN Compute Project®

+

arm

Arm contributions across the full stack

Chiplet enablement

Server standardization

Server designs



Diverse silicon ecosystem

- Foundation Chiplet System Architecture (FCSA)
- Modular integration of compute, memory, and I/O
- Reduced design complexity for custom silicon

Software just works on Arm servers

- Standardized server architecture (SBSA, SBMR)*
- Compliance and validation framework (ADAC)
- Built-in system management and security

** Contributions in progress*

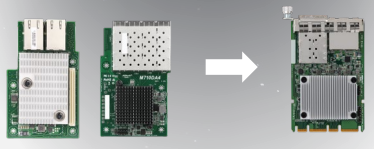
Helping OxMs launch Arm servers faster

- Pre-defined server hardware specifications^
- Aligned with Meta Yosemite V5 rack design
- Firmware development frameworks

^ Future contribution

ASRock Rack OCP Journey

OCP NIC 3.0



OCP Server

ORv2



ORv3

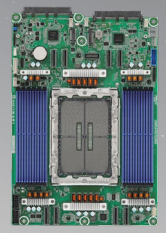




DC-MHS

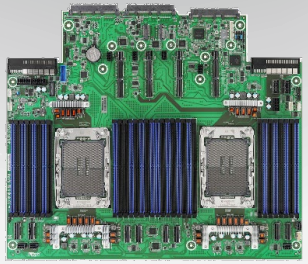
M-DNO
x86
ARM

DC-SCM
AST 2700
AST 2600

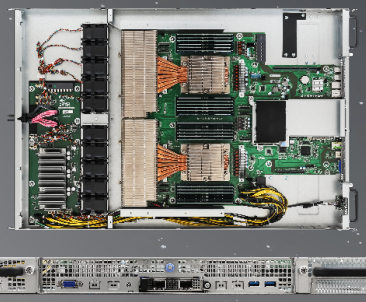




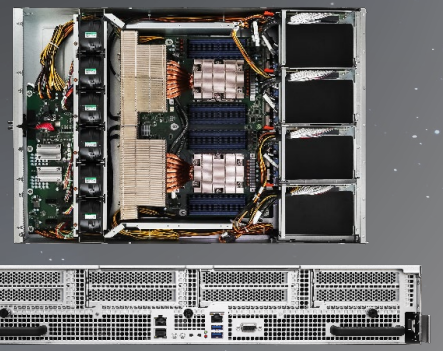
M-FLW



1OU1N



2OU1N



1OU3N





AMD EPYC9005
AMD EPYC8005
Intel Xeon 6

2OU3N





AMD EPYC9005
AMD EPYC8005
Intel Xeon 6

4OU12N





AMD EPYC4005

2OU3N3G



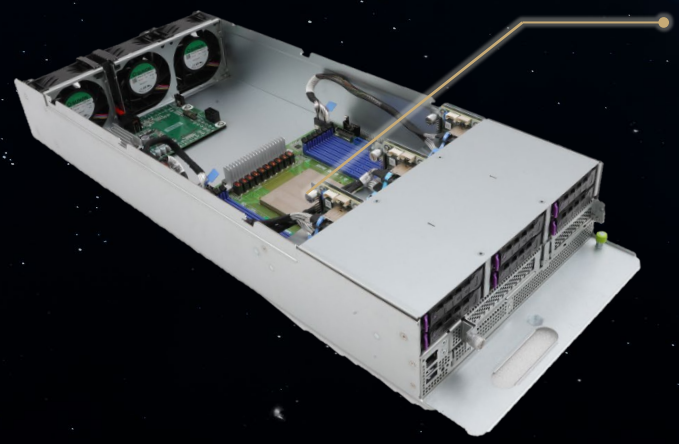


AMD EPYC9005
Intel Xeon 6

44 OU 36 Nodes

Open Rack V3
EDK II UEFI
OpenBMC





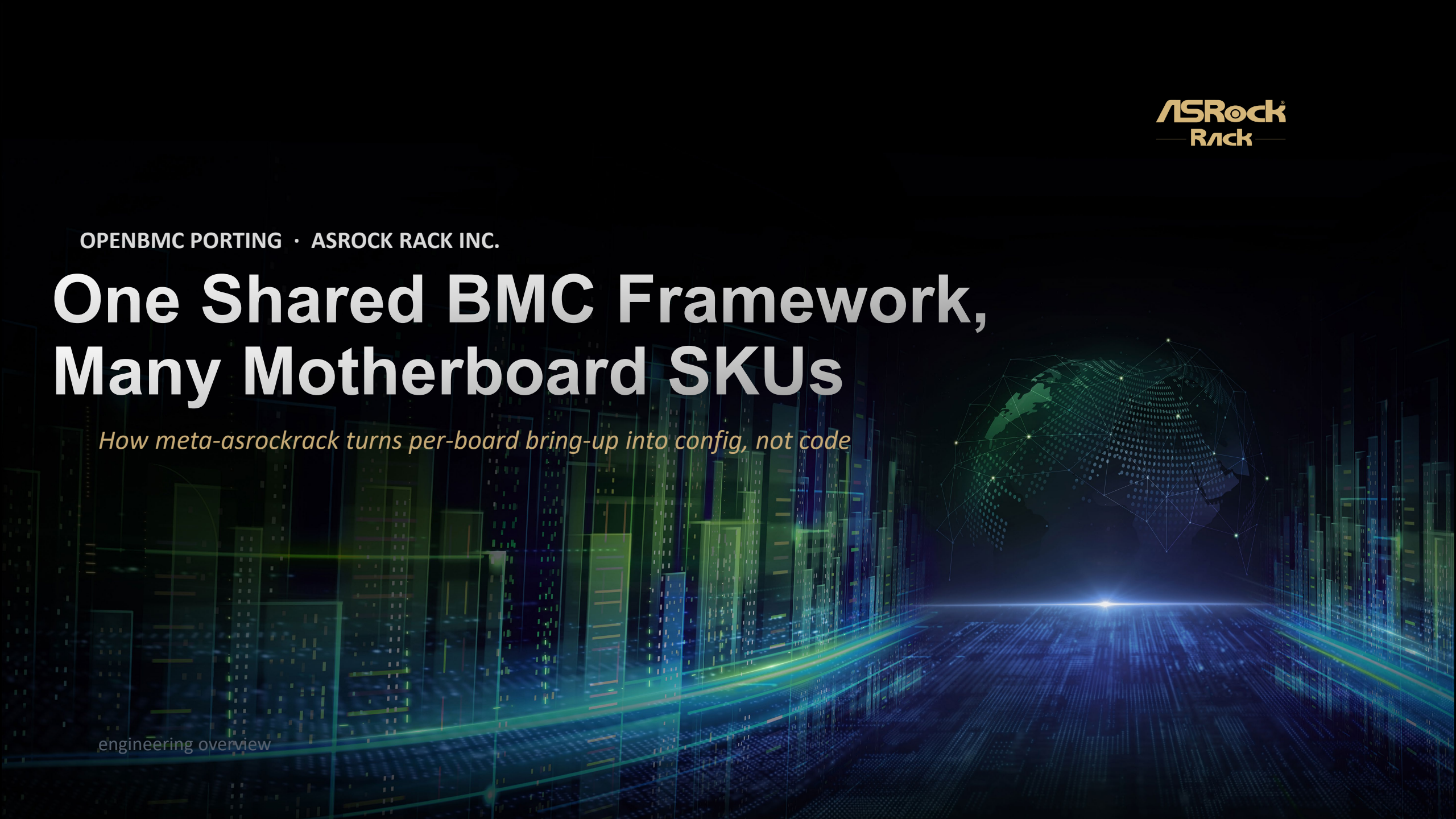
2OU2N-ARM

Arm AGI CPU up to 136 cores
12 DDR5 RDIMM 8000MT/s
6 E3.S (PCIe 6.0)
OCP NIC 3.0 (PCIe 6.0)

OPENBMC PORTING · ASROCK RACK INC.

One Shared BMC Framework, Many Motherboard SKUs

How meta-asrockrack turns per-board bring-up into config, not code

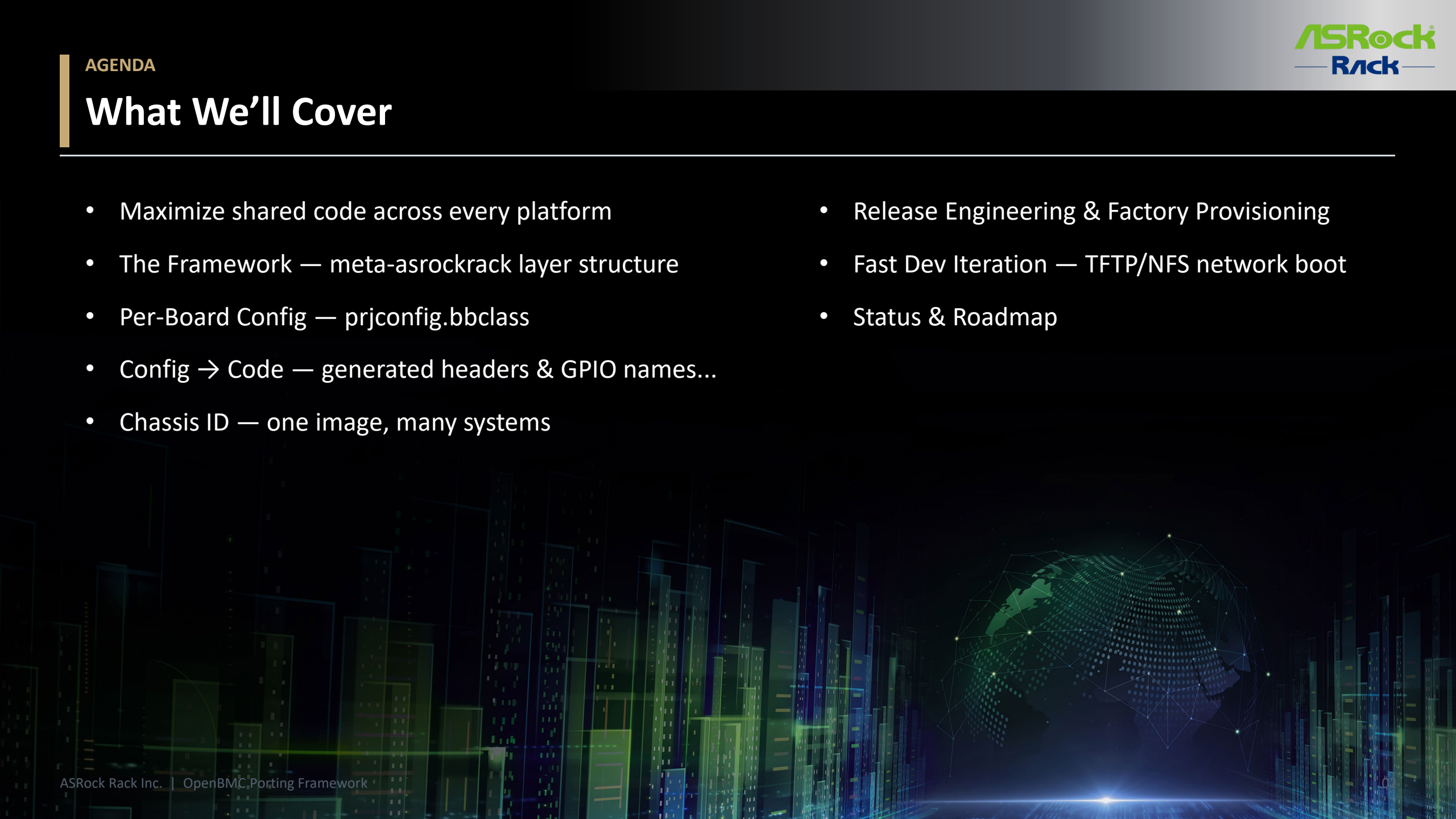


engineering overview

AGENDA

What We'll Cover

- Maximize shared code across every platform
- The Framework — meta-asrockrack layer structure
- Per-Board Config — prjconfig.bbclass
- Config → Code — generated headers & GPIO names...
- Chassis ID — one image, many systems
- Release Engineering & Factory Provisioning
- Fast Dev Iteration — TFTP/NFS network boot
- Status & Roadmap



THE GOAL

Maximize Shared Code Across Every Platform

- Motherboard SKUs in the tree today —

  GNR, Genoa, Turin, AM5, EPYC, W680,

  AST2500, AST2600, AST2700 and Nuvoton NPCM845

- Even the same motherboard ships in multiple physical system configurations (chassis, backplane, sensor, fan, led).
- Every new board traditionally meant re-doing BMC bring-up almost from scratch.
- RD headcount doesn't scale with SKU count —

  it has to be the other way around

n

board overlays,
4 SoC platforms

1

shared meta-common pattern, every board

How:

Push per-board differences down into some config files, not a forked codebase.



meta-asrockrack: One Layer, Many Boards

meta-asrockrack
Top-level BSP layer for all boards

meta-ast2500 / 2600 / 2700 / npcm845
Split only where the SoC or vendor differs

meta-common
Shared services, recipes, u-boot/kernel, image policy

meta-<model>
Per-board overlay: prjconfig.bbclass + dts + config deltas

meta-asrockrack layer tree

```
meta-asrockrack
├── meta-ast2500
├── meta-ast2600
│   ├── meta-common   (SoC-specific)
│   │     linux-aspeed, u-boot-aspeed-sdk, ...
│   ├── meta-am5d4id-2t
│   ├── meta-genoad8hm3
│   ├── meta-gnr2d32mgx
│   ├── meta-armd12m3
│   └── ... more boards
├── meta-ast2700
├── meta-npcm845
└── meta-common   (shared by all SoCs)
      ipmi-oem, fancontrol, inventory,
      dbus-sensors, multi-node, autosnmp, ...
```

New SKU checklist: add a meta-<model> overlay, modify prjconfig.bbclass, wire the dts, write sensors, leds config file.

prjconfig.bbclass: Every Board Knob, One File

- Every hardware difference for a board lives in exactly one file.
- EEPROM layout, fan/PWM map, GPIO pins, BIOS-update behavior, dual-image support, RAID bus mapping, SNMP config...
- Same shared services (platform-control, ipmi-oem, u-boot) read this file through every board.
- Porting a new board = filling in this file, not writing new services.

meta-<model>/classes/prjconfig.bbclass

```
MODEL_NAME ?= “ARMD12M3"
VER_MAJOR ?= "1"
VER_MINOR ?= "0"

EEPROM_I2C_BUS ?= "0"
EEPROM_I2C_ADDRESS ?= "0x57"
EEPROM_CHASSIS_ID_OFFSET ?= "0x3f60"

NUM_FAN_SUPPORTED ?= "8"
FAN1_DEFAULT_DUTY ?= "30"  ...

#BIOS update function
BIOS_SPI_GPIO_BMC_LEVEL ?= "1"
BIOS_UPDATE_FORCE_HOST_OFF ?= "1"

PWMIC_FW_I2C_BUS ?= "6"
PWMIC_FW_SLAVE_ADDR ?= "0x60"
...
inherit prjcommon
```

prjcommon.bbclass Turns Bitbake Vars into C++ Headers

- The bbclass walks every variable at build time.
- Auto-generates prjconfig.hpp:

  #define CONFIG_<VAR> <value>

- Every shared C/C++ service and u-boot module #includes it — config and code can never drift apart.

generated: include/prjconfig.hpp

```
// Auto-generated by prjconfig.bbclass

#define CONFIG_EEPROM_I2C_BUS 0
#define CONFIG_NUM_FAN_SUPPORTED 8
#define CONFIG_FAN1_DEFAULT_DUTY 30
#define CONFIG_BIOS_UPDATE_FORCE_HOST_OFF 1
#define CONFIG_GPIO_MFG_MODE 0x...
```

One bitbake variable,
One build step,
Zero hand-maintained header files per board.

The Board .dts Is the Source of Truth for GPIO, SPI, MAC...

- Parses gpio-line-names straight out of the board's kernel .dts.
- Auto-generates GPIO_<NAME> bitbake variables — no separate GPIO table to keep in sync.
- Give warning if missing GPIO definition in dts for enabled features.
- Retrieve SPI/MAC used by the platform to update the header file to build the source.

board .dts

```
&gpio0 {
  gpio-line-names =
    "BMC_MFG_MODE_L", ...;
};

→ GPIO_BMC_MFG_MODE_L = "GPIOA3"
```

Single source of truth: the pin name lives once, in the dts that already describes the hardware.

Chassis ID: One Image, Many System SKUs

- Chassis ID is written into an EEPROM field on the assembly line.
- u-boot reads it and picks the matching device tree config to boot.
- The BMC service matches it against config.json, runs the matching sku-manager script, and publishes SystemId over D-Bus for every other service to consume.

meta-genoad8hm3/.../platform-control/config.json

```
{ "chassisid": "ASRR",
  "run": ".../platform_asrr.sh" },
{ "chassisid": "2OU2N-ARM",
  "run": ".../platform_2ou2n_arm.sh" }
```

dynamic FIT config selection

```
// asrrcommon.c (u-boot)
if (chassis_id == "$Your_ID")
  conf = "...-other.dtb";
else
  conf = "...-asrr.dtb";
```

Real example: meta-armd12m3 ships BOTH platform_asrr.sh and platform_2ou2n_arm.sh in the same build — the chassis ID picked up at boot decides which one runs. One image, two physical systems.

Release Folders Build Themselves

- MC_FW_VERSION is computed from the board's own VER_MAJOR / VER_MINOR / VER_AUX in prjconfig.
- Every .mtd / .tar output is auto-copied into RELEASE_DIR/<MODEL>_<version>...
- No manual renaming or copying between boards or versions.
- Fewer release mistakes, faster hand-off to QA and to customers.

obmc-phosphor-image.bbappend

```
inherit prjconfig

RELEASE_DIR = "${IMGDEPLOYDIR}/${MC_FW_VERSION}"

do_mk_static_symlinks:append() {
 cp image.static.mtd \
   ${RELEASE_DIR}/${MACHINE}_${MC_FW_VERSION}.mtd
}
```

Pre-Formatted rwfs = Faster First Boot = Cheaper to Build

- Normally the read-write overlay (rwfs) formats itself on first boot — and that's slow.
- Pre-builds a padded jffs2 image at build time, matching the exact partition size and erase-block size from the board's dts.
- Shipped pre-formatted inside the release mtd —

  first boot on the line is fast.
- Less time per unit on the production line, at real production volume, is real money saved.

obmc-phosphor-image.bbappend

```
do_generate_formatted_rwfs_static() {
  RWFS_SIZE_HEX=$(printf "0x%x" ${RWFS_SIZE})
  ERASE_SIZE_HEX=$(printf "0x%x" ${ERASE_BLOCK_SIZE})

  # pad + erase-block match the board's dts rwfs partition,
  # so it drops in ready to flash
  mkfs.jffs2 --root=${S}/rootfs_rw \
       --faketime \
       --output=${IMAGE_LINK_NAME}.jffs2
}
```

Network-Boot Over TFTP/NFS Instead of Reflashing

- Copies the build's fitImage + initramfs into a TFTP root and unsquashfs's rofs into an NFS-exported directory —

  no SPI programming per test cycle.

- A drop-in /sbin/init replaces the initramfs default:

  it fetches the kernel over TFTP and mounts the rootfs from NFS through the same overlayfs + jffs2 rwfs scheme as a real flash boot.

- Modify boot command in u-boot environment to boot from remote server.

- Updates the test image finish in seconds, not minutes.

u-boot setenv script

```
setenv model bmcs

setenv nfssvr 192.168.xx.xx:/nfs/${model}
setenv tftpsvr 192.168.xx.xx:${model}

setenv rambootargs "root=/dev/nfs rw
    ip=dhcp nfsroot=${nfssvr},v3
    console=ttyS4,115200n8"

setenv rambootcmd "dhcp;
    tftp 0x83000000 ${tftpsvr}/fitImage;
    tftp 0x93000000 ${tftpsvr}/initrd;
    bootm 0x83000000 - 0x93000000"
```

Full reflash cycles take minutes;
TFTP/NFS iteration takes seconds —
the BMC just re-mounts a network rootfs,
it never reprograms flash.

WHERE THIS GETS US

Scales Forward, Not Just Sideways

n	4	a few files
board SKUs on one shared core	SoC platforms, same framework	per-board porting surface (prjconfig...)

- New board = overlay + prjconfig.bbclass + dts + board config, not a new codebase.
- Same chassis-ID mechanism already lets one image serve multiple physical systems.
- AST2500, AST2600, AST2700, and Nuvoton NPCM845 all landed in the same framework — only the SoC-specific layer changes.
- Result: less RD time per board, fewer regressions, faster releases, cheaper factory boot.

Arm AGI platform landed as one more overlay, with no new codebase.

arm

Booth C72

ASRock Rack

Booth D37

Efficiency by design.
Built for agentic.

