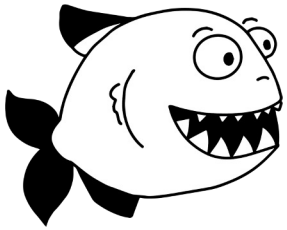


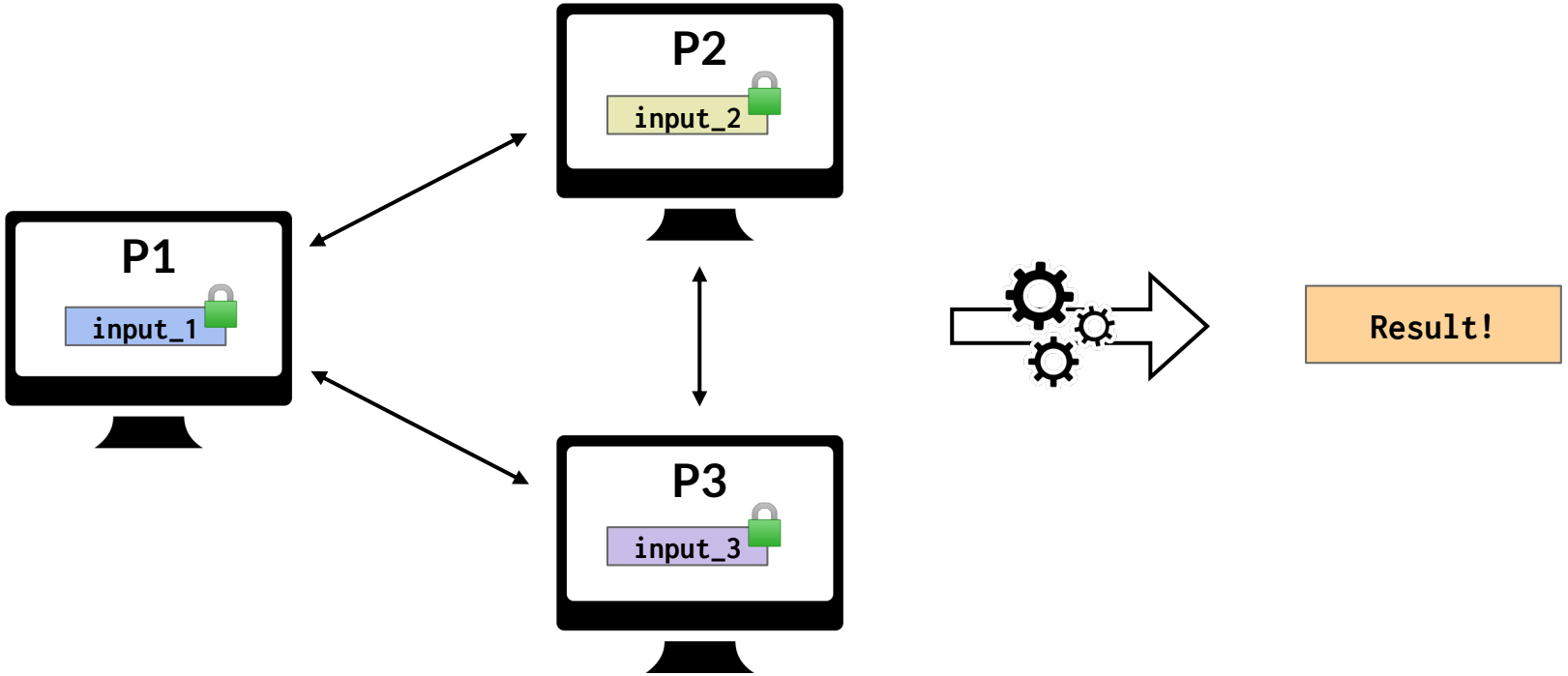
Piranha

A GPU Platform for Accelerating Secure Computation

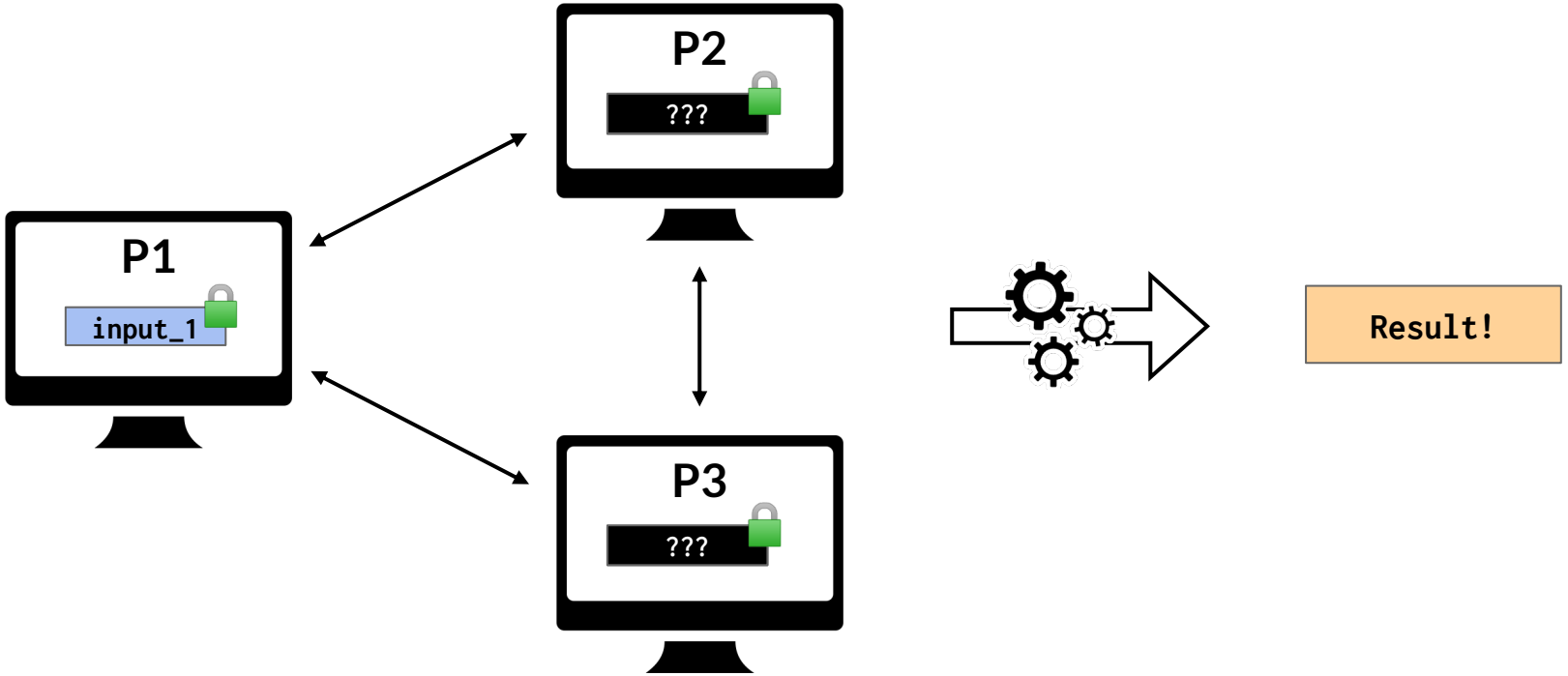
Jean-Luc Watson[1], Sameer Wagh[1,2], Raluca Ada Popa[1]

[1]University of California, Berkeley
[2]Devron Corporation

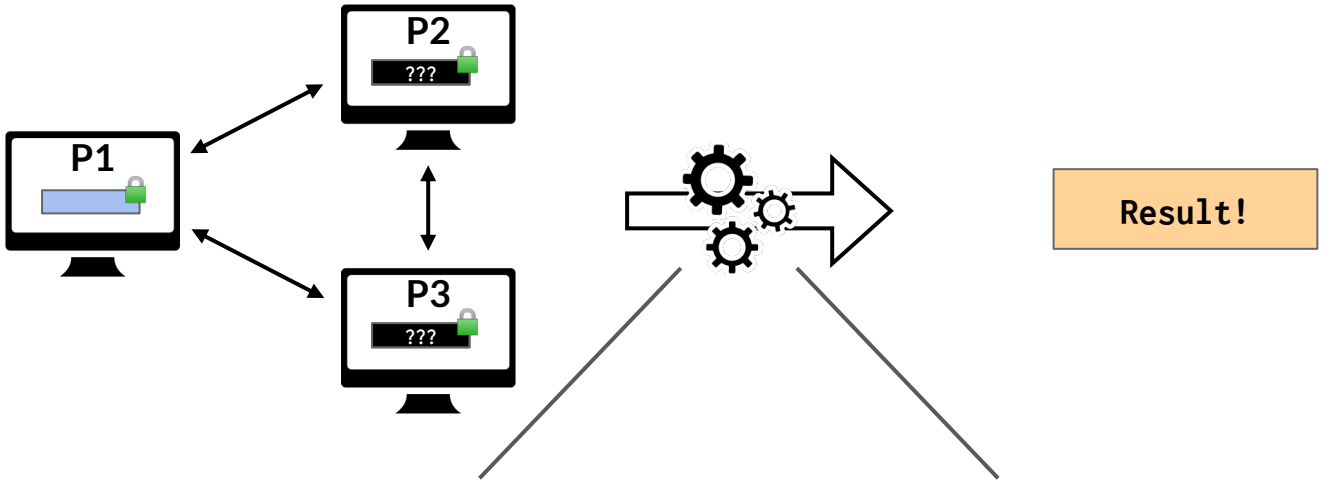
Secure multi-party computation (MPC) [Yao86, GMW87]



Secure multi-party computation (MPC) [Yao86, GMW87]



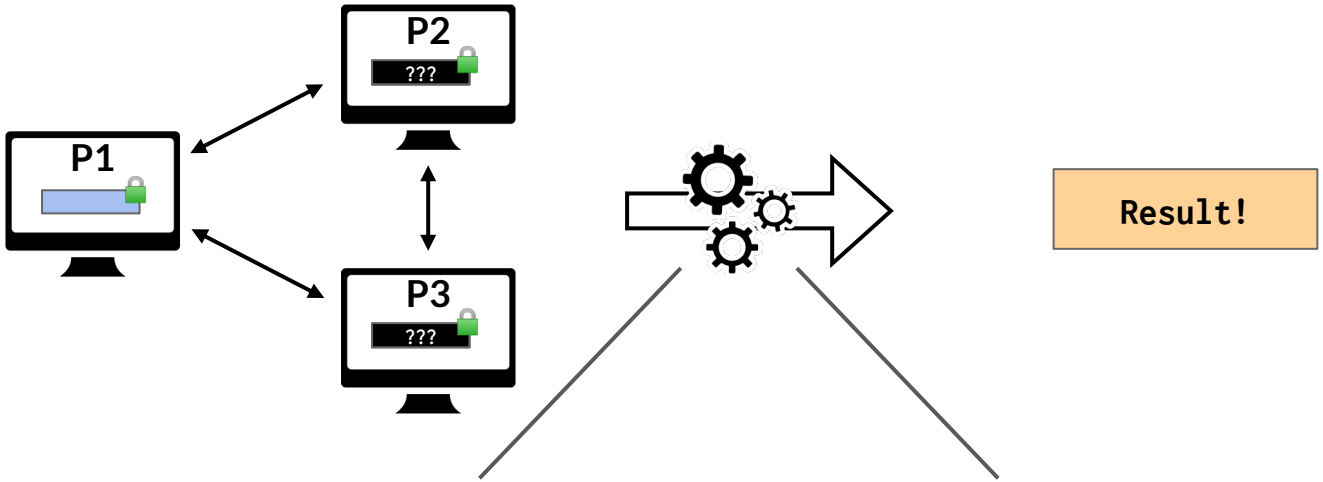
MPC has a performance problem



	Plaintext	MPC-based
AES Encryption	< 100 ns[1]	***~1 ms / block*** [DG21]

[1] https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf, assuming a 3.0GHz processor

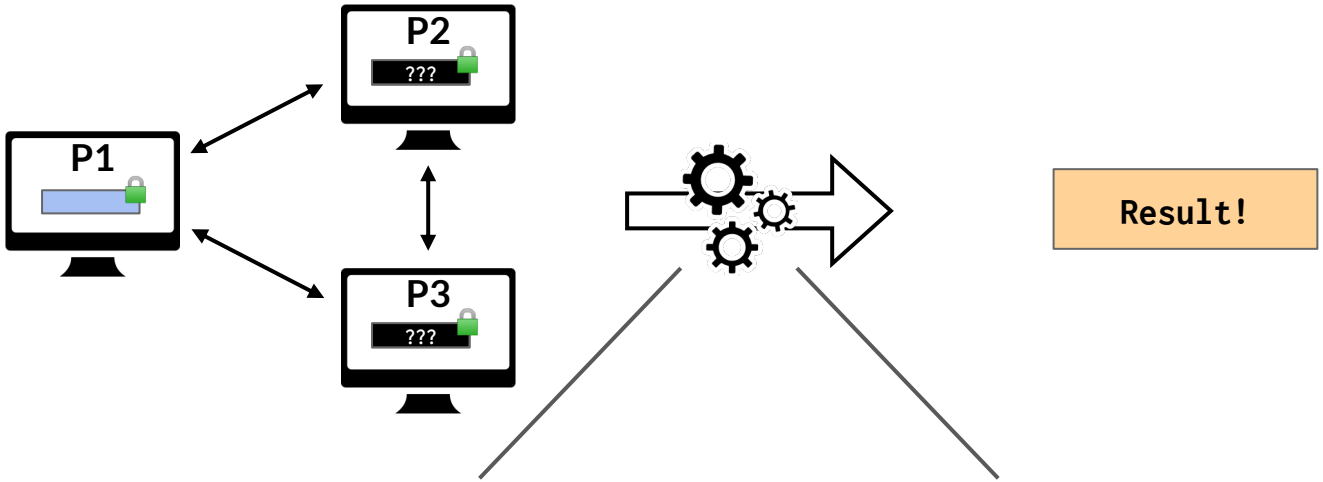
MPC has a performance problem



	Plaintext	MPC-based
AES Encryption	< 100 ns[1]	***~1 ms / block*** [DG21]
ML Inference (VGG16)	58 ms	***100 seconds*** [WTB+21]

[1] https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf, assuming a 3.0GHz processor

MPC has a performance problem



	Plaintext	MPC-based
AES Encryption	< 100 ns[1]	***~1 ms / block*** [DG21]
ML Inference (VGG16)	58 ms	***100 seconds*** [WTB+21]
ML Training (VGG16)	250 seconds	***Estimated 14 days*** [WTB+21]

[1] https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf, assuming a 3.0GHz processor

Privacy-preserving training with MPC

MPC

- ❑ Pick a protocol
- ❑ Implement needed functionality
- ❑ Network parties together
- ❑ Test for correctness
- ❑ Don’t forget to implement training!

Privacy-preserving training with MPC

MPC

- ❑ Pick a protocol
- ❑ Implement needed functionality
- ❑ Network parties together
- ❑ Test for correctness
- ❑ Don’t forget to implement training!

GPU

- ❑ Manage data in GPU memory hierarchy
- ❑ Build useful kernels for application
- ❑ Communicate with CPU
- ❑ Vectorize operations

Privacy-preserving training with MPC

MPC

- ❑ Pick a protocol
- ❑ Implement needed functionality
- ❑ Network parties together
- ❑ Test for correctness
- ❑ Don’t forget to implement training!

Huge gap in expertise

GPU

- ❑ Manage data in GPU memory hierarchy
- ❑ Build useful kernels for application
- ❑ Communicate with CPU
- ❑ Vectorize operations

Bridging the gap: Piranha

MPC

- ❑ Pick a protocol
- ❑ Implement needed functionality
- ❑ Network parties together
- ❑ Test for correctness
- ❑ Don’t forget to implement training!

GPU

- ❑ Manage data in GPU memory hierarchy
- ❑ Build useful kernels for application
- ❑ Communicate with CPU
- ❑ Vectorize operations

Piranha

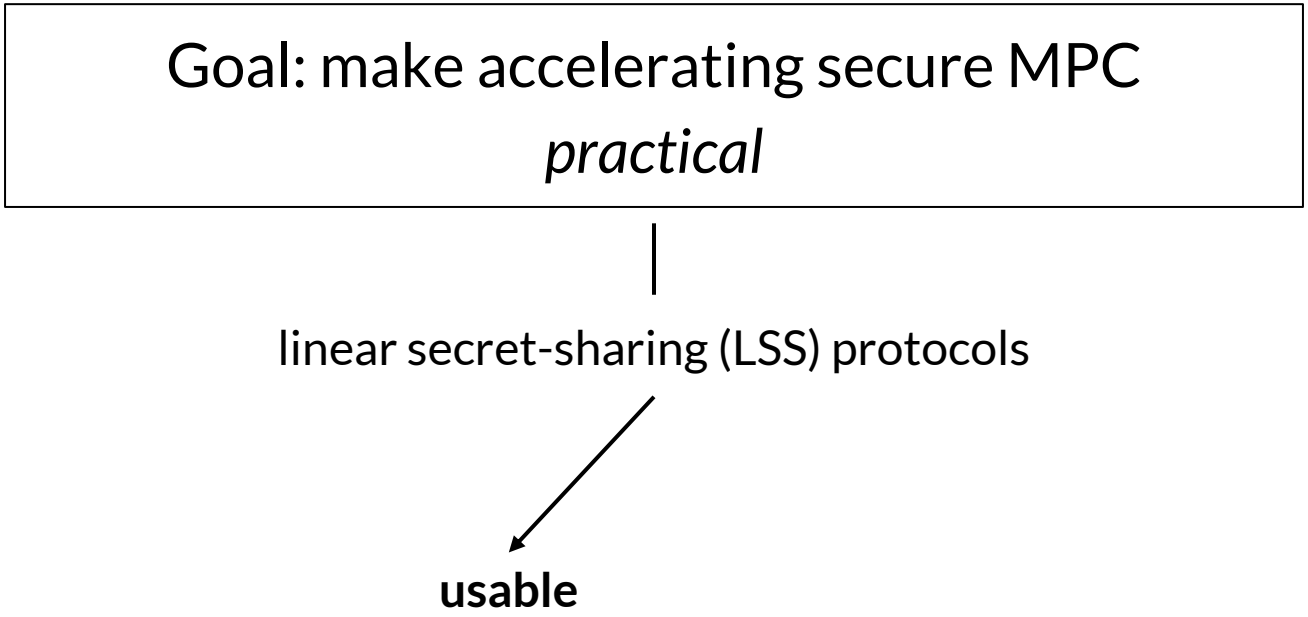
Goal: make accelerating secure MPC *practical*

Piranha

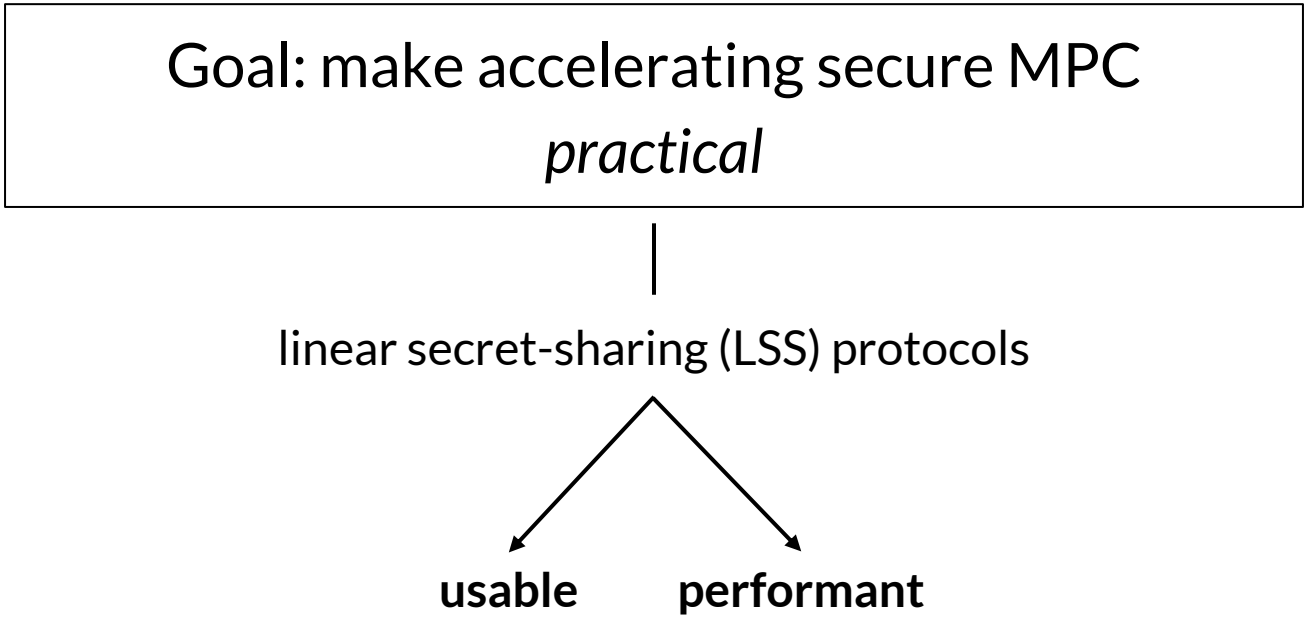
Goal: make accelerating secure MPC *practical*

linear secret-sharing (LSS) protocols

Piranha


Goal: make accelerating secure MPC *practical*
linear secret-sharing (LSS) protocols
usable

Piranha


Goal: make accelerating secure MPC
practical
linear secret-sharing (LSS) protocols
usable
performant

Overview

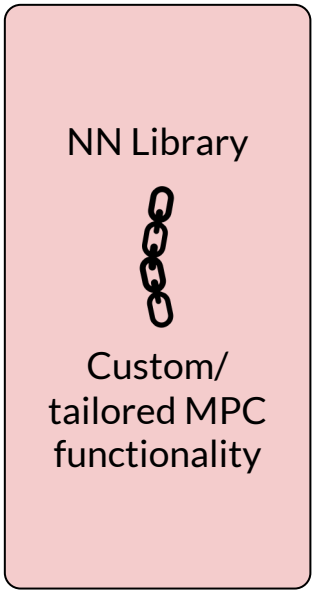
Bringing MPC to the GPU with **Piranha**

Piranha's architecture

Key challenges: acceleration and memory

Evaluation

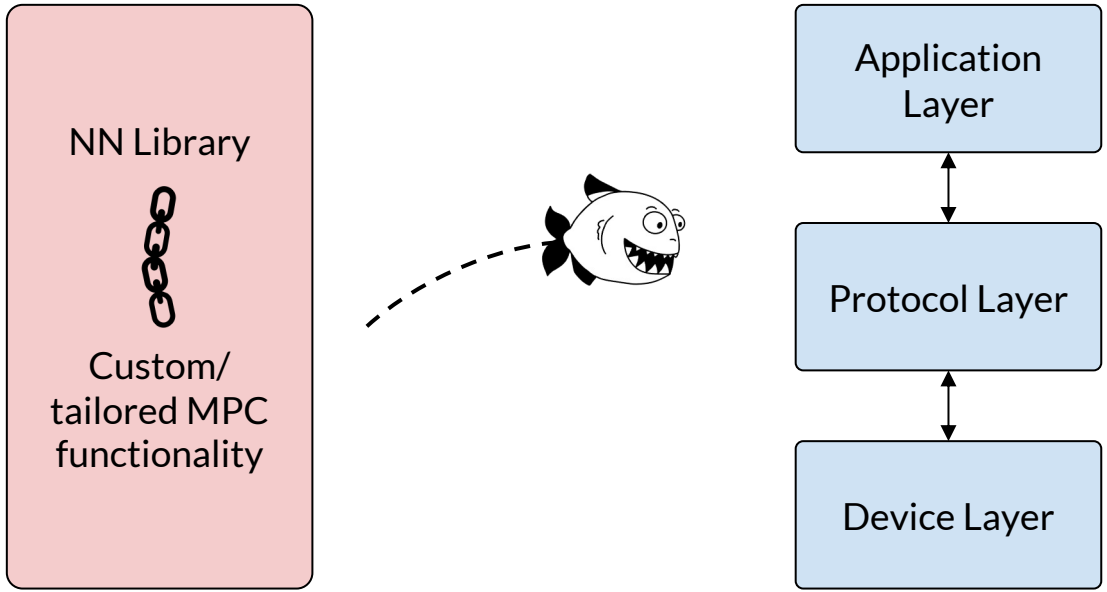
Creating a usable *platform* for MPC



Monolithic

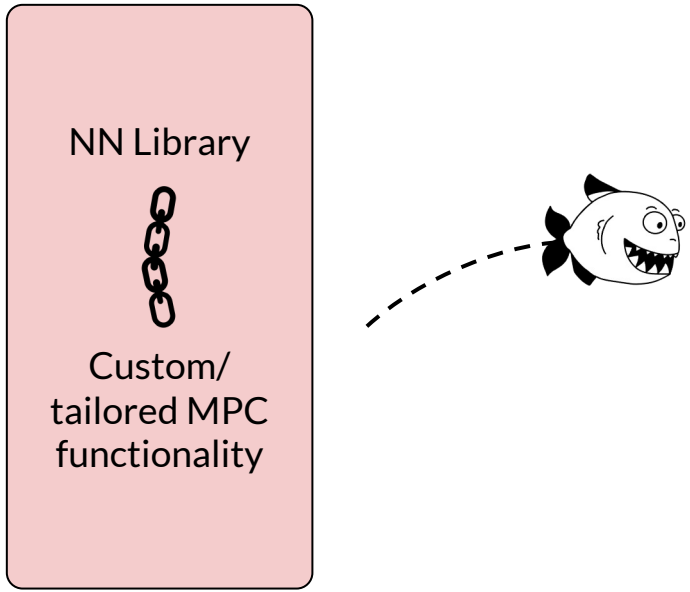
Creating a usable *platform* for MPC

Piranha uses a modular approach to avoid redundancy and easily reuse MPC protocols in different settings.

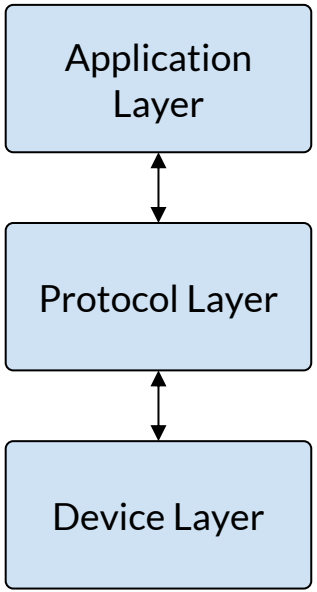


Piranha adds a separation-of-concerns to MPC

In doing so, preserves the security properties of each protocol.



Monolithic



Modular

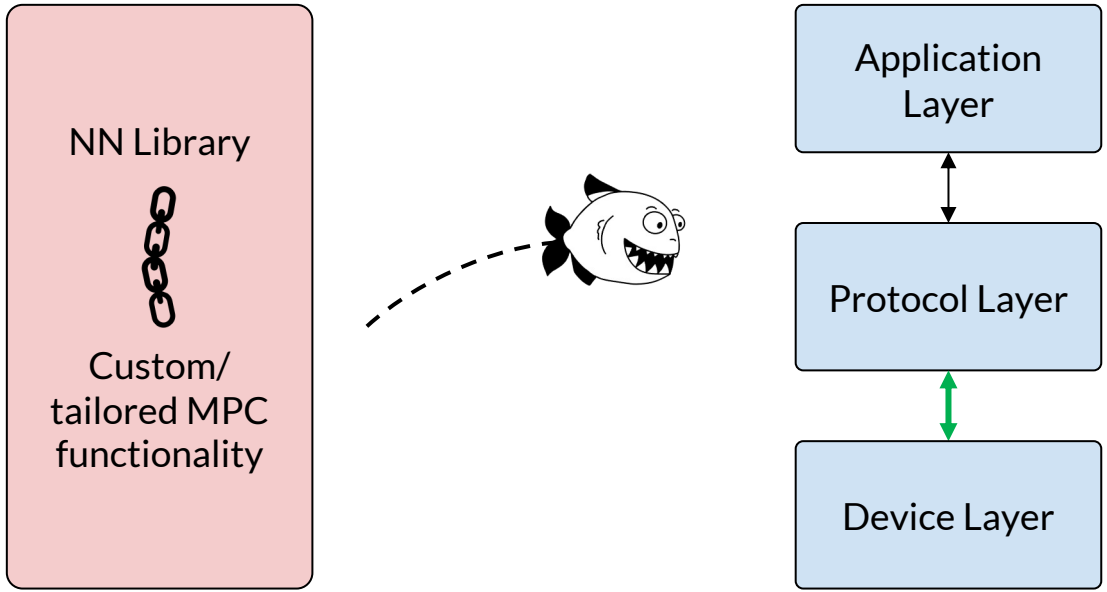
High-level logic

Composing computation and communication

Accelerating local computation

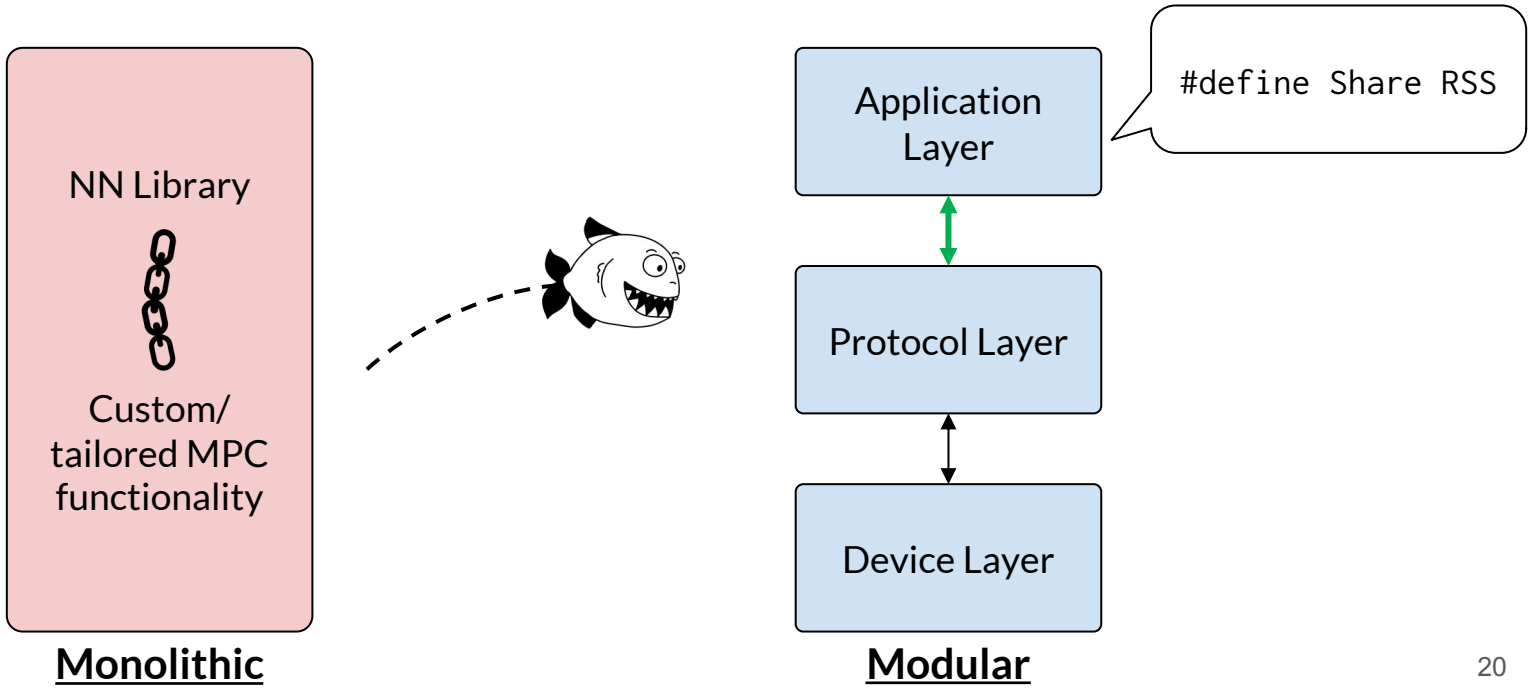
Acceleration is protocol-independent

Piranha implements kernels for operations over **local shares,** which any protocol can use.

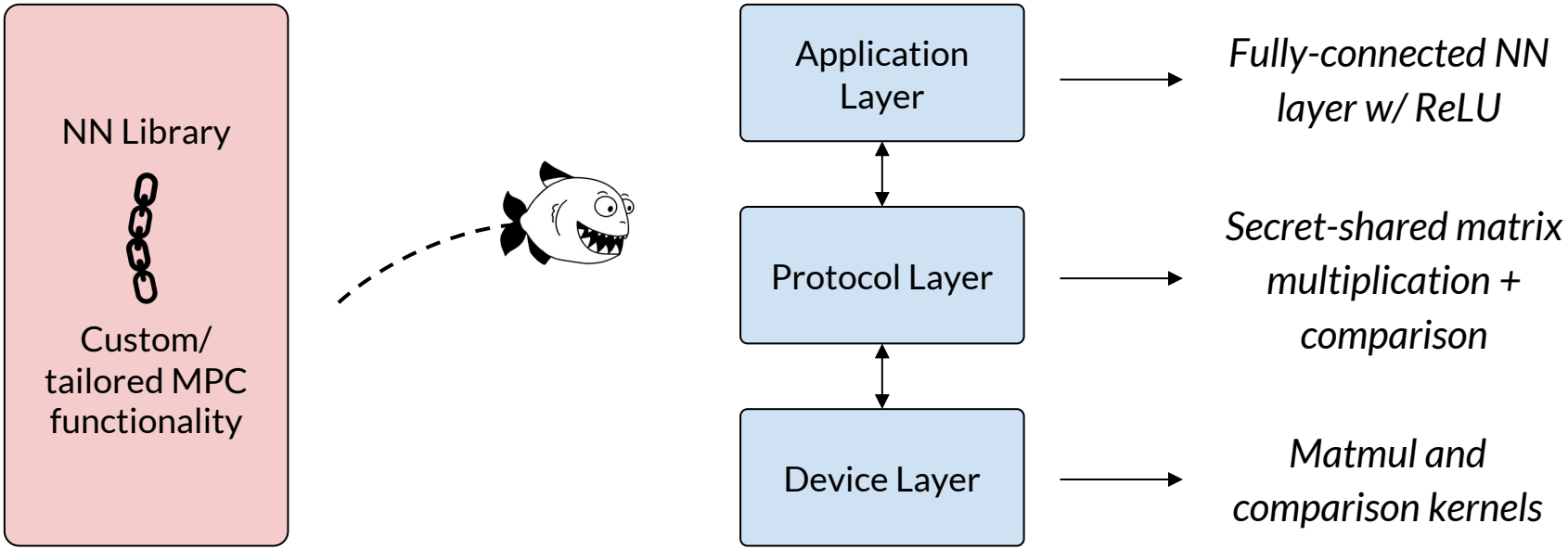


Applications change protocols with one `#define`

Applications see **opaque vectorized data types** defined by each protocol.



Piranha's architecture in practice



Overview

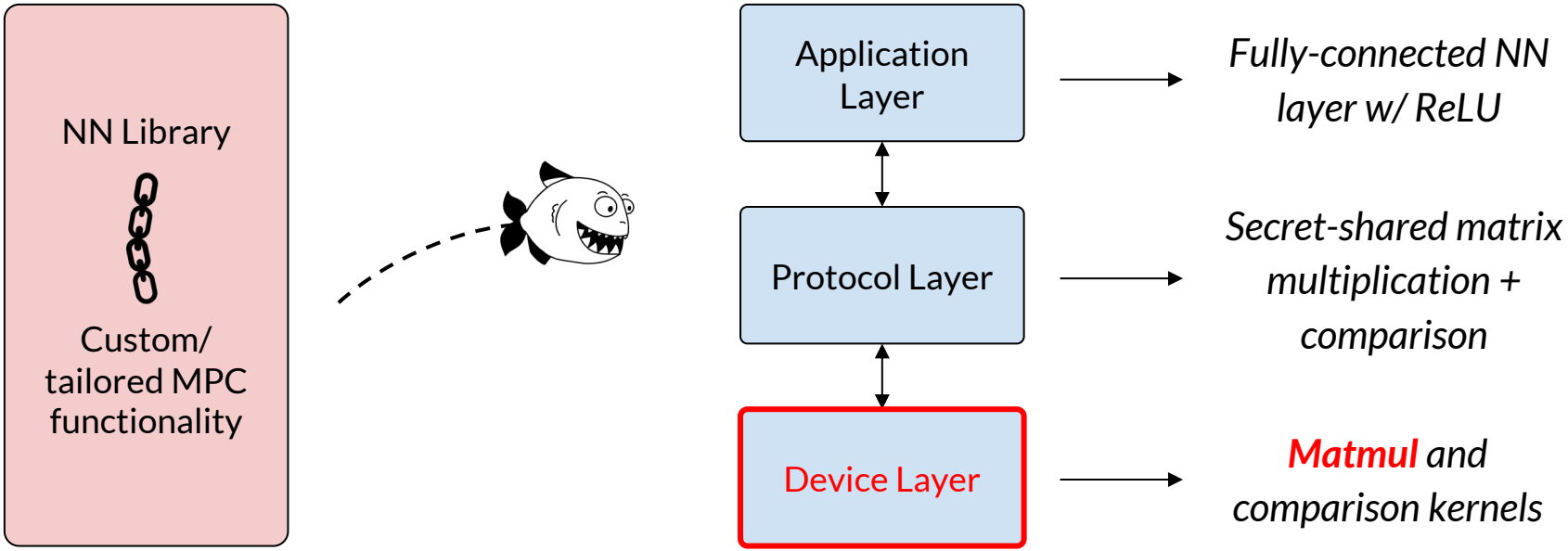
Bringing MPC to the GPU with **Piranha**

Piranha's architecture

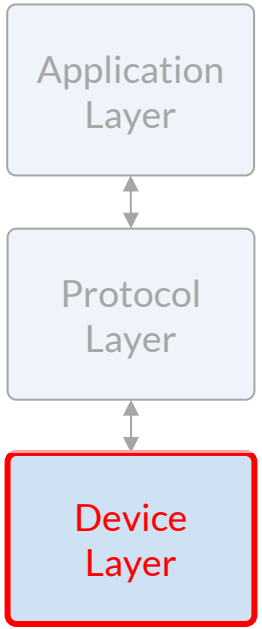
Key challenges: acceleration and memory

Evaluation

Problem 1: Performant linear operations for MPC



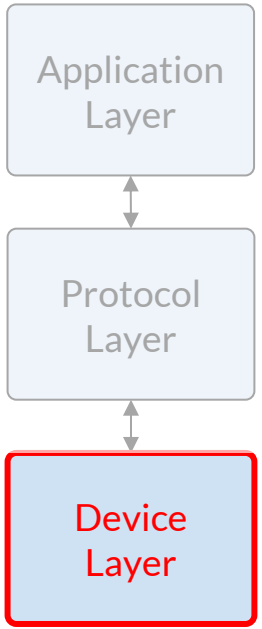
(1) Integer-based GPU acceleration is missing



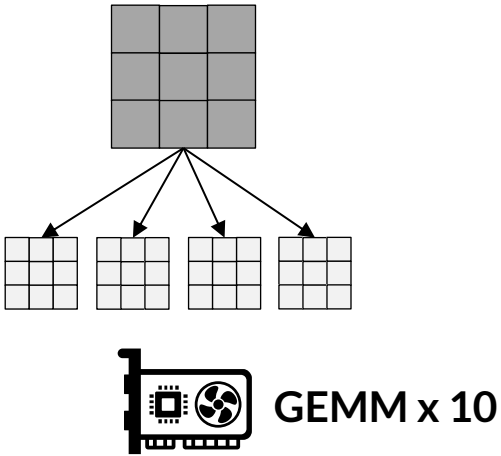
LSS protocols operate over integer rings and use *fixed point encoding* for ML training to encode real values.

Big issue: no performant kernels are available for integer GEMM (general matrix multiplication)

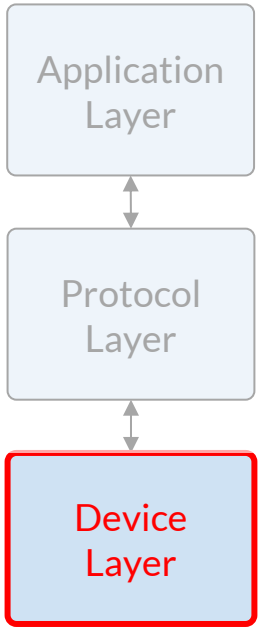
(1) Prior work adapts floating point kernels



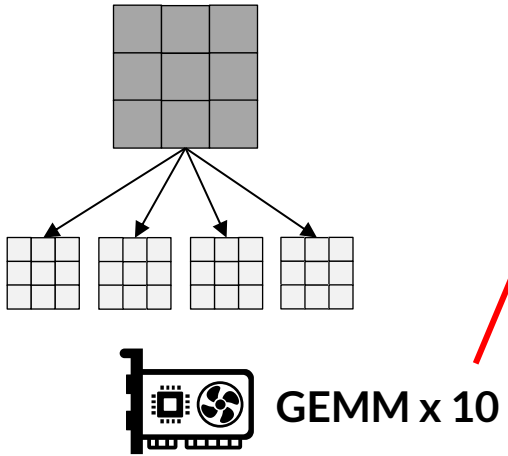
Prior work [TKT+21] splits 64-bit integers into 16-bit float chunks, incurring compute overhead.



(1) Prior work adapts floating point kernels

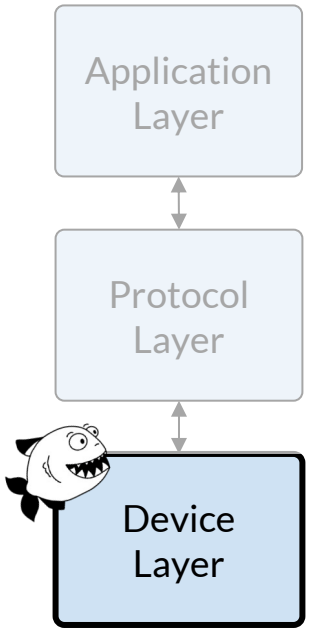


Prior work [TKT+21] splits 64-bit integers into 16-bit float chunks, incurring compute overhead.



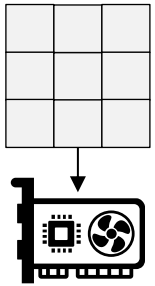
Assumes floating point performance outweighs overhead.

(1) Piranha directly uses GPU integer cores



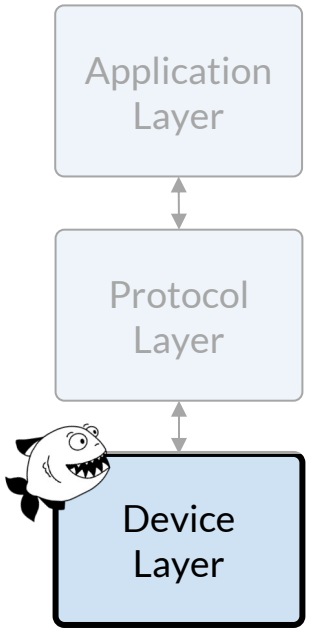
Piranha provides integer kernels directly to MPC protocols

We implement **32/64-bit integer** kernels with CUTLASS[1].



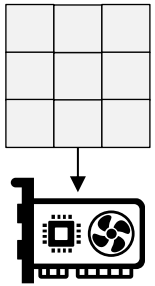
[1] https://github.com/NVIDIA/cutlass

(1) Piranha directly uses GPU integer cores



Piranha provides integer kernels directly to MPC protocols

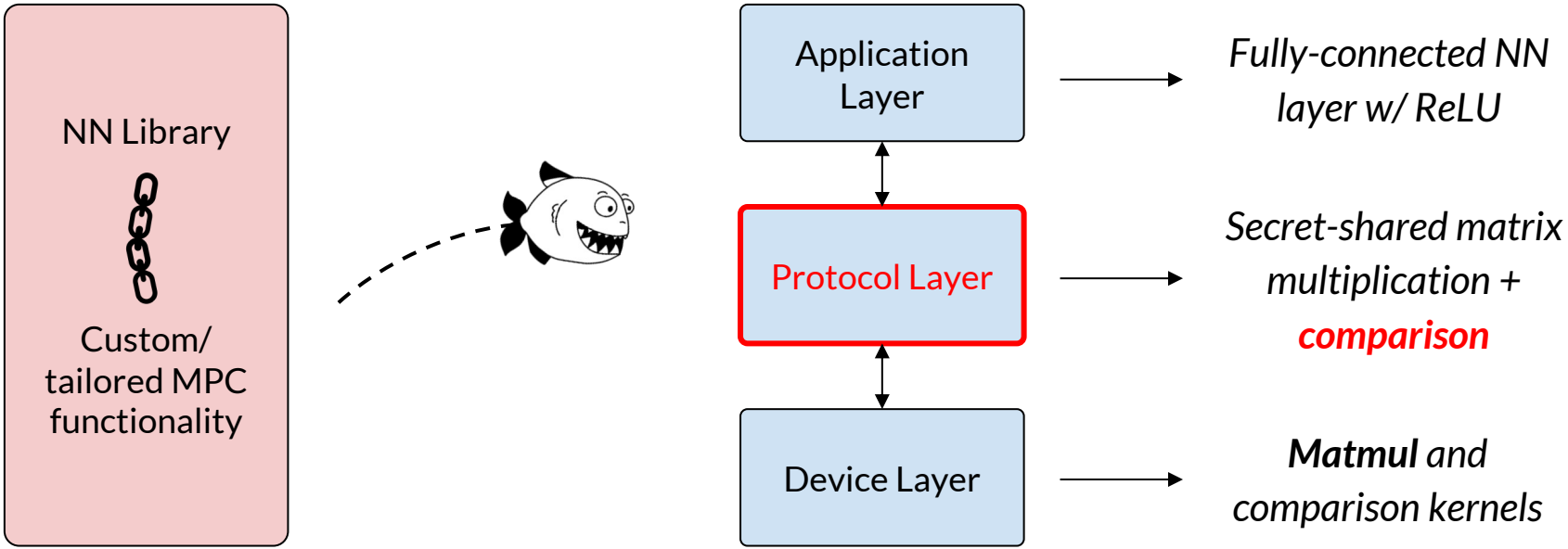
We implement **32/64-bit integer** kernels with CUTLASS[1].



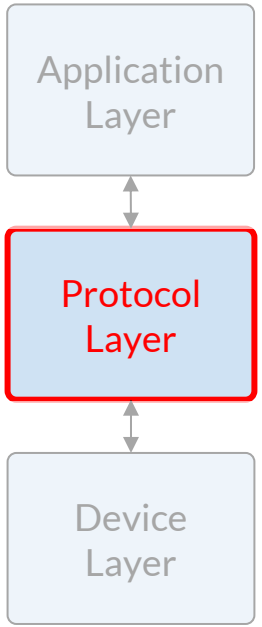
10x cuBLAS `f64`: 47 ms | Piranha `int64`: **4.9 ms**

[1] https://github.com/NVIDIA/cutlass

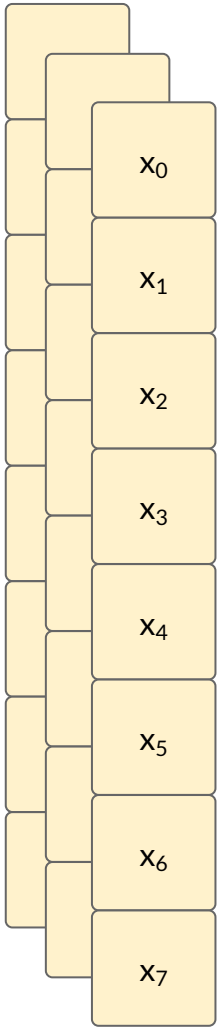
Problem 2: Memory-efficient comparisons



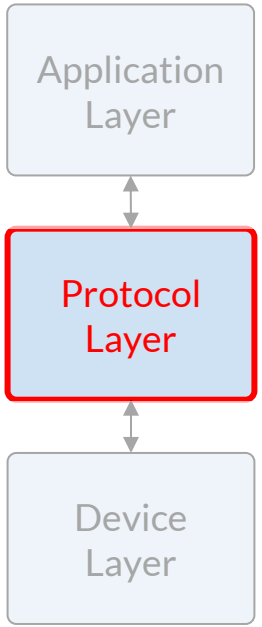
(2) MPC rapidly consumes GPU memory



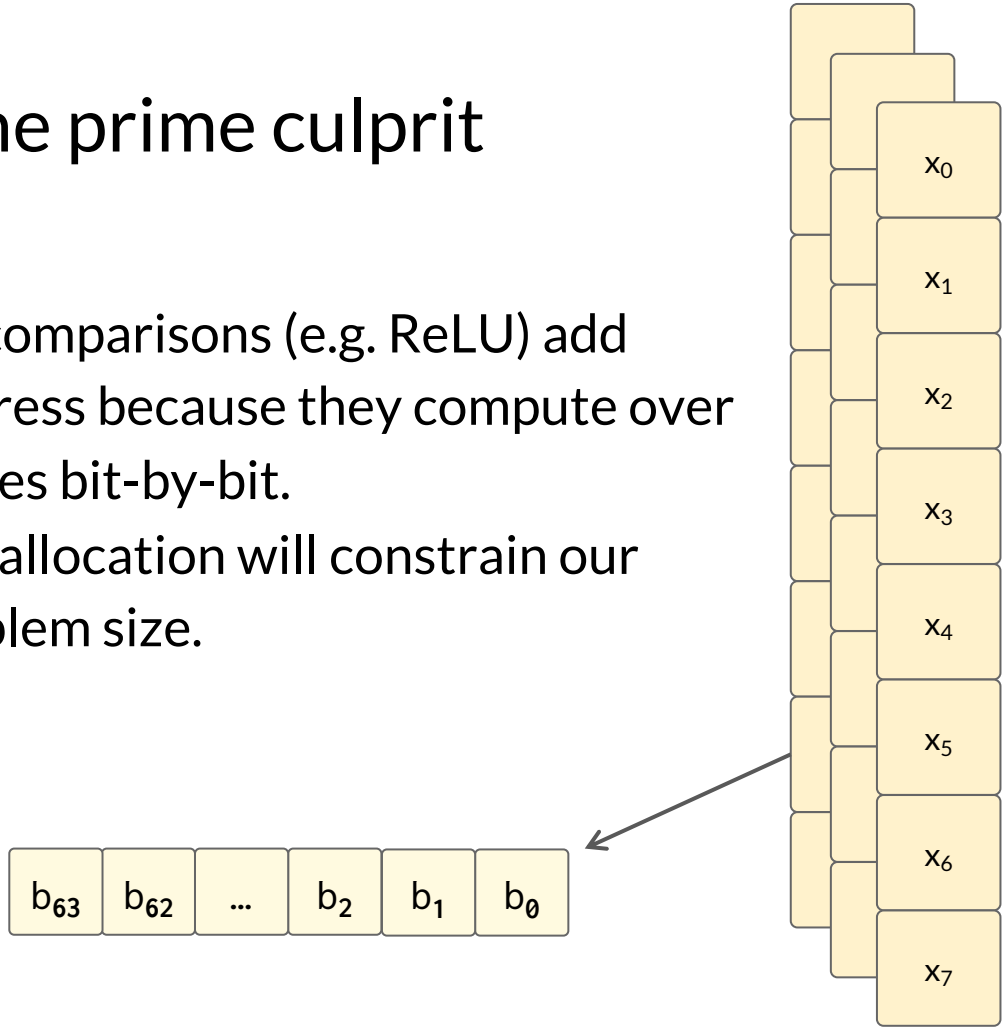
- **The issue:** Secret-sharing induces data duplication that stresses on-GPU memory.



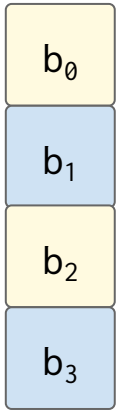
(2) Comparisons are the prime culprit



- Oblivious comparisons (e.g. ReLU) add memory stress because they compute over secret values bit-by-bit.
- Additional allocation will constrain our useful problem size.

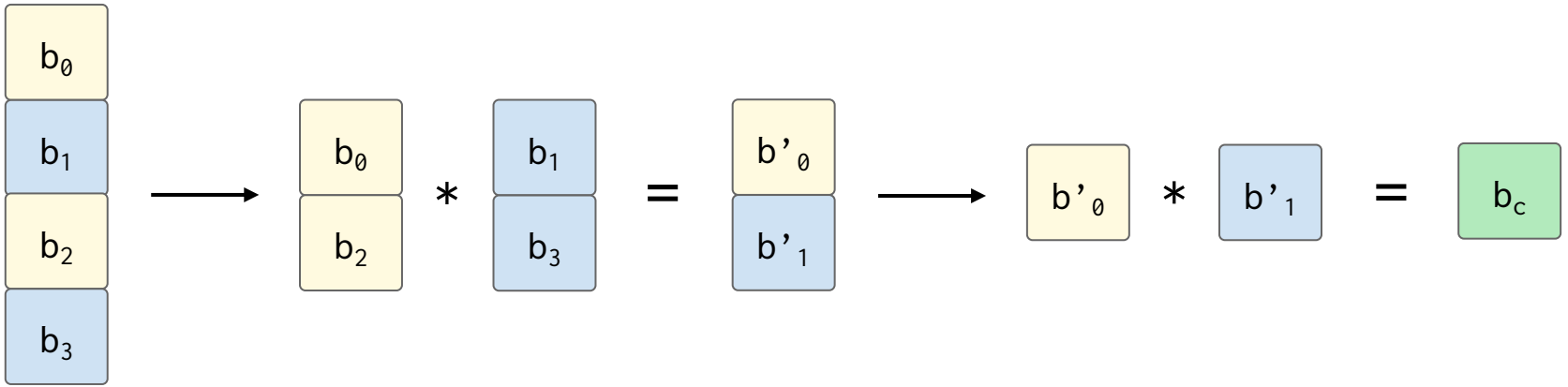


(2) Naïve string multiplication



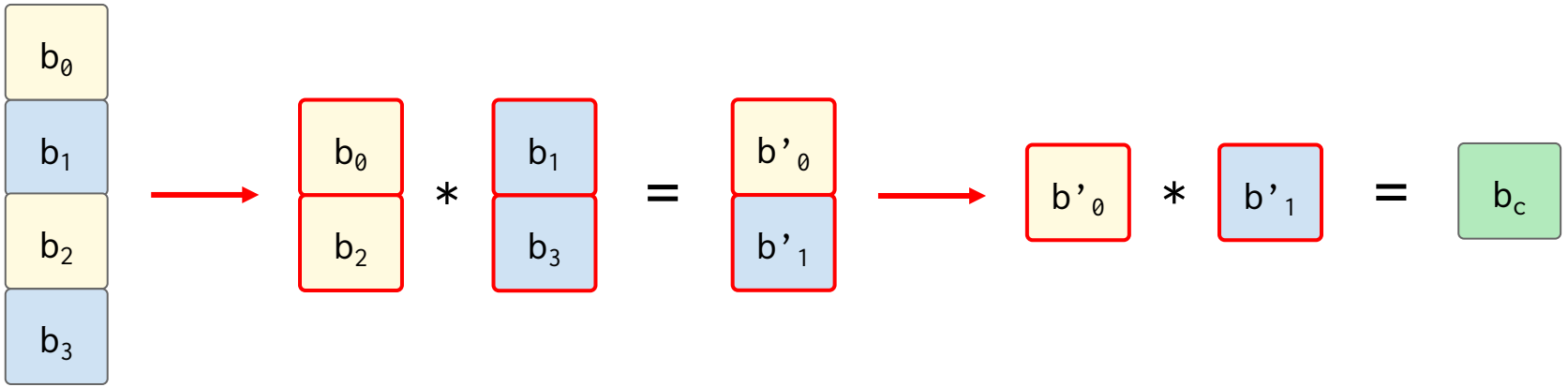
$$b_c = \prod_i b_i$$

(2) Naïve string multiplication



$$b_c = \prod_i b_i$$

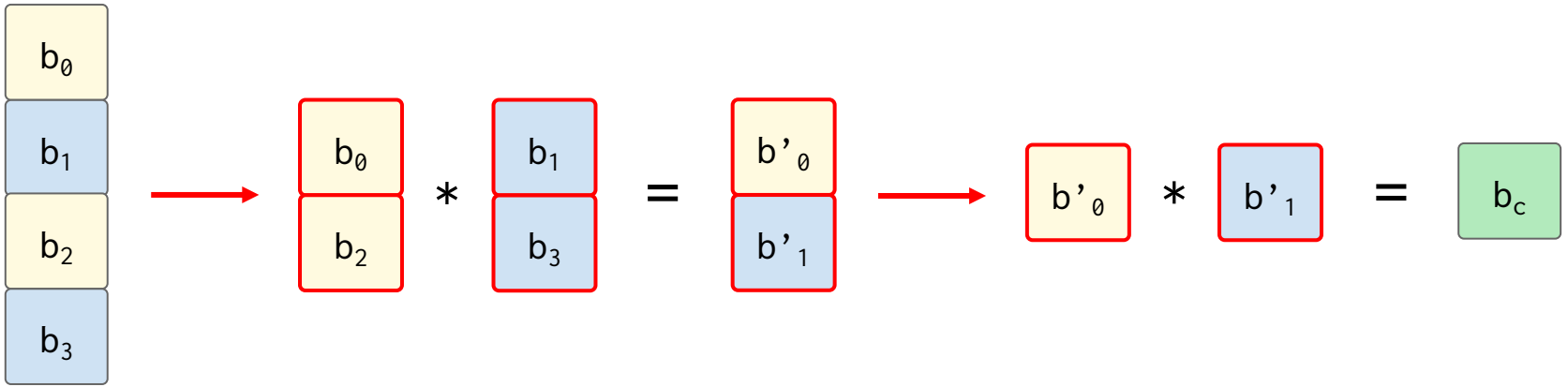
(2) The naïve protocol wastes memory



$$b_c = \prod_i b_i$$

(2) Iterator-based views keep memory in one place

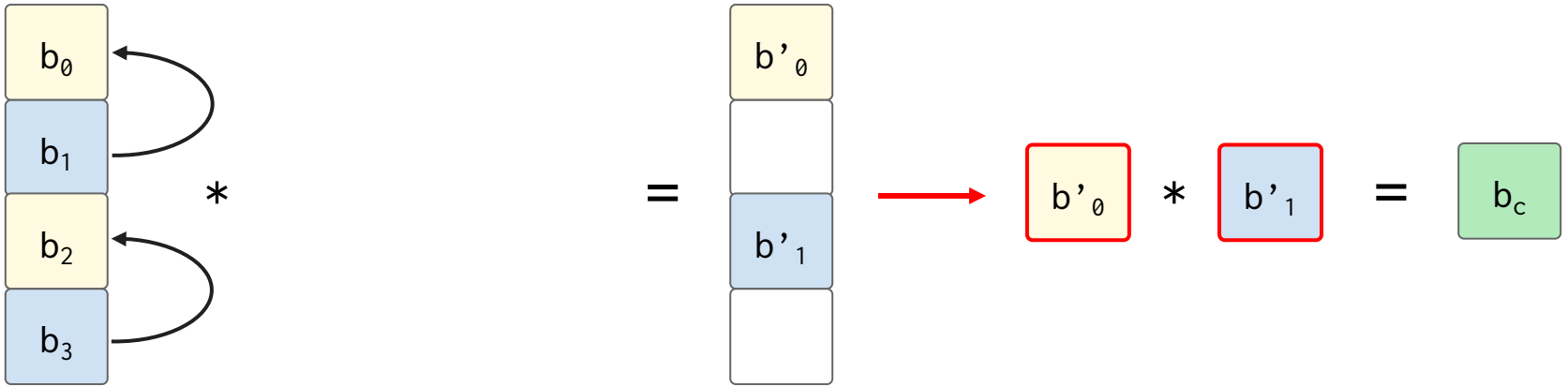
- Piranha allows protocols to use **iterator-based views for intricate data access patterns**:



$$b_c = \prod_i b_i$$

(2) Iterator-based views keep memory in one place

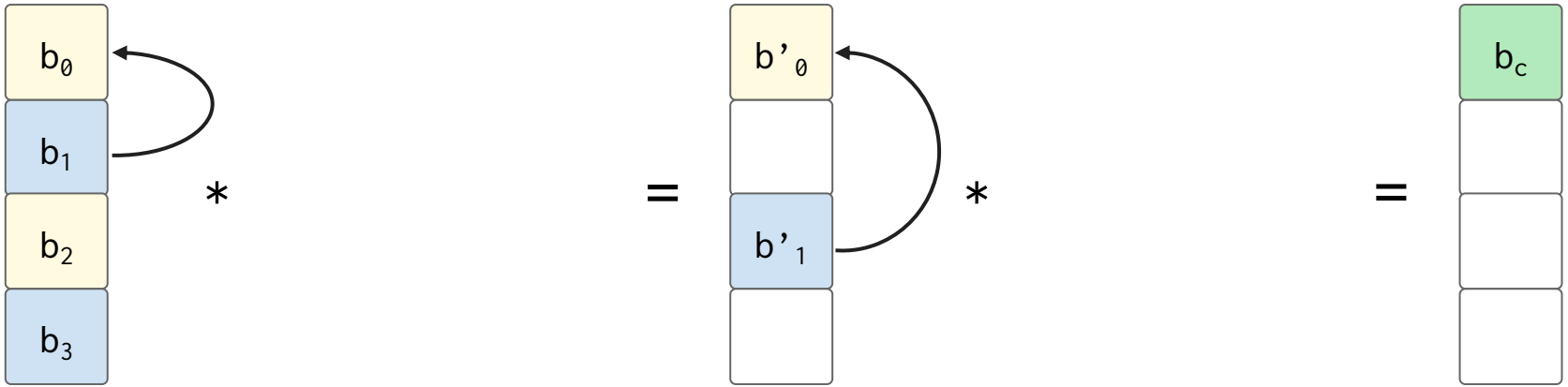
- Piranha allows protocols to use **iterator-based views for intricate data access patterns**:



$$b_c = \prod_i b_i$$

(2) Iterator-based views keep memory in one place

- Piranha allows protocols to use **iterator-based views for intricate data access patterns**:



$$b_c = \prod_i b_i$$

Overview

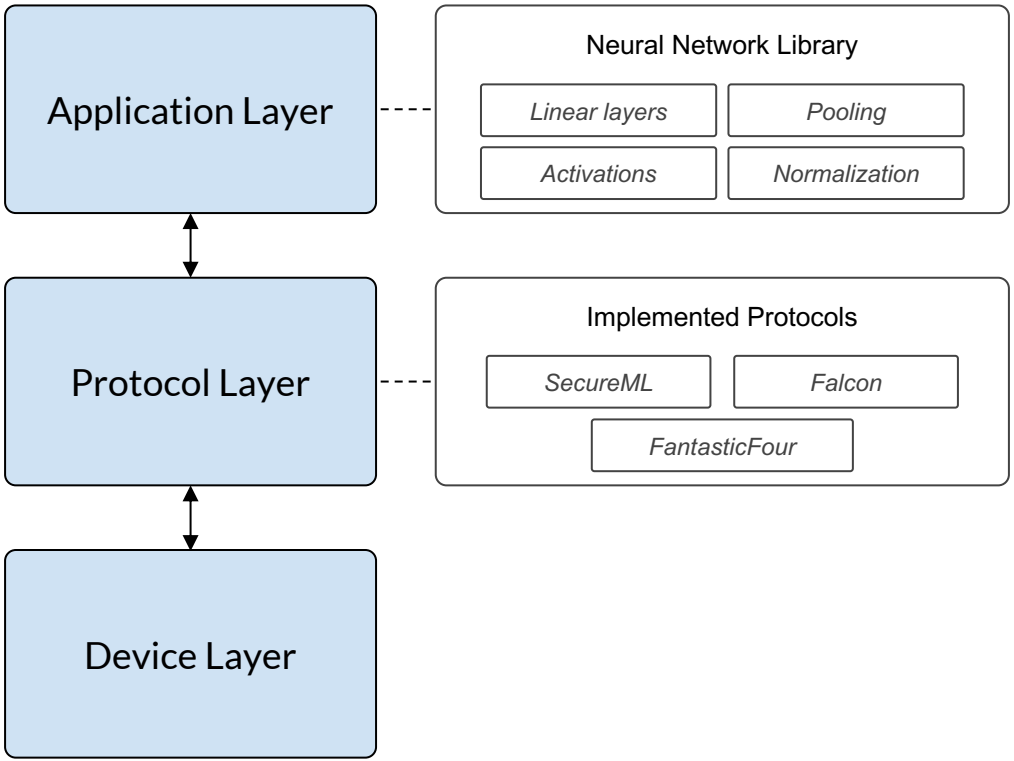
Bringing MPC to the GPU with **Piranha**

Piranha's architecture

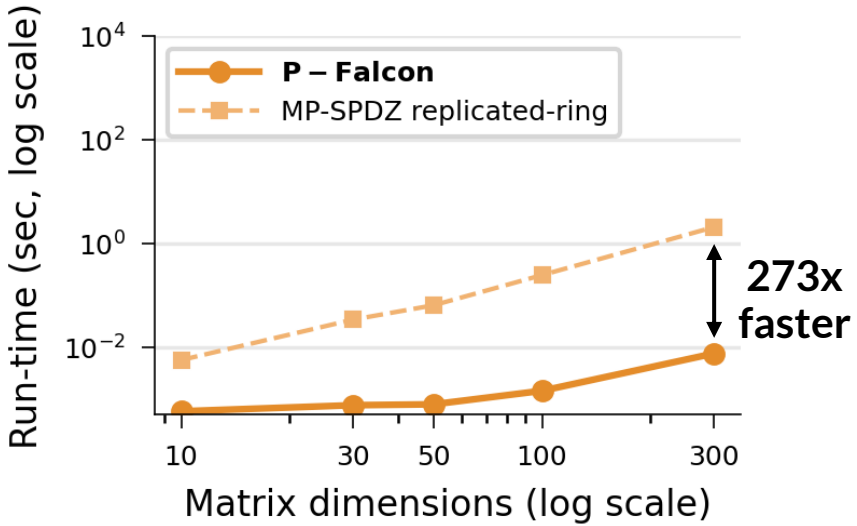
Key challenges: acceleration and memory

Evaluation

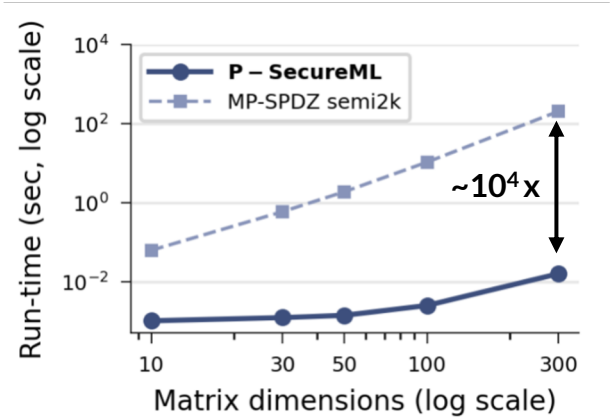
Developing with Piranha

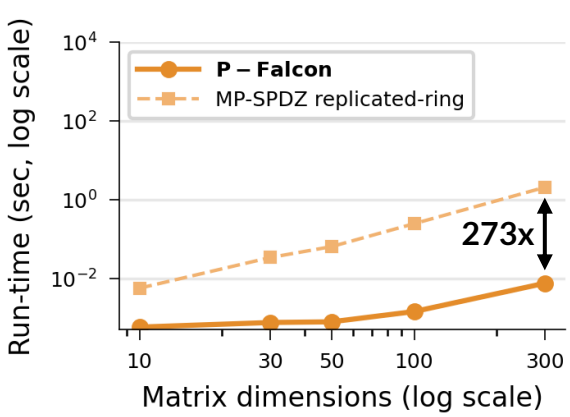


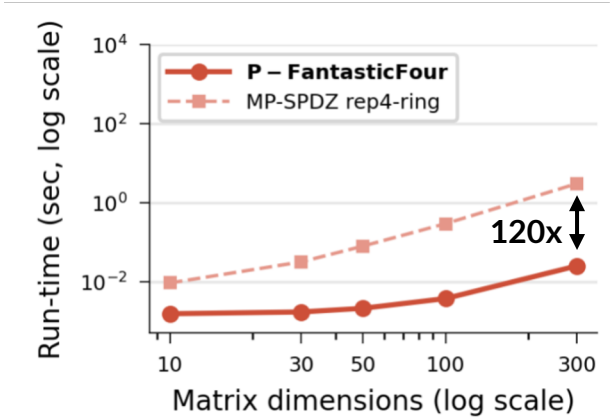
Microbenchmarks: is Piranha *performant*?



Microbenchmarks: is Piranha *performant*?

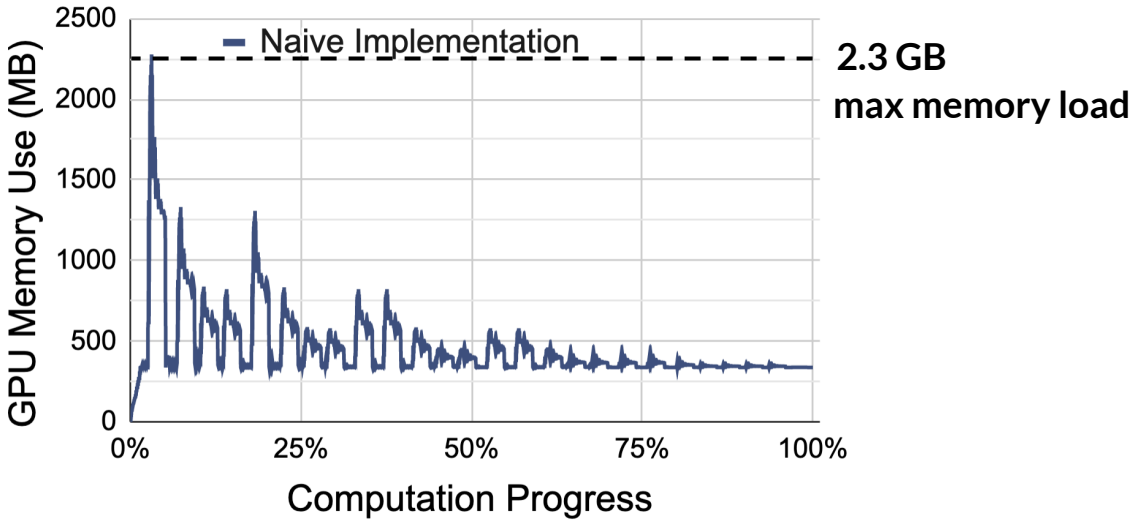




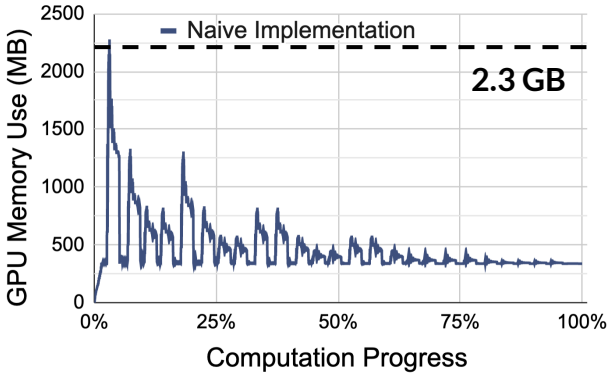


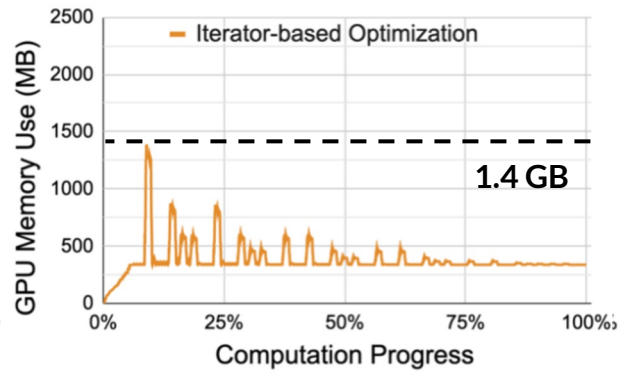
Piranha boosts performance by several orders of magnitude *across a range implemented MPC protocols.*

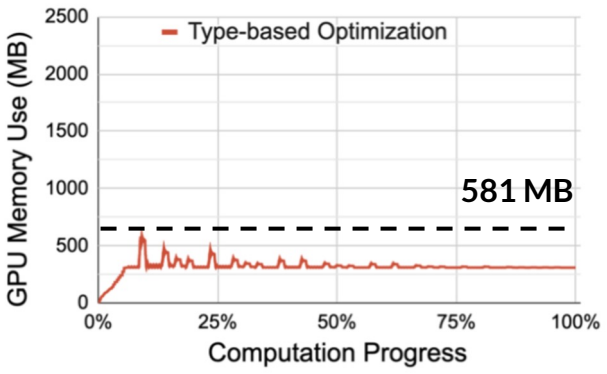
Memory Efficiency



Memory Efficiency







Iterator-based and correct typing allows Piranha to *drastically reduce on-device memory consumption.*

End-to-end training: is Piranha *usable*?

Falcon estimated that the same training run would take it **14 days** on a CPU

Piranha accelerates a 3-party protocol to complete 10 epochs of VGG16 training in just **33** hours!

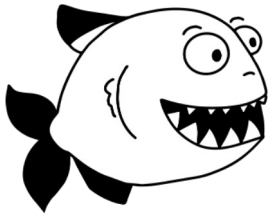
Network (Dataset)	Protocol	Time (min)	Comm. (GB)	Accuracy	
				Train (%)	Test (%)
SecureML (MNIST)	P-SecureML	12.99	49.55	97.37	96.56
	P-Falcon	7.51	22.84	97.37	96.56
	P-FantasticFour	23.39	33.01	97.37	96.56
LeNet (MNIST)	P-SecureML	87.55	683.18	96.78	96.80
	P-Falcon	71.56	485.90	96.88	97.10
	P-FantasticFour	219.20	676.13	96.88	97.11
AlexNet (CIFAR10)	P-SecureML	156.01	740.50	40.74	40.47
	P-Falcon	110.66	382.18	40.59	40.71
VGG16 (CIFAR10)	P-Falcon	1979.92	17235.35	55.13	54.26
	P-FantasticFour	7697.54	29106.24	55.02	54.35

Summary

Piranha is a general-purpose platform for accelerating MPC on GPUs.

Use our code to build new protocols and implement new applications!

`github.com/ucbrise/piranha`



Jean-Luc Watson | `jlw@berkeley.edu`