Blackhole & TT-Metalium

The Standalone AI Computer and its Programming Model

Jasmina Vasiljevic, Senior Fellow
Davor Capalija, Senior Fellow

August 2024

tenstorrent

Agenda

- Architecture
- Micro-architecture
- Scale-out
- Software

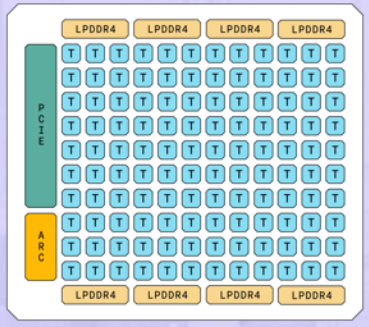
AI Silicon Roadmap

2021 →

High Perf AI ASIC

Grayskull

AI Processor



- 120 Tensix Cores
- 12nm
- 276 TOPS (FP8)
- 100 GB/s LPDDR4
- Gen4x16

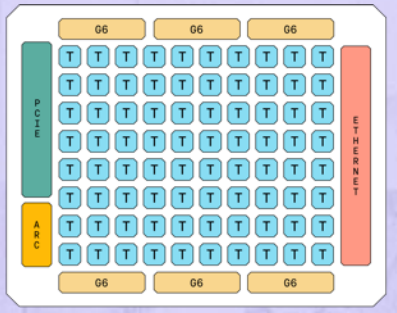
GEN 1

2022 →

Scalability

Wormhole

Networked AI Processor



- 80 Tensix+ Cores
- 12nm
- 328 TOPS (FP8)
- 336 GB/s GDDR6
- Gen4x16
- 16x100 Gbps Ethernet

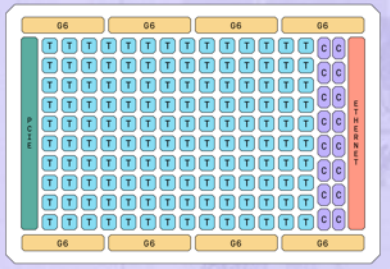
GEN 1

2023 →

Heterogeny

Blackhole

Standalone AI Computer

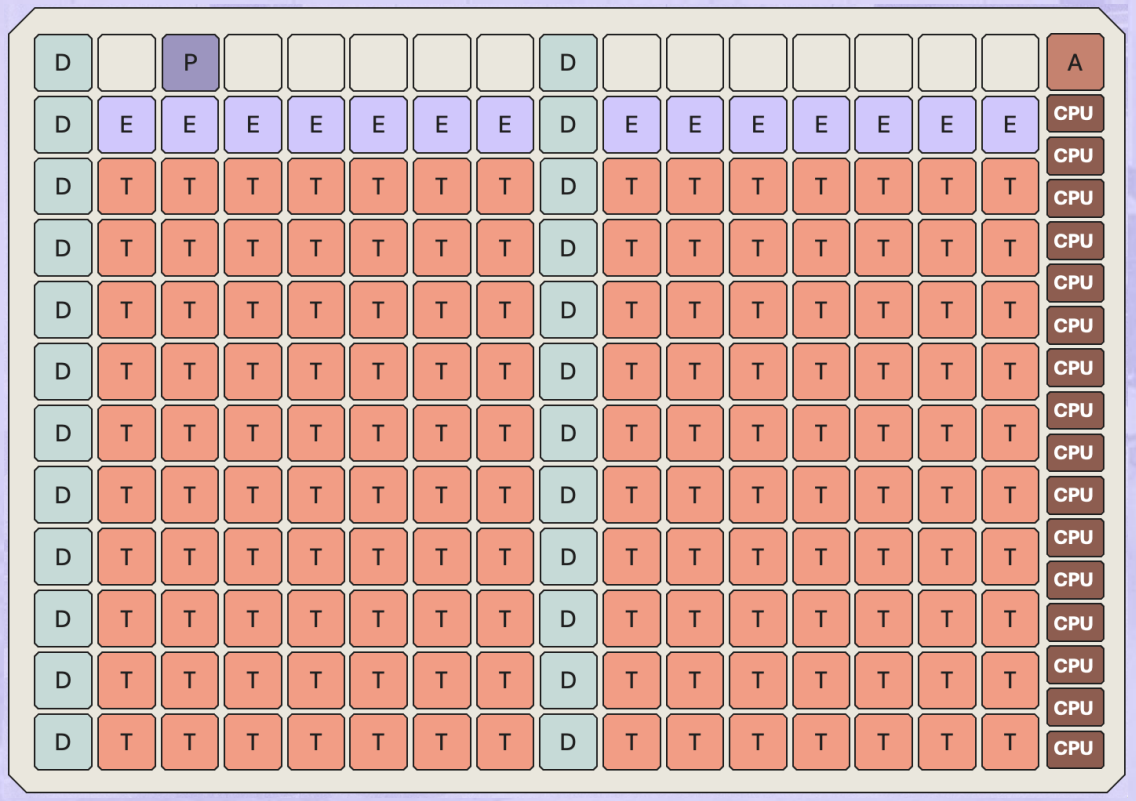


- 140 Tensix++ Cores
- 6nm
- 745 TOPS (FP8)
- 512 GB/s GDDR6
- Gen5x16
- 10x400 Gbps Ethernet
- 16 RISC-V CPU cores

GEN 2

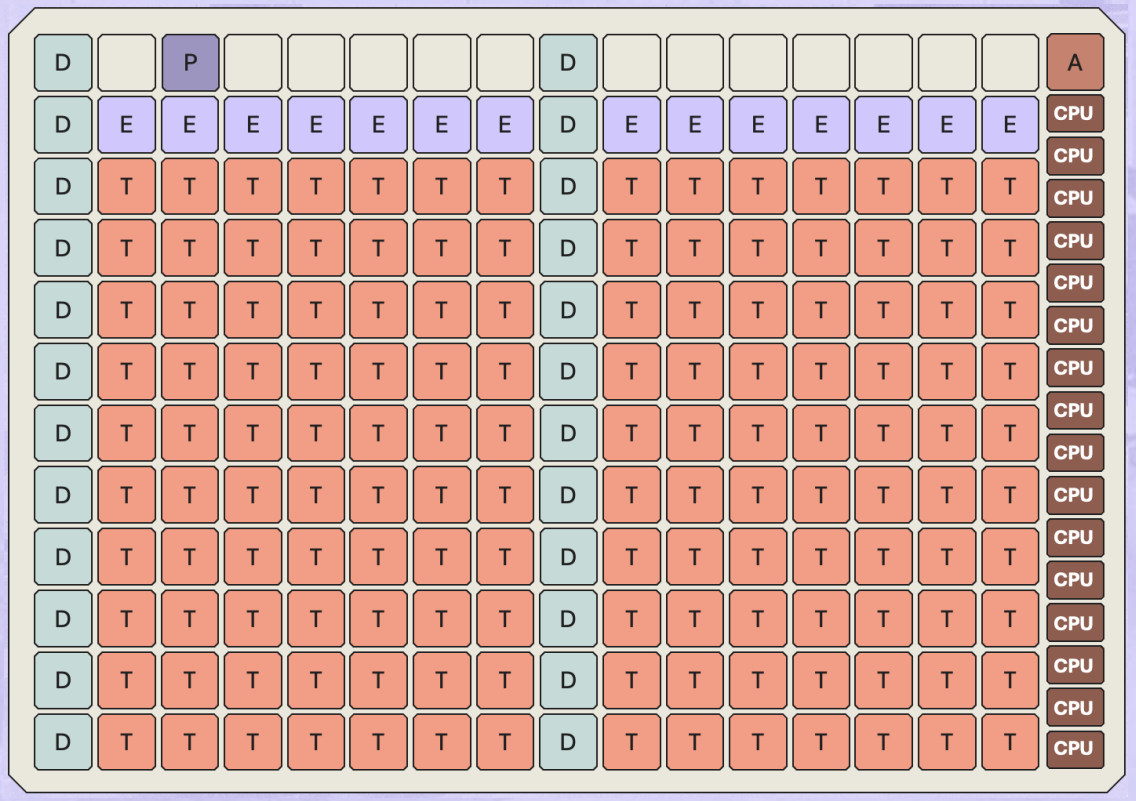
Blackhole - A Standalone AI Computer

C	RISC-V CPUs
T	Tensix cores
D	DRAM cores
E	ETH cores
P	PCIe core
A	ARC core



Blackhole - A Standalone AI Computer

Feature	Spec
Tensix	745 TFLOPs (8-bit) 372 TFLOPs (16-bit)
SRAM	241 MBs
Ethernet	10x 400 Gbps
DRAM	512 GB/s BW 32 GBs capacity
Baby RISC-Vs	752
Big RISC-Vs	16
PCIe	Gen5x16, 64 GB/s
NoC	2 NOCs 2D Torus 256 B per core



Big RISC-V & Baby RISC-V

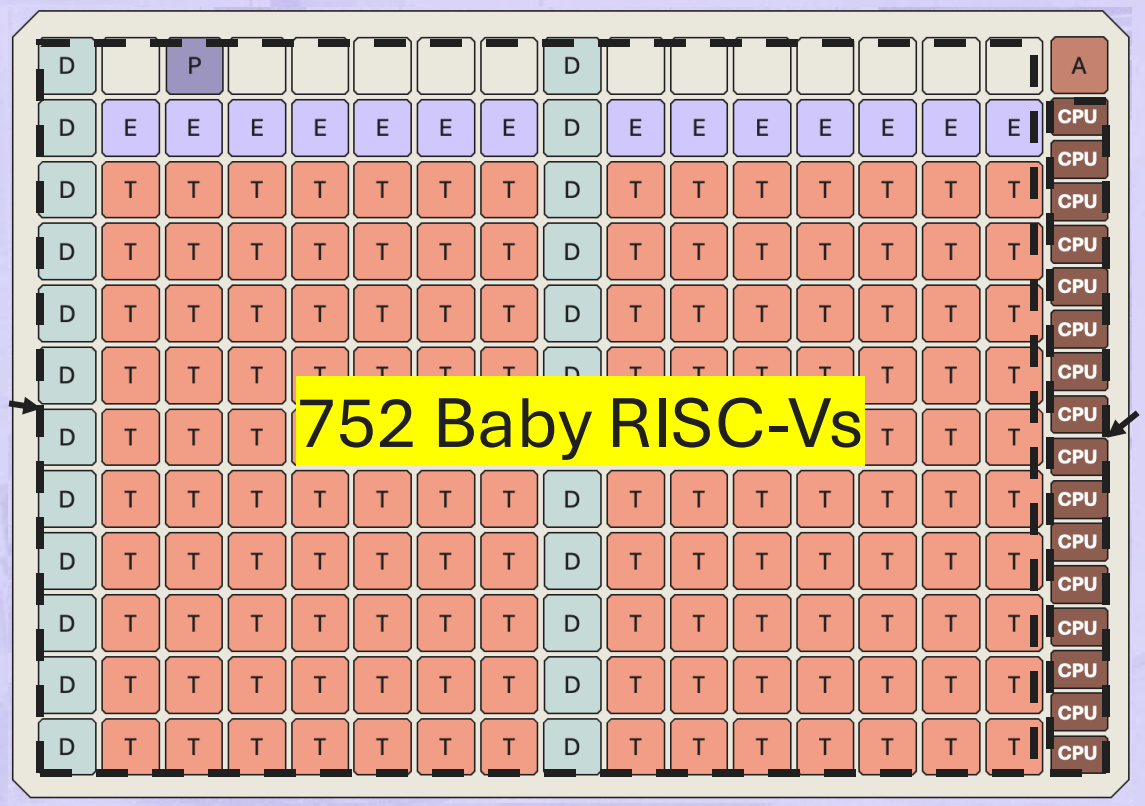
Baby RISC-V

Feature	Spec
Total Baby RISC-Vs	752
Compute	32-bit Int multiplier / divider Floating point (FP32 / BFLOAT16) 128-bit vector (1 per Tensix)
I-cache	4 KB
D-scratch	8 KB

T Tensix cores

D DRAM cores

E ETH cores



Big RISC-V

Feature	Spec
RISC-V CPUs	x16 (4 clusters of 4)
Compute	64-bit, dual-issue, in-order
L3 cache	2 MB / CPU
L2 cache	128 KB / CPU
L1 I-cache	32 KB / CPU (2 way associative)
L1 D-cache	32 KB / CPU (4 way associative)

16 Big RISC-Vs

- Runs Linux
- On-device host for the AI accelerator

C RISC-V CPUs

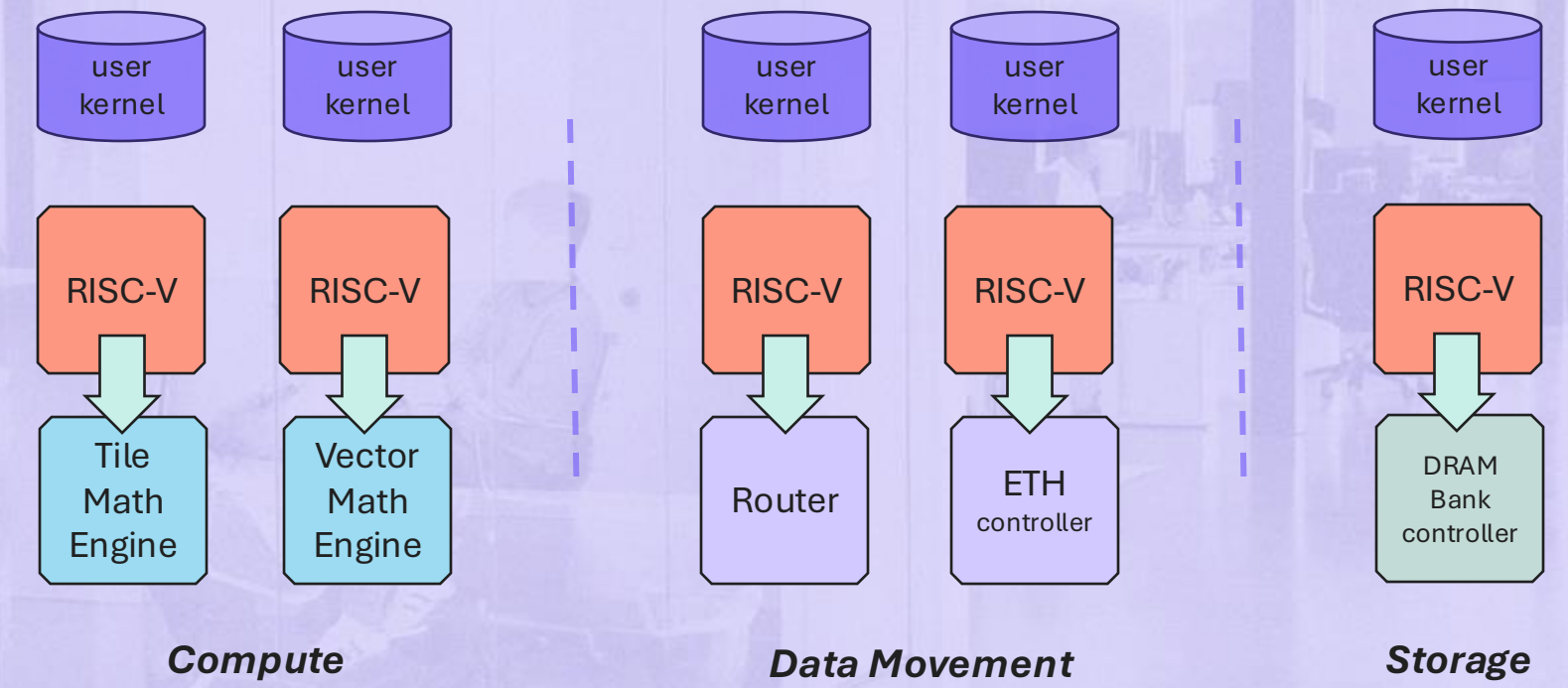
Micro-Architecture:

All RISC-V Programmable

tenstorrent

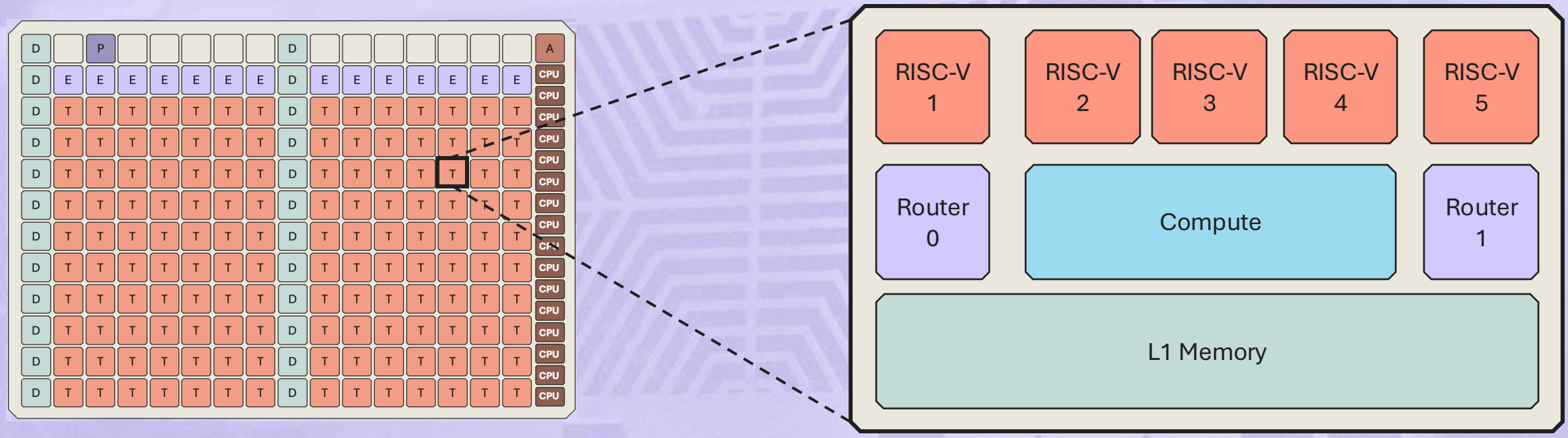
All RISC-V Programmable Baby RISC-Vs

Feature	Spec
Total Baby RISC-Vs	752
Compute	32-bit Int multiplier / divider Floating point (FP32 / BFLOAT16) 128-bit vector (1 per Tensix)
I-cache	4 KB
D-scratch	8 KB



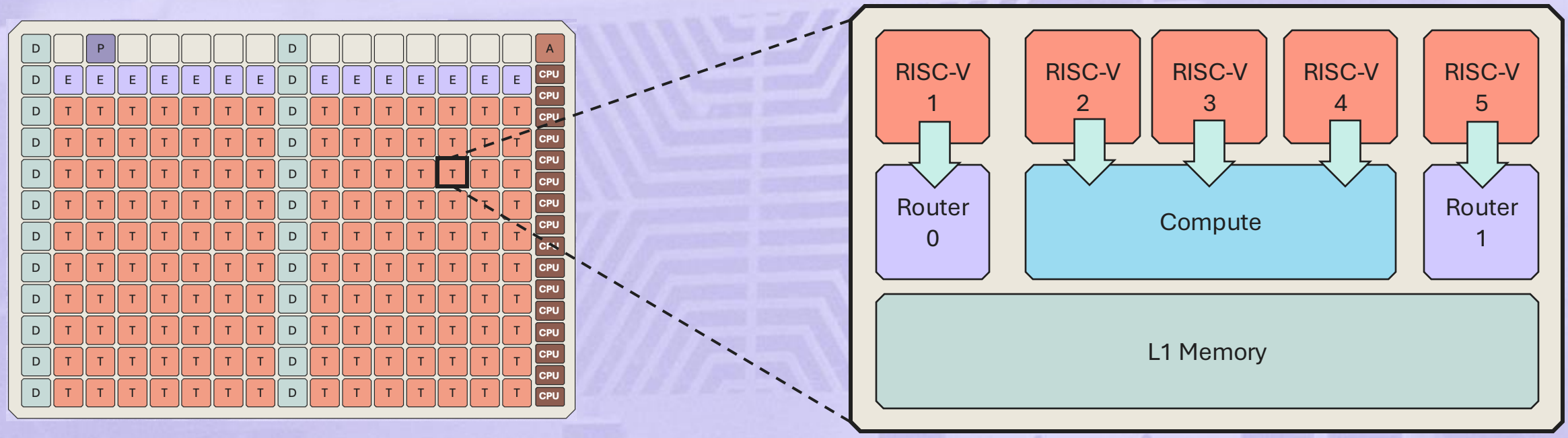
All RISC-V Programmable
Within the Tensix Core

- 5 baby RISC-Vs
- 32-bit RISC-V ISA



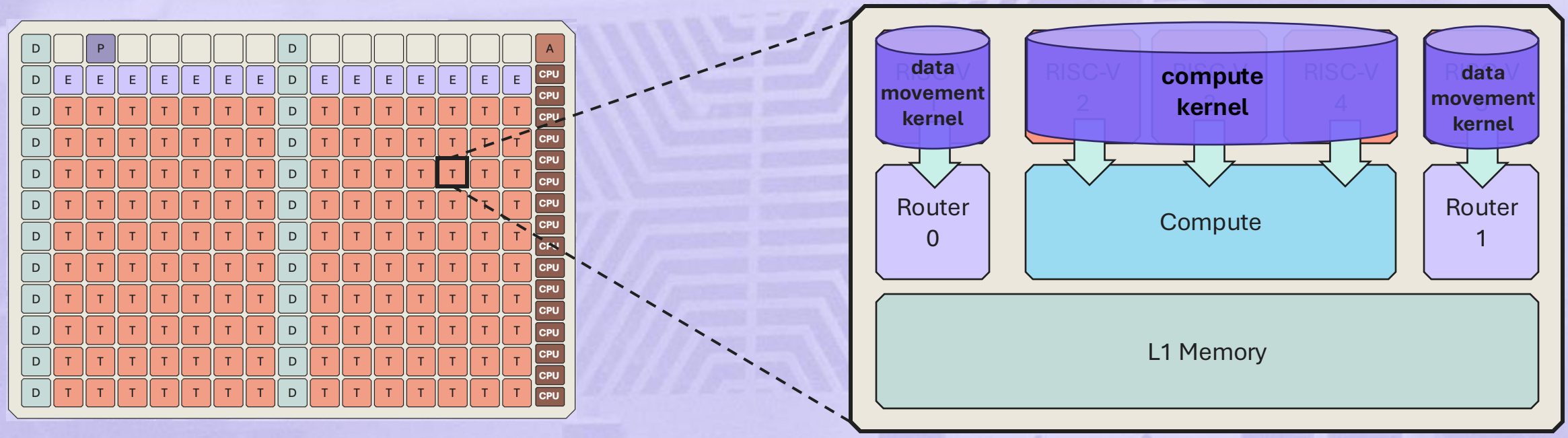
All RISC-V Programmable
Within the Tensix Core

- 5 baby RISC-Vs
- 32-bit RISC-V ISA



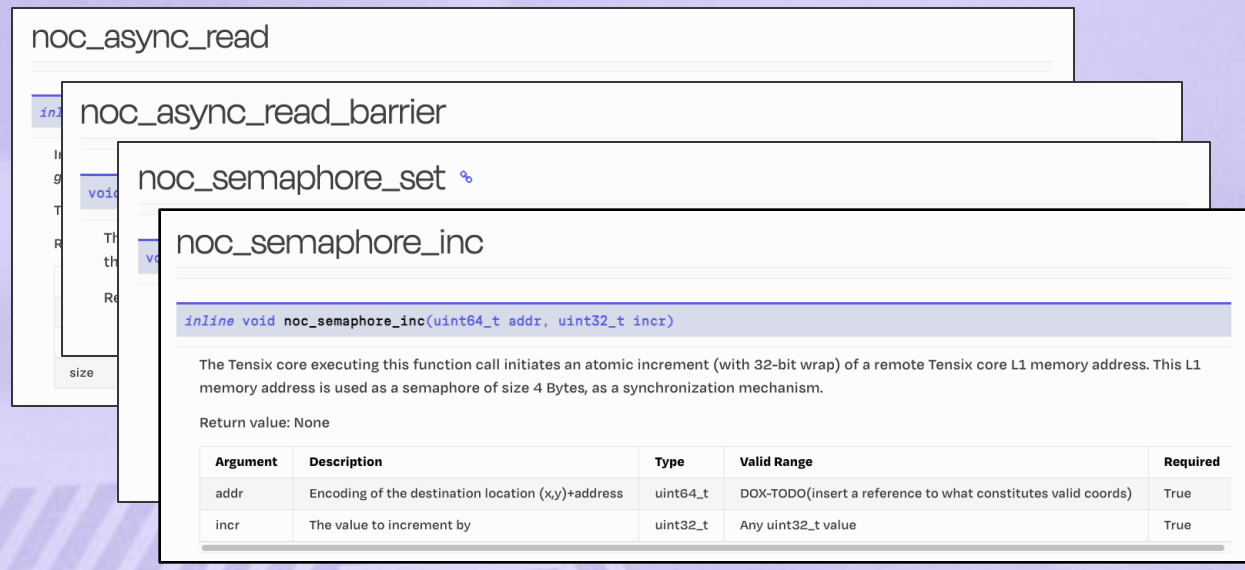
All RISC-V Programmable
Within the Tensix Core

- 3 user C kernels program a single Tensix core
  - 1 compute kernel
  - 2 data movement kernels



Tensix Core – Data Movement

- 2 data movement kernels
- Asynchronous reads & writes
- Access to all SRAM & DRAM banks
- Memory barriers
- Atomic semaphores



noc_async_read

noc_async_read_barrier

noc_semaphore_set

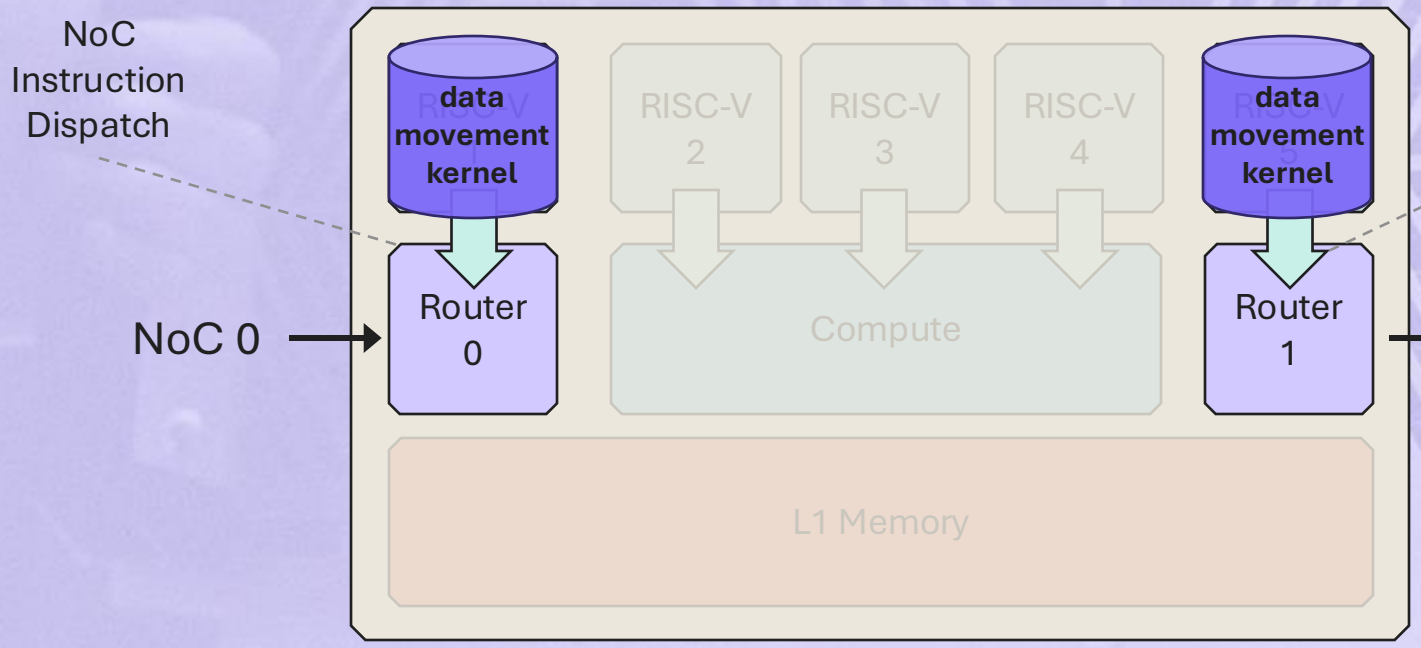
noc_semaphore_inc

```
inline void noc_semaphore_inc(uint64_t addr, uint32_t incr)
```

The Tensix core executing this function call initiates an atomic increment (with 32-bit wrap) of a remote Tensix core L1 memory address. This L1 memory address is used as a semaphore of size 4 Bytes, as a synchronization mechanism.

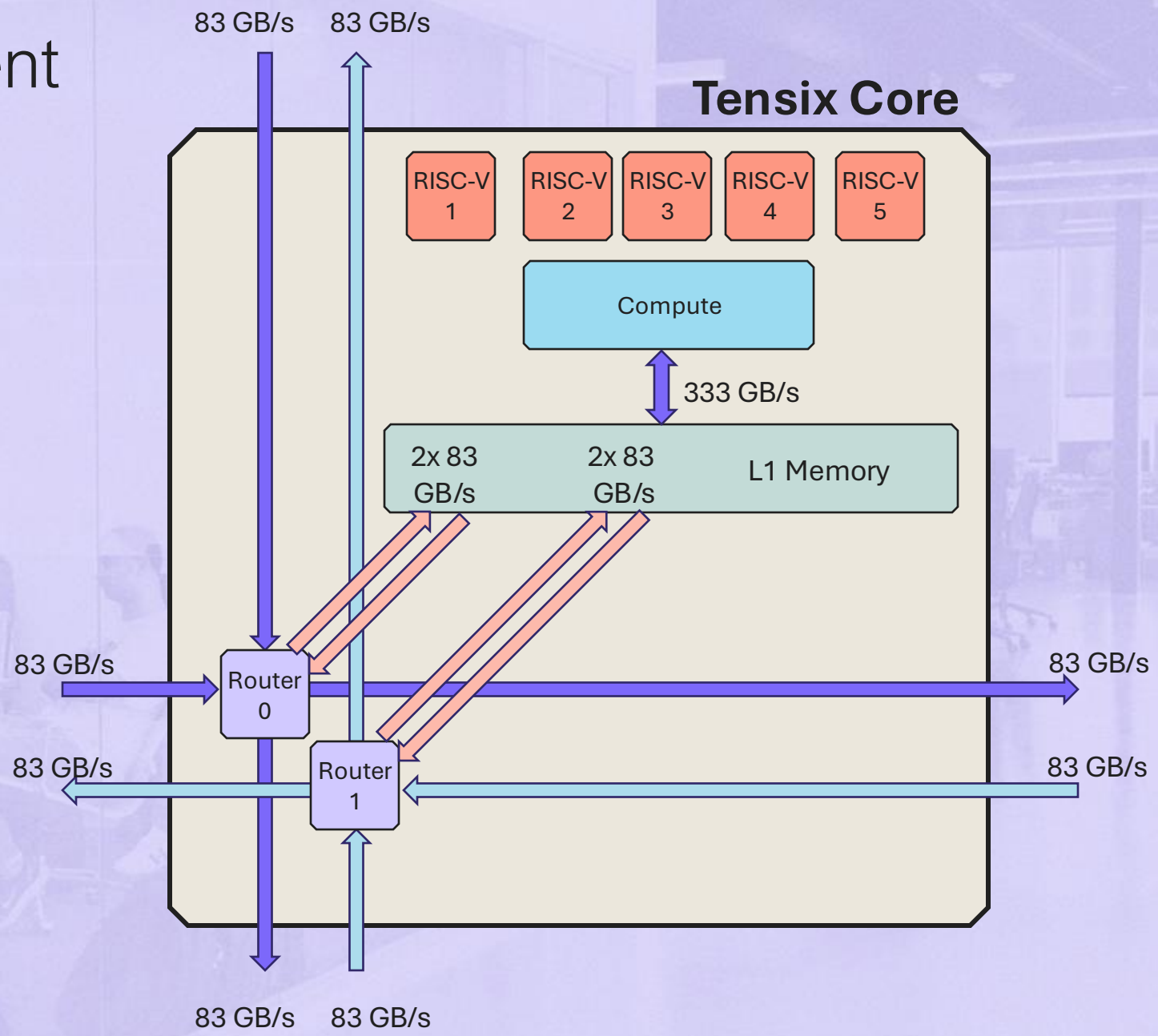
Return value: None

Argument	Description	Type	Valid Range	Required
addr	Encoding of the destination location (x,y)+address	uint64_t	DOX-TODO(insert a reference to what constitutes valid coords)	True
incr	The value to increment by	uint32_t	Any uint32_t value	True



Tensix Core – Data Movement

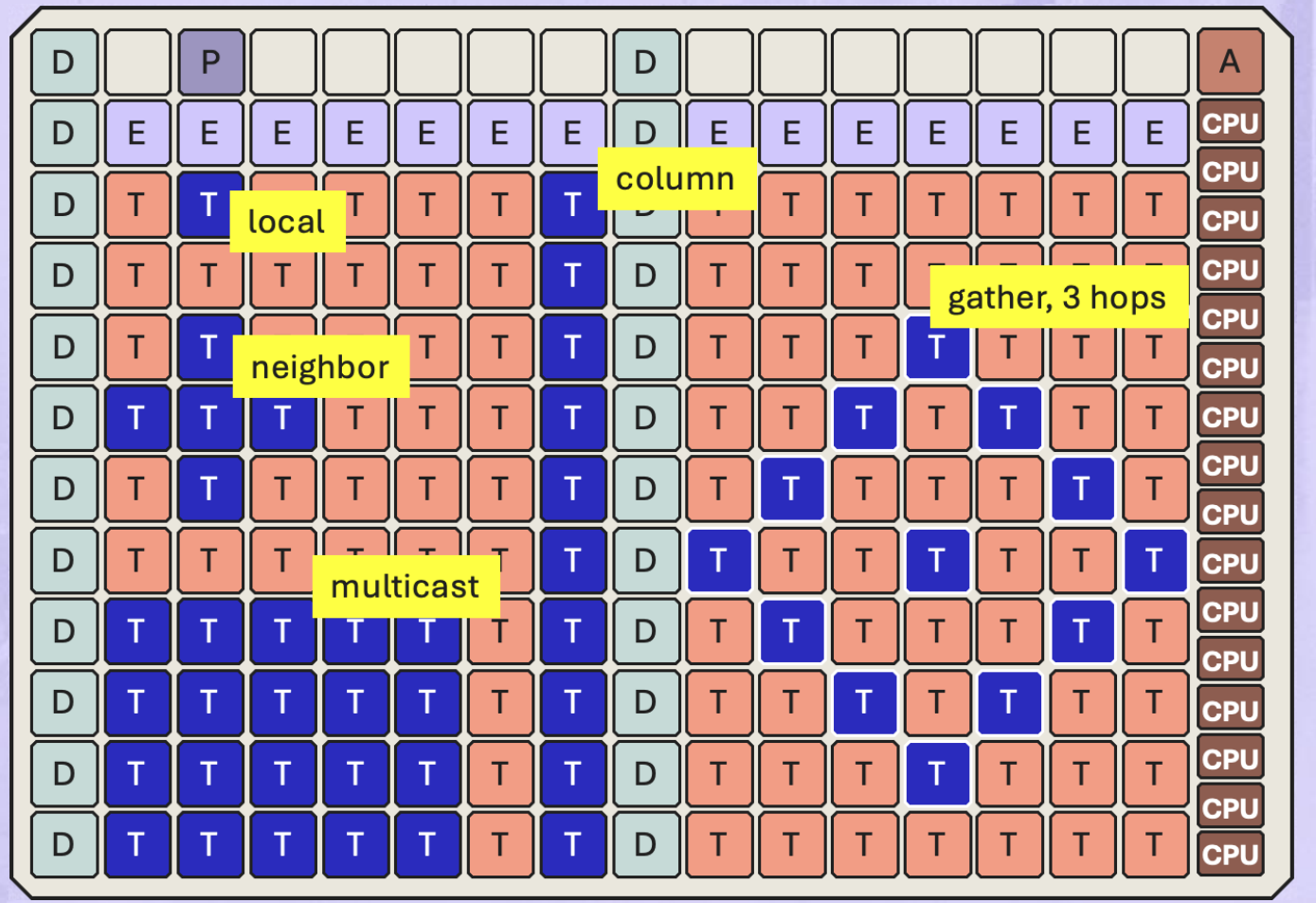
Feature	Spec
Independent NoCs	2
NoC type	2-dimensional torus
NoC link width	64 Bytes
NoC link BW	83 GB/s
Tensix -> NoC I/O BW	665 GB/s
SRAM <-> NoCs	333 GB/s
SRAM <-> NoC aggregate BW	47 TB/s



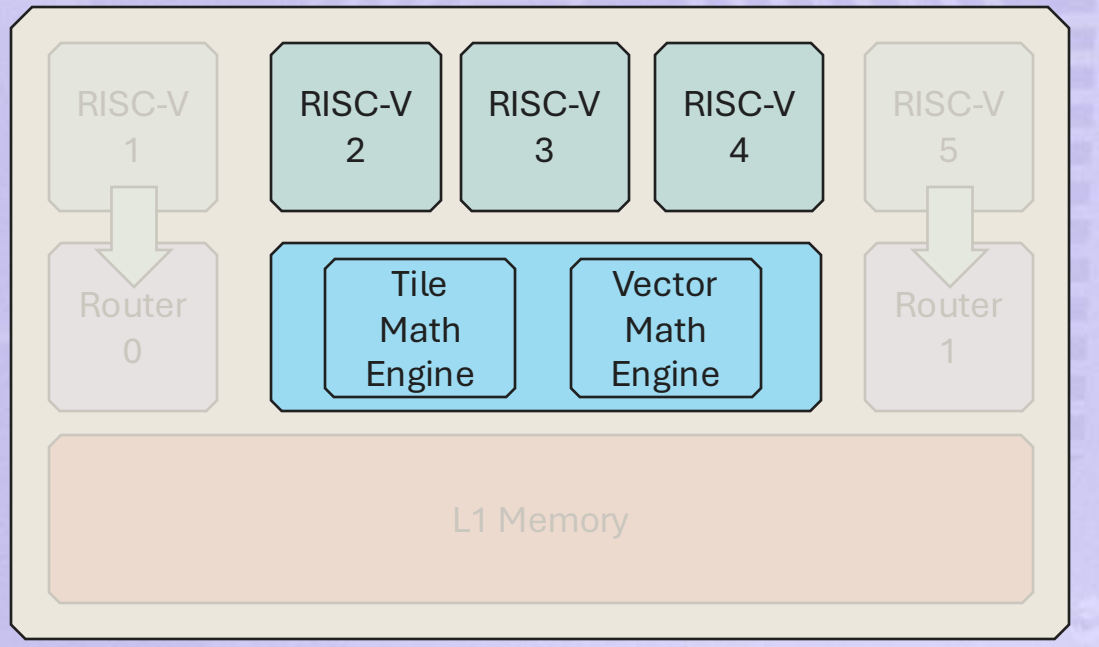
Blackhole: Built for AI Data Movement Patterns

- Data patterns in MatMuls, Convolutions, and Sharded Data Layouts are regular.
- They have a great mapping to Mesh Architecture

Memory & I/O	Data Movement Pattern	Bandwidth
SRAM	Local / Sharded	94 TB/s
SRAM	Neighbor (Halo)	47 TB/s
SRAM	Row / Column / Mesh Multicast	24 TB/s
SRAM	Gather / Scatter (3 hops)	16 TB/s
SRAM	Gather / Scatter (10 hops)	5 TB/s
DRAM	Row	512 GB/s
Ethernet	Column	1 TB/s

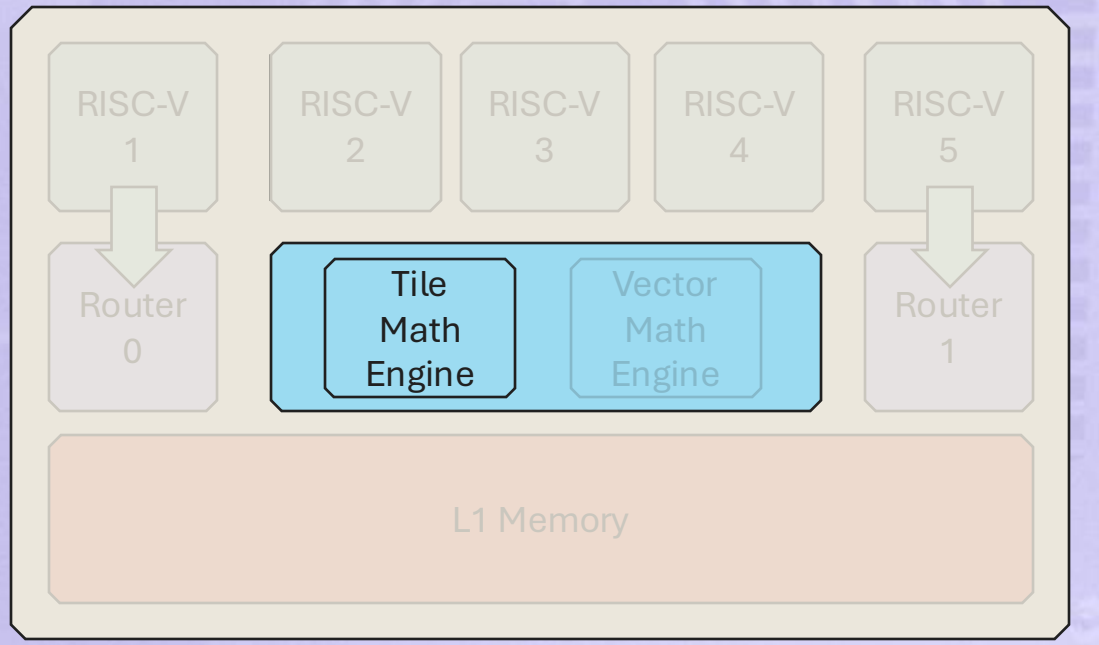


Tensix Core - Compute

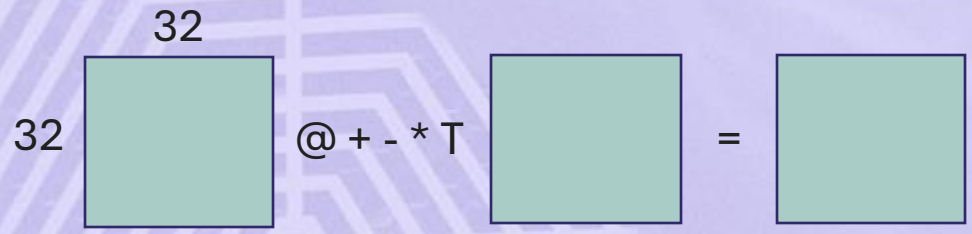


Tensix Core - Compute

- Powerful tile-based math engine



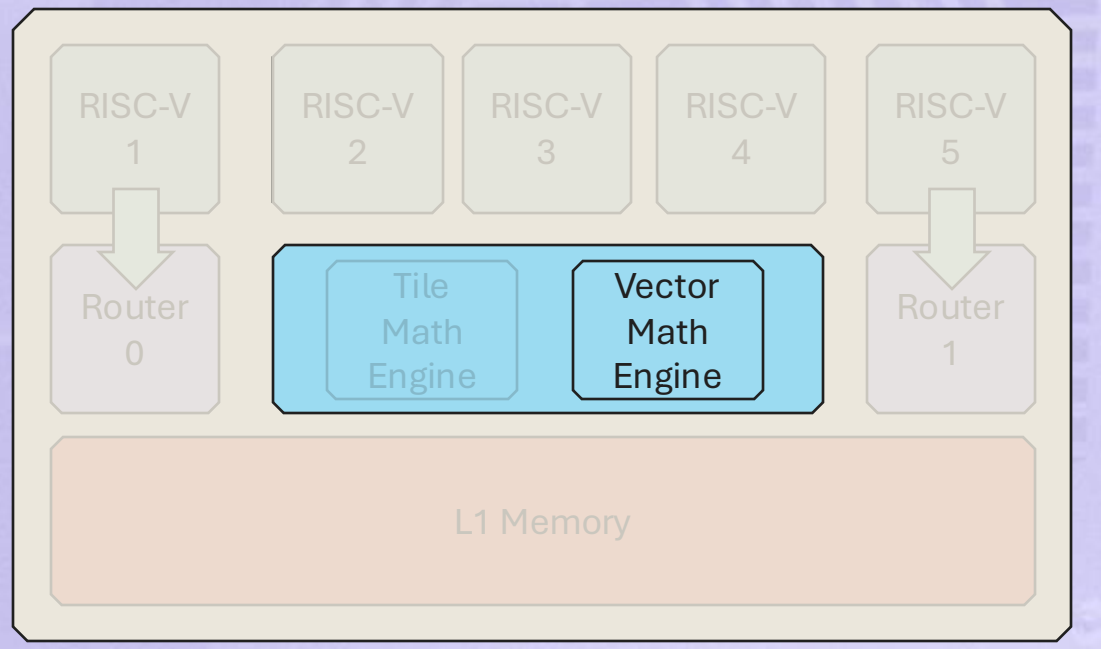
Rich Matrix ISA:
Mat Mul, dot product, elementwise, transpose



Engine	Data Format	Accumulator	TFLOPs
Matrix	Block FP2	FP32	745
Matrix	Block FP4	FP32	745
Matrix	Block FP8	FP32	745
Matrix	FP8	FP32	745
Matrix	BFLOAT16	FP32	373
Matrix	TF32	FP32	186
Matrix	INT8	INT32	186

Tensix Core - Compute

Engine	Data Format	Accumulator	TFLOPs
Vector	FP32	FP32	12
Vector	INT16	INT32	6
Vector	INT32	INT32	6



tan_tile

leaky_relu_tile

exp_tile

```
template<bool fast_and_approx = false>
void ckernel::exp_tile_init()
```

Please refer to documentation for any_init.

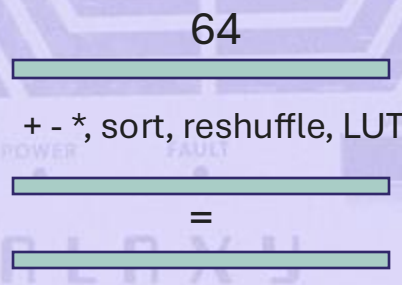
```
template<bool fast_and_approx = false>
void ckernel::exp_tile(uint32_t idst)
```

Performs element-wise computation of exponential on each element of a tile in DST register at index tile_index. The DST register buffer must be in acquired state via *acquire_dst* call. This call is blocking and is only available on the compute engine.

Return value: None

Argument	Description	Type	Valid Range
tile_index	The index of the tile in DST register buffer to perform the computation on	uint32_t	Must be less than the size of the DST register buf
fast_and_approx	Computation to be done faster and approximate	bool	

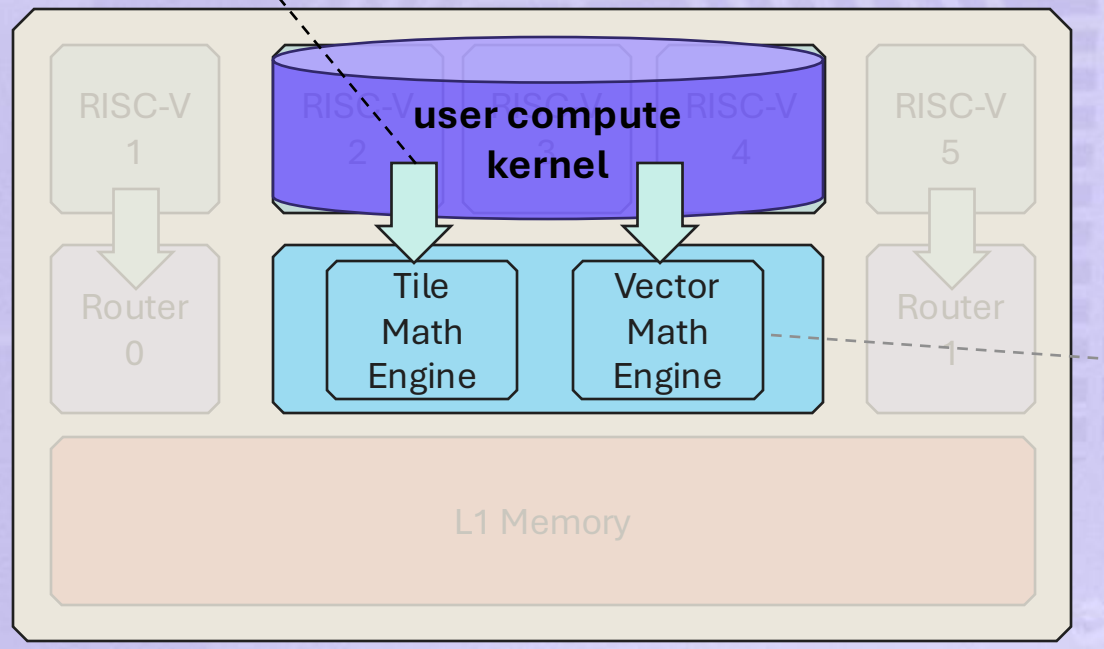
General Purpose Vector ISA:
elementwise, sort, re-shuffle, LUT



Tensix Core - Compute

- 1 user compute kernel
- Automatically compiled to 3 RISC-V threads

Compute Instruction Dispatch



Library of Low-Level APIs

- Open source library of low-level kernels
- 1 API per math function
- 100s of tile & vector math LLKs

transpose_wh_tile

reduce_tile

tilize

matmul_tiles

```
void ckernel::mm_init(uint32_t in0_cb_id = 0, uint32_t in1_cb_id = 1, uint32_t out_cb_id = 16, const uint32_t transpose = 0)
```

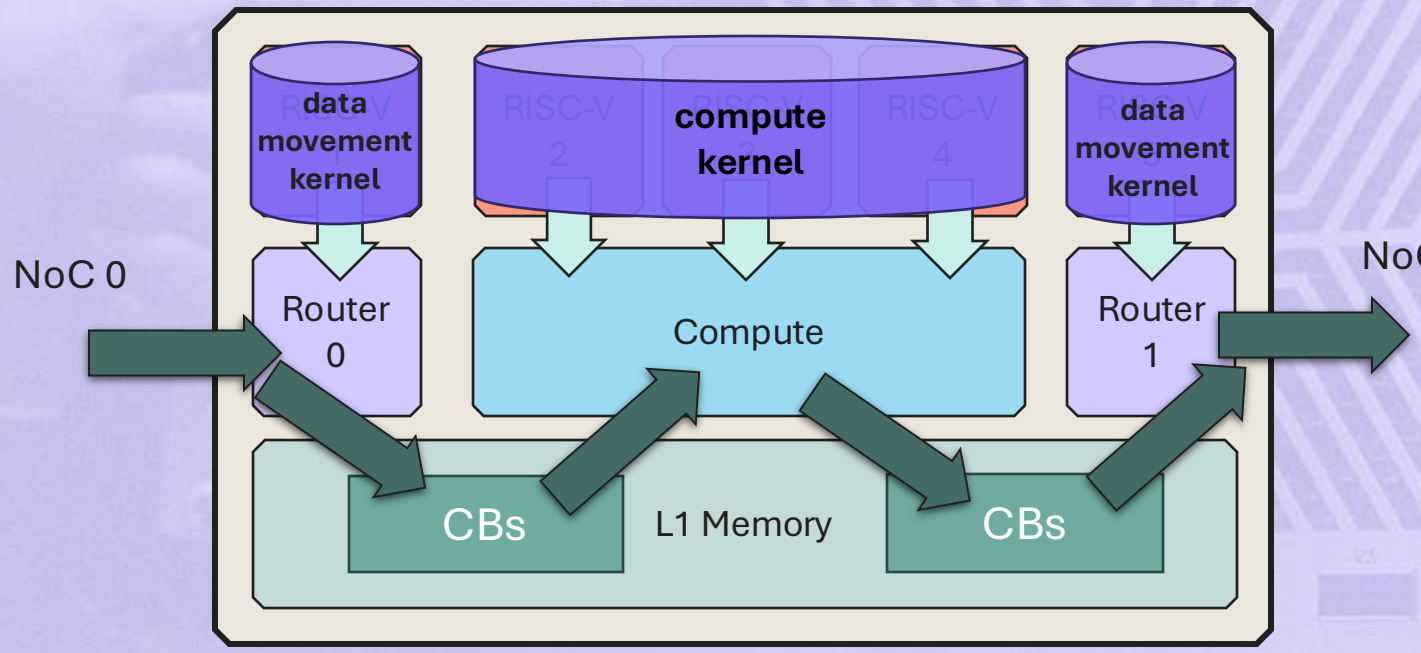
Initialization for matmul_tiles operation. Must be called before matmul_tiles.

Return value: None

Argument	Description	Type	Valid Range	Required
in0_cb_id	The identifier of the first input circular buffer (CB)	uint32_t	0 to 31	False
in1_cb_id	The identifier of the second input circular buffer (CB)	uint32_t	0 to 31	False
out_cb_id	The identifier of the output circular buffer (CB)	uint32_t	0 to 31	False
transpose	The transpose flag for performing transpose operation on B	uint32_t	Any positive value will indicate tranpose is set	False

Kernel Synchronization

- Circular Buffer (**CB**)
- SRAM memory object with hardware-enabled flow control



cb_push_back

cb_pop_front

cb_wait_front

cb_reserve_back

```
void cb_reserve_back(int32_t operand, int32_t num_pages)
```

A blocking call that waits for the specified number of tiles to be free in the specified circular buffer. This call is used by the producer to wait for the consumer to consume (ie. free up) the specified number of tiles.

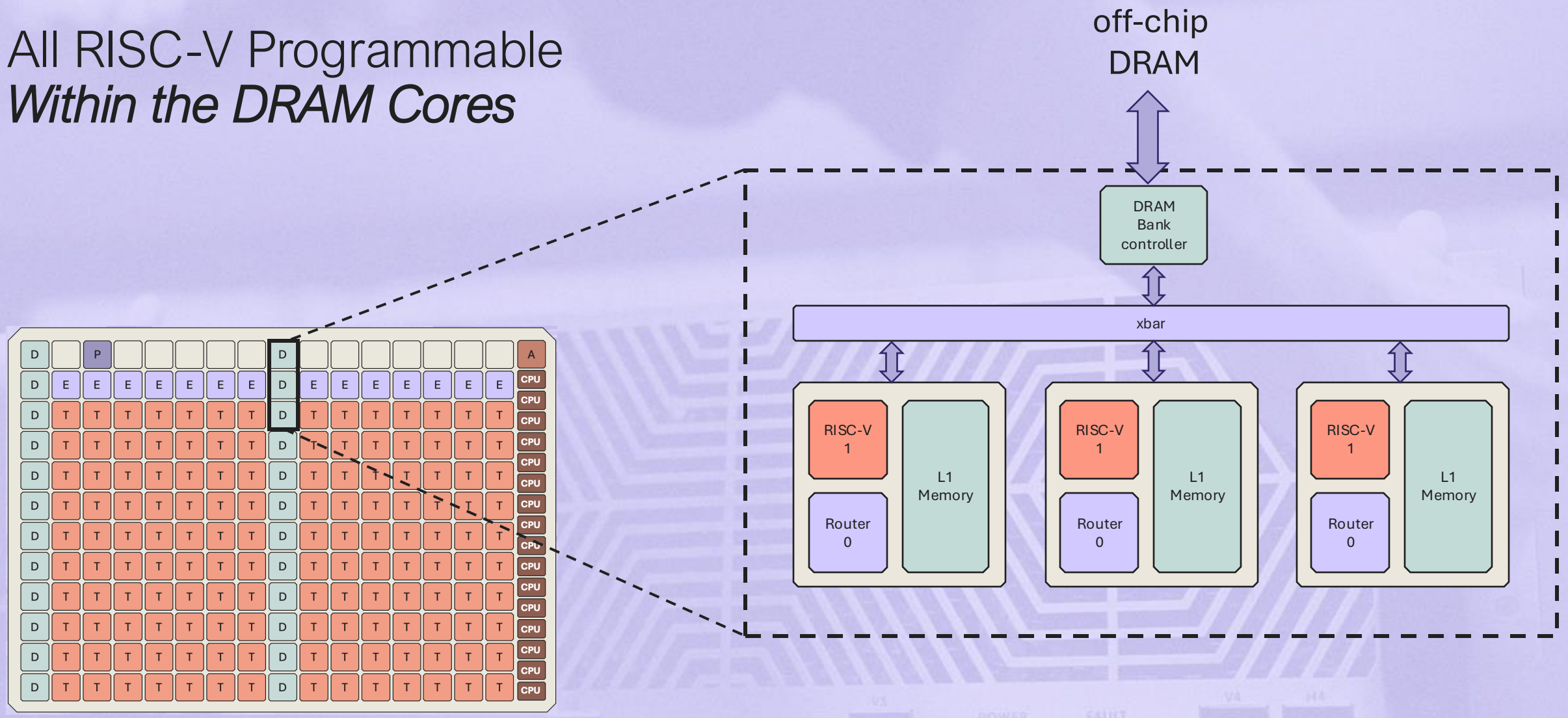
CB total size must be an even multiple of the argument passed to this call.

Return value: None

Argument	Description	Type	Valid Range
cb_id	The index of the circular buffer (CB)	uint32_t	0 to 31
num_tiles	The number of free tiles to wait for	uint32_t	It must be less or equal than the size of the CB (the total number of tiles that fit into the CB)

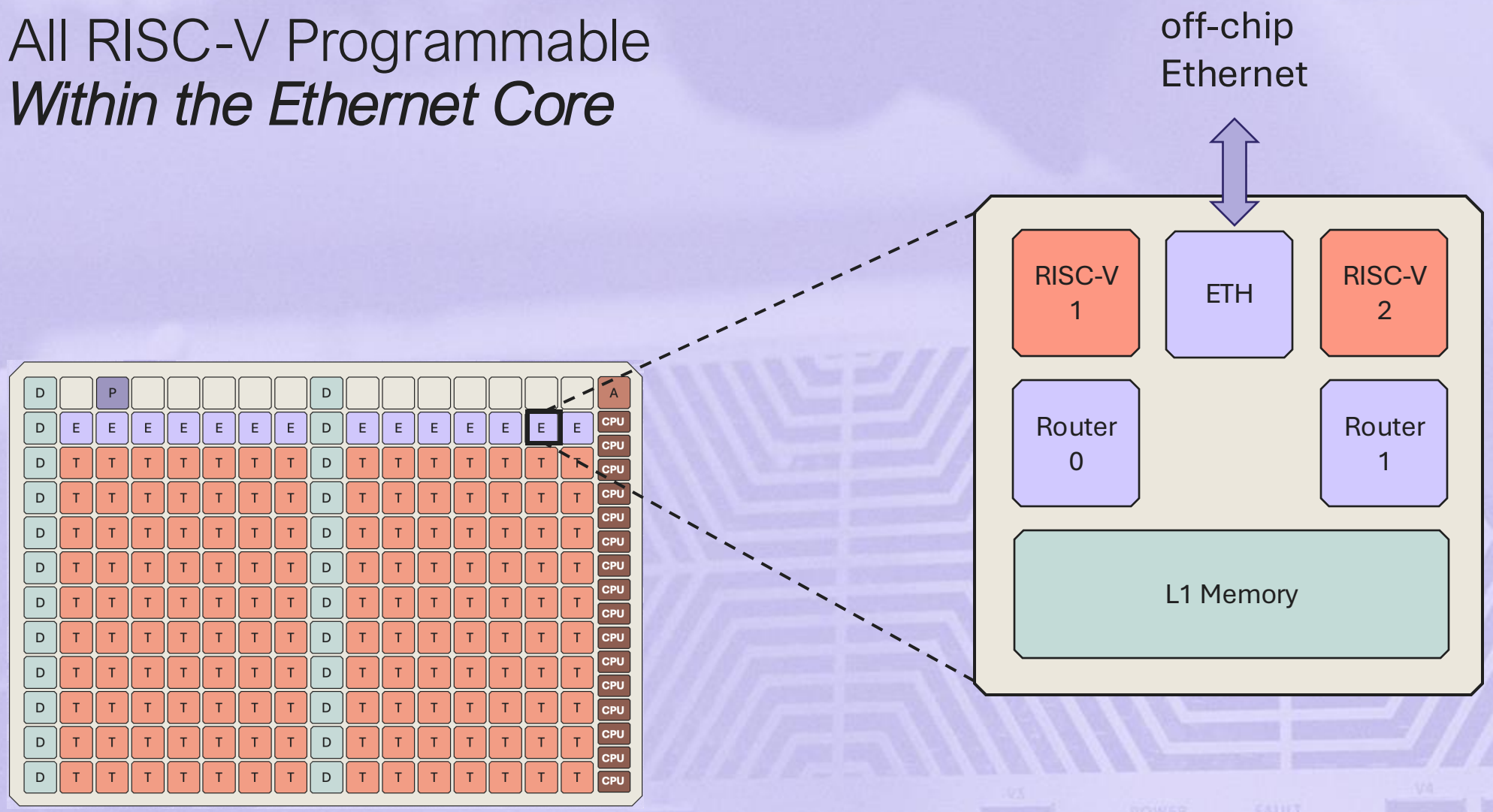
Argument	Description	Type	Valid Range
cb_id	The index of the circular buffer (CB)	uint32_t	0 to 31
num_tiles	The number of tiles to wait for	uint32_t	It must be less or equal than the size of the CB (the total number of tiles that fit into the CB)

All RISC-V Programmable
Within the DRAM Cores



- Kernels for asynchronous pre-load / spill to DRAM

All RISC-V Programmable
Within the Ethernet Core



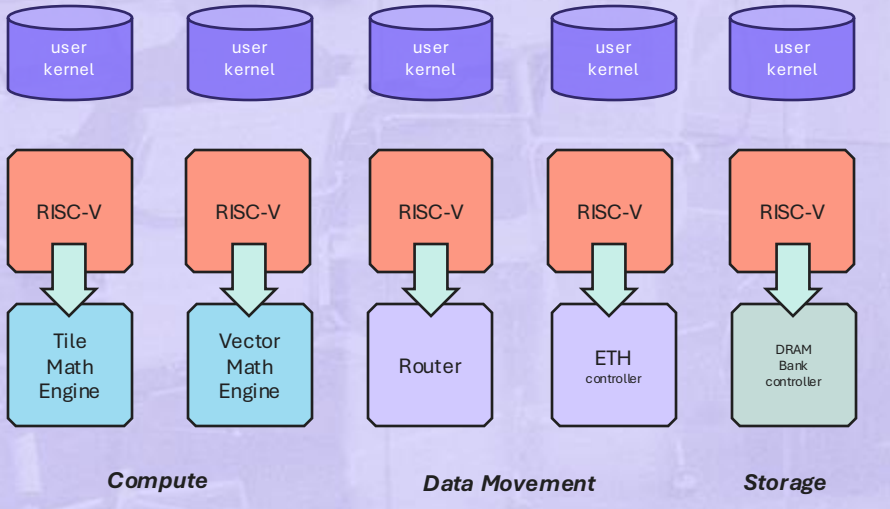
Scale-out

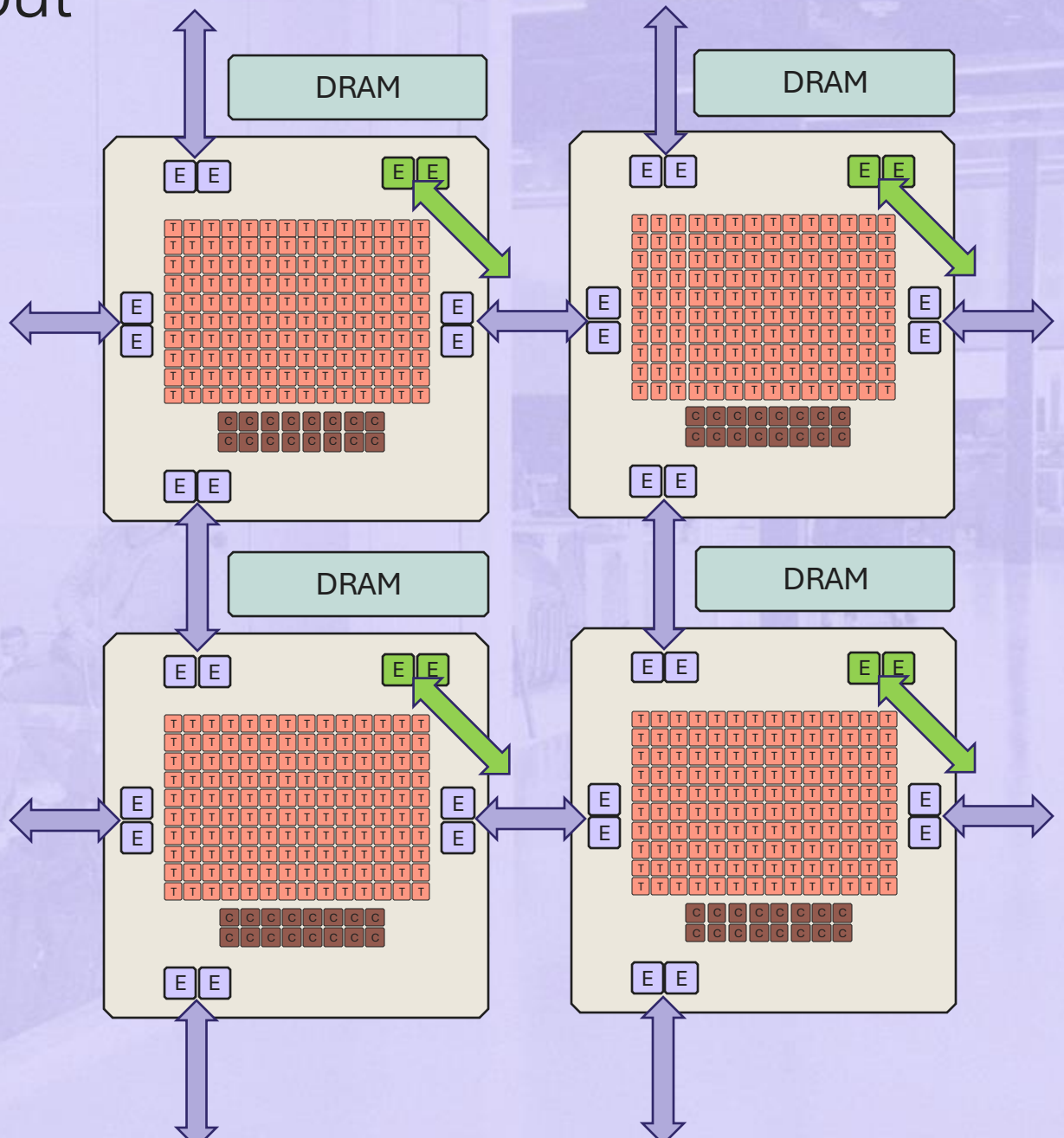
tenstorrent

Blackhole: Ethernet-Based Scale out

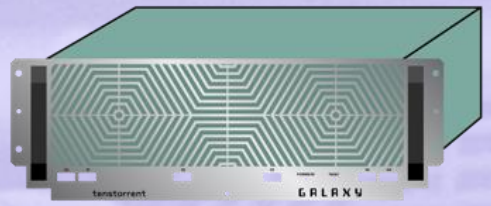
- 1 TB/s of Blackhole Ethernet
- Can be connected into any topology

- Mesh topology is great for AI
  - Locality and regularity of data movement
  - Sharded data
  - 200 GB/s in N / S / W / E / Z
  - 2D / 3D torus





Blackhole Galaxy: 32 chips in a 4x8 Mesh

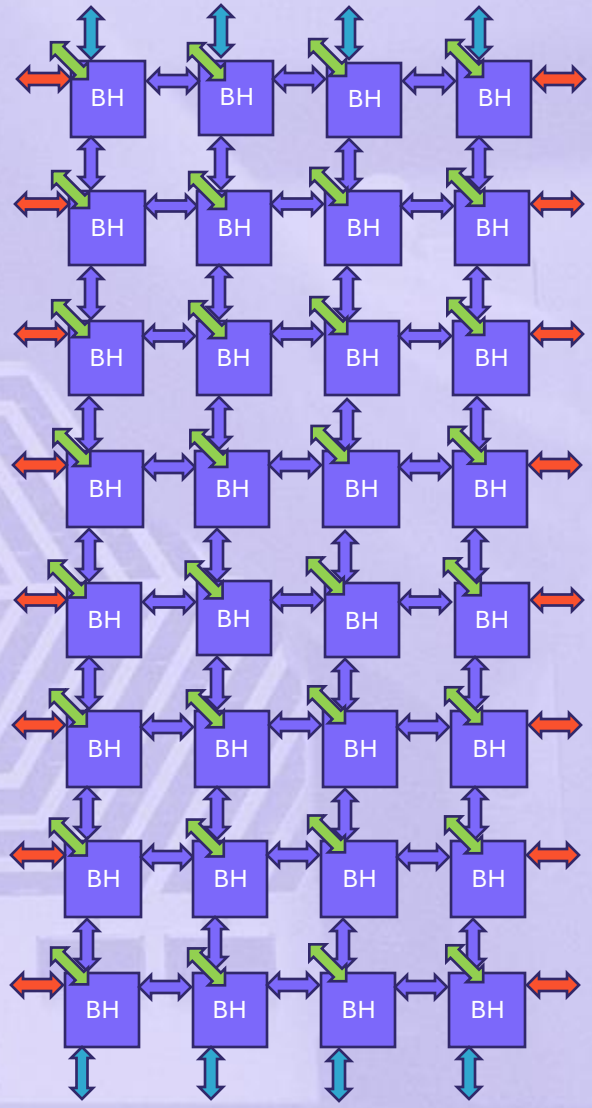


11.2 TB/s Galaxy I/O

X dim I/O: 8 x 200 GB/s

Y dim I/O: 16 x 200 GB/s

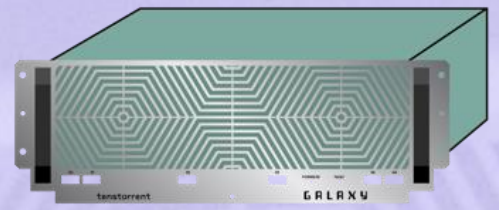
Z dim I/O: 32 x 200 GB/s

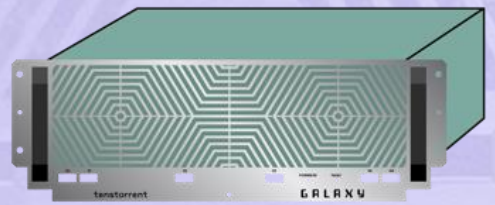


Blackhole Galaxy: Scale-out “Lego" building block

24 PFLOPs
”AI compute”

16 TB/s BW, 1 TB capacity
”AI memory”





11.2 TB/s I/O
”AI switch”

Software

tenstorrent

TT-Metalium: Built for AI and Scale-out

- Kernels are plain C++ with APIs
- Dedicated data movement & compute kernels
  - Optimize data movement and compute overlap directly
- Any core can read/write/sync to any core or chip directly
- Full control of data layout and persistency in SRAM and DRAM
- Different cores can run different kernels and flow data directly between them
- Native multi-device kernels
  - Fused and overlapped compute and inter-chip communication within the kernels

TT-Metalium

Native multi-device kernels & OPs

TT-NN

TT-Metalium
C++ Host API

TT-Metalium
C++ Kernel API

VS.

GPU Programming

Deep Learning OPs

Collective Comms OPs

DNN

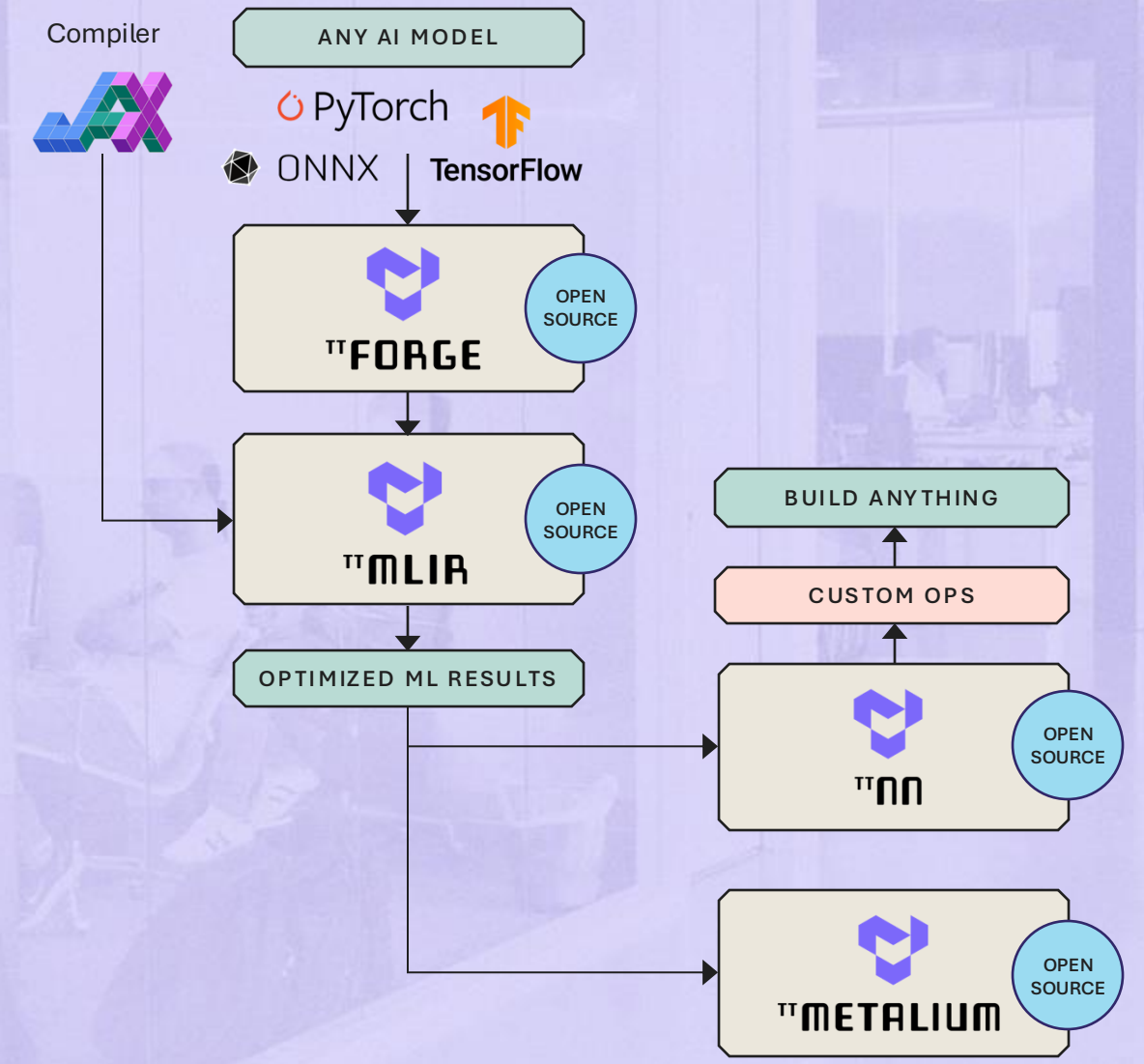
CCL

C++ Host API

GPU
Kernel language

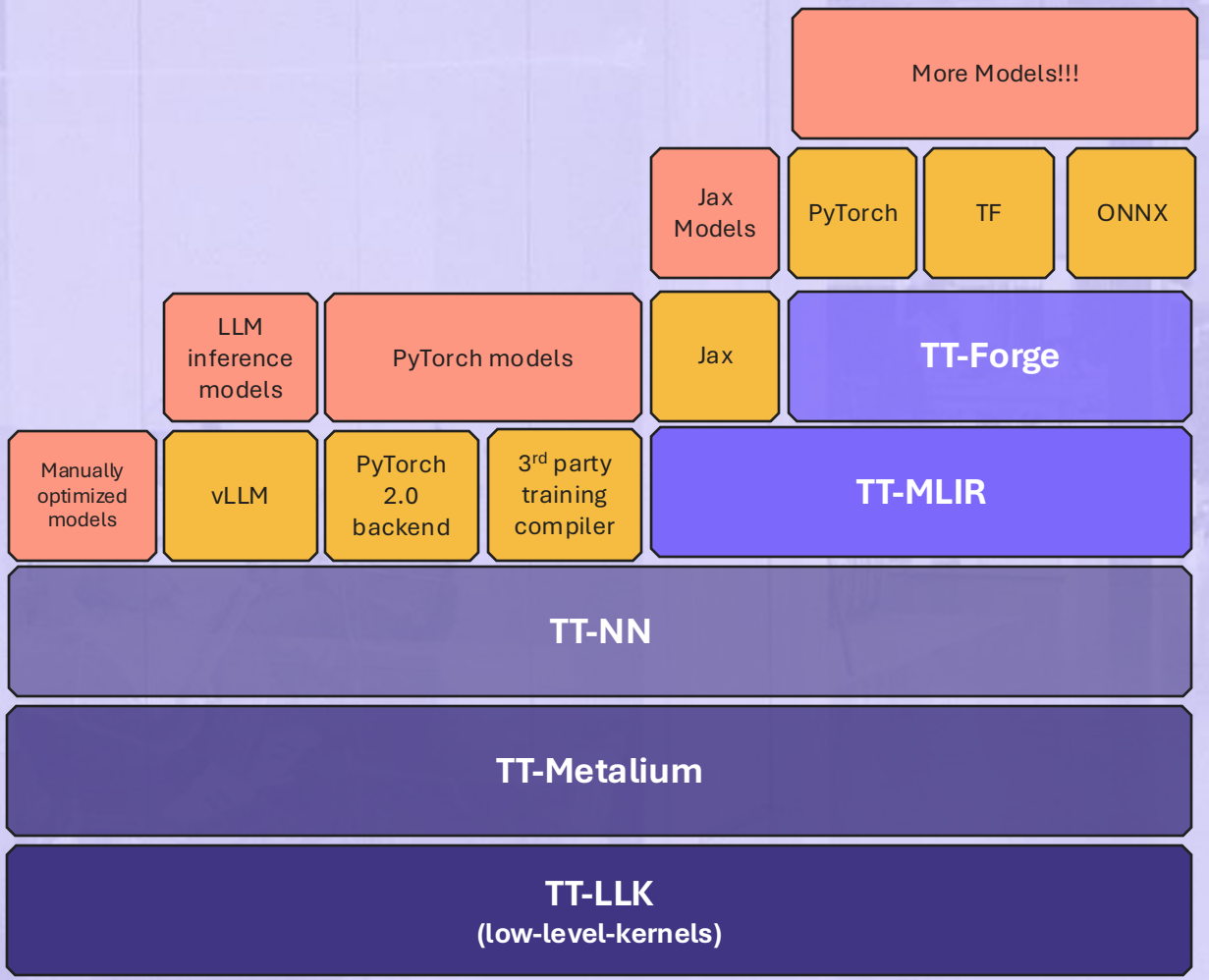
Tenstorrent Open Source Software

- **TT-Forge** - Integrated into various frameworks for native model ingest
- **TT-MLIR** - new MLIR-based compiler
- **TT-NN** – a library of optimized operators
  - ATen coverage
  - PyTorch-like API
- **TT-Metalium** – low level programming model & entry point



Software Ecosystem & Integrations

https://github.com/tenstorrent
https://github.com/tenstorrent/tt-metal
https://github.com/tenstorrent/tt-mlir



Thank you

tenstorrent