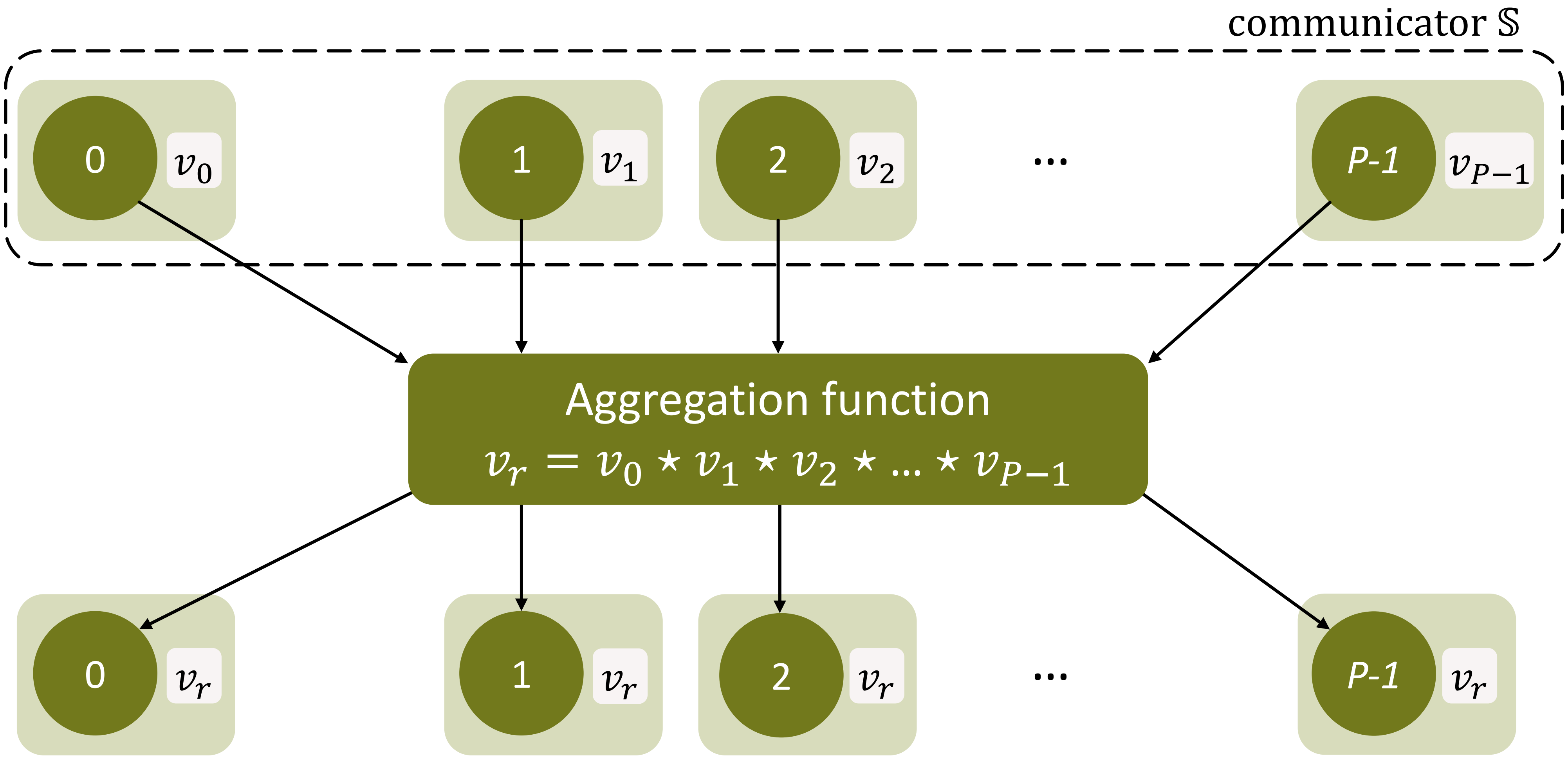
Marcin Chrapek, Mikhail Khalilov, Torsten Hoefler

HEAR: Homomorphically Encrypted Allreduce

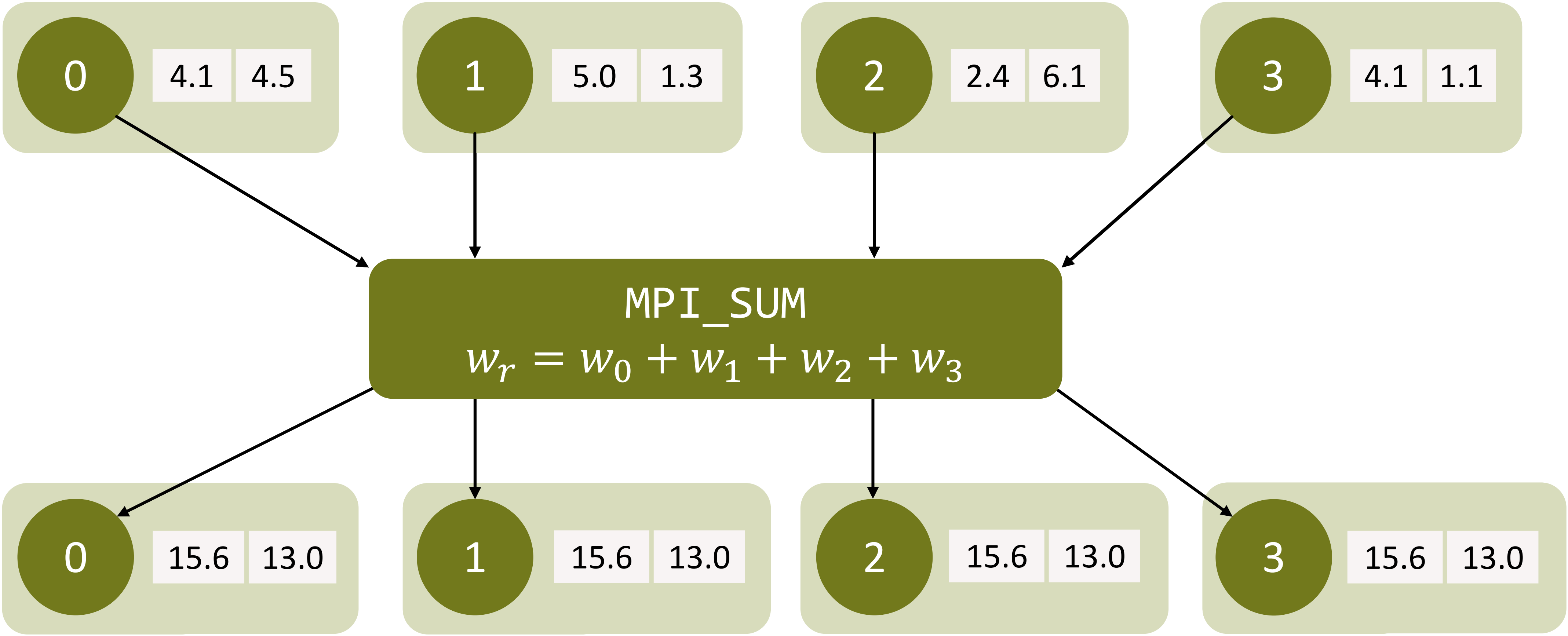
Artifacts Available
Artifacts Evaluated Functional
Results Replicated

SPCL

Allreduce


communicator $\mathbb{S}$
0
v_0
1
v_1
2
v_2
…
P-1
v_{P-1}
Aggregation function
$v_r = v_0 \star v_1 \star v_2 \star \ldots \star v_{P-1}$
0
v_r
1
v_r
2
v_r
…
P-1
v_r

Stochastic gradient descent (SGD)



Dominance of Allreduce

A Large-Scale Study of MPI Usage in Open-Source HPC Applications

Ignacio Laguna
Lawrence Livermore National Laboratory
ilaguna@llnl.gov

Ryan Marshall
University of Tennessee, Chattanooga
ryan-marshall@utc.edu

Kathryn Mohror
Lawrence Livermore National Laboratory
mohror1@llnl.gov

Martin Ruefenacht
University of Tennessee, Chattanooga
martin-ruefenacht@utc.edu

Anthony Skjellum
University of Tennessee, Chattanooga
Tony-Skjellum@utc.edu

Nawrin Sultana
Auburn University
nzs0034@auburn.edu

A survey of MPI usage in the U. S. Exascale Computing Project†

David E. Bernholdt[1] | Swen Boehm[1] | George Bosilca[2] | Manjunath Gorentla Venkata[1] | Ryan E. Grant[3] | Thomas Naughton[1] | Howard P. Pritchard[4] | Martin Schulz[5,6] | Geoffroy R. Vallee[1]

Characterization of MPI Usage on a Production Supercomputer

Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms and Kalyan Kumaran
Argonne National Laboratory,
{*sudheer, sparker, balaji, kharms, kumaran*}@*anl.gov*

Optimization of Collective Reduction Operations

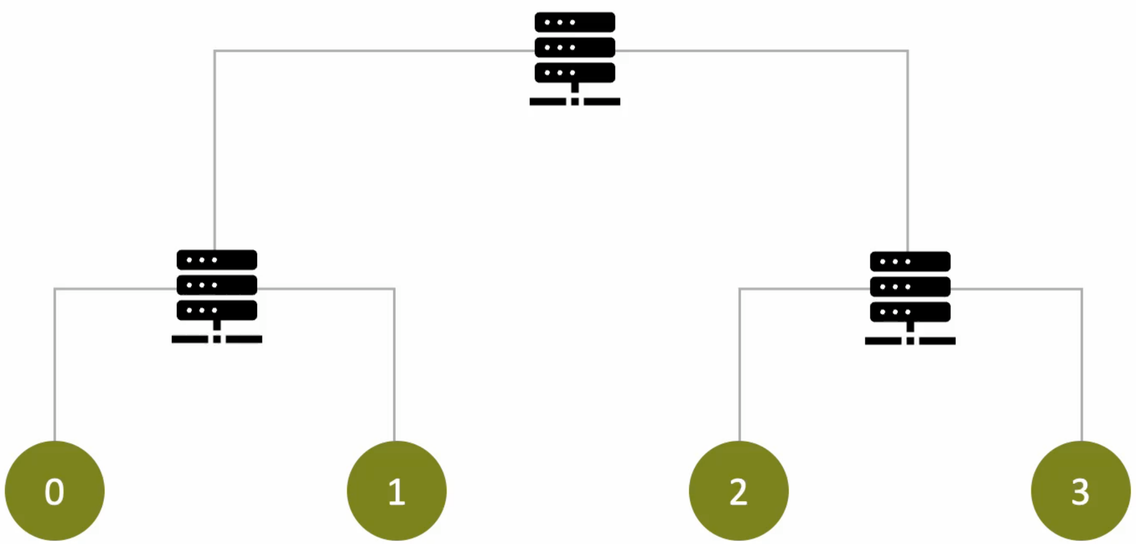
Rolf Rabenseifner

High-Performance Computing-Center (HLRS), University of Stuttgart
Allmandring 30, D-70550 Stuttgart, Germany
rabenseifner@hlrs.de,
www.hlrs.de/people/rabenseifner/

92% of common HPC applications use Allreduce

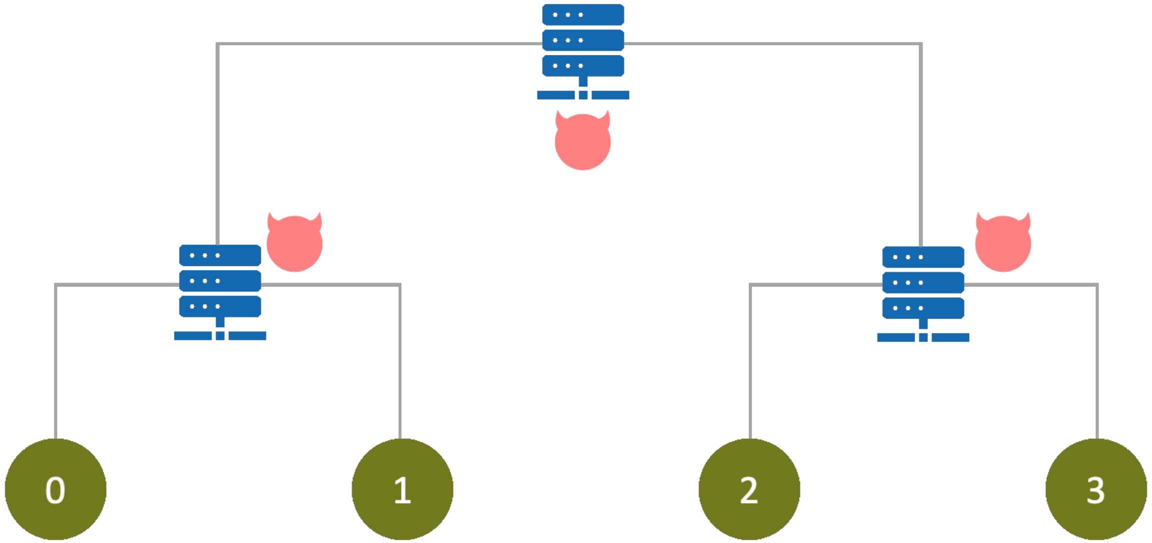
Up to 30% of all core hours spent in Allreduce

Non-accelerated Allreduce



Security achieved using end-to-end encryption

In-network computed Allreduce



- ✓ Lower latency (3-18x)
  - ✓ Higher performance (1.5-5.5x)
- ✓ Lower bandwidth usage (2x)
  - ✓ Lower power usage
  - ✓ Lower contention

Security?

“Security is essential to achieving the anticipated benefits of HPC […]”

“HPC […] environment is very different from ordinary IT. As such, security solutions must be tailored to the HPC system’s requirements[…]”

“HPC users may consider security valuable only to the extent that it does not significantly slow down the HPC system.”

NIST Special Publication
NIST SP 800-223 ipd

High-Performance Computing (HPC) Security:

Architecture, Threat Analysis, and Security Posture

Initial Public Draft

Yang Guo
Ramaswamy Chandramouli
Lowell Wofford
Rickey Gregg
Gary Key
Antwan Clark
Catherine Hinton
Andrew Prout
Albert Reuther
Ryan Adamson
Aron Warren
Purushotham Bangalore
Erik Deumens
Csilla Farkas

But how?

Q: But how can we reduce if data is needed in plain for processing?

A: Confidential computing (CC)
Compute operator does not know the data their system evaluates

Trusted Execution Environments (TEEs)
Black box creating isolated, secure environment protecting sensitive data and code from outside parties.
SGX, TDX, SEV SNP, etc.

Homomorphic Encryption (HE)

$$E(x \star y) = E(x) \star E(y)$$

Scaling issues (context switch, sharing keys)
Increased latency (encryption, decryption)
Requires considerable hardware changes

x = plaintext / message
$E(x)$ = ciphertext

Encryption challenges

R1

Ciphertext at most 2x plaintext

$len(E(x)) < 2len(x)$

R2

Unlimited operation count

Unlimited number of operations without refreshing the ciphertext.

R3

Efficient implementation

Encryption, decryption, and homomorphic operations need to be performant.

R4

Multiple operation types supported

We want most of the common MPI operations not just one.

State of the art homomorphic encryption not fulfilling these

HEAR the idea

Idea

Introduce a symmetric scheme based on **ring noise scrambling**

$$E(x) = x \star \text{noise}$$
$$D(x) = x \star \text{noise}^{-1}$$

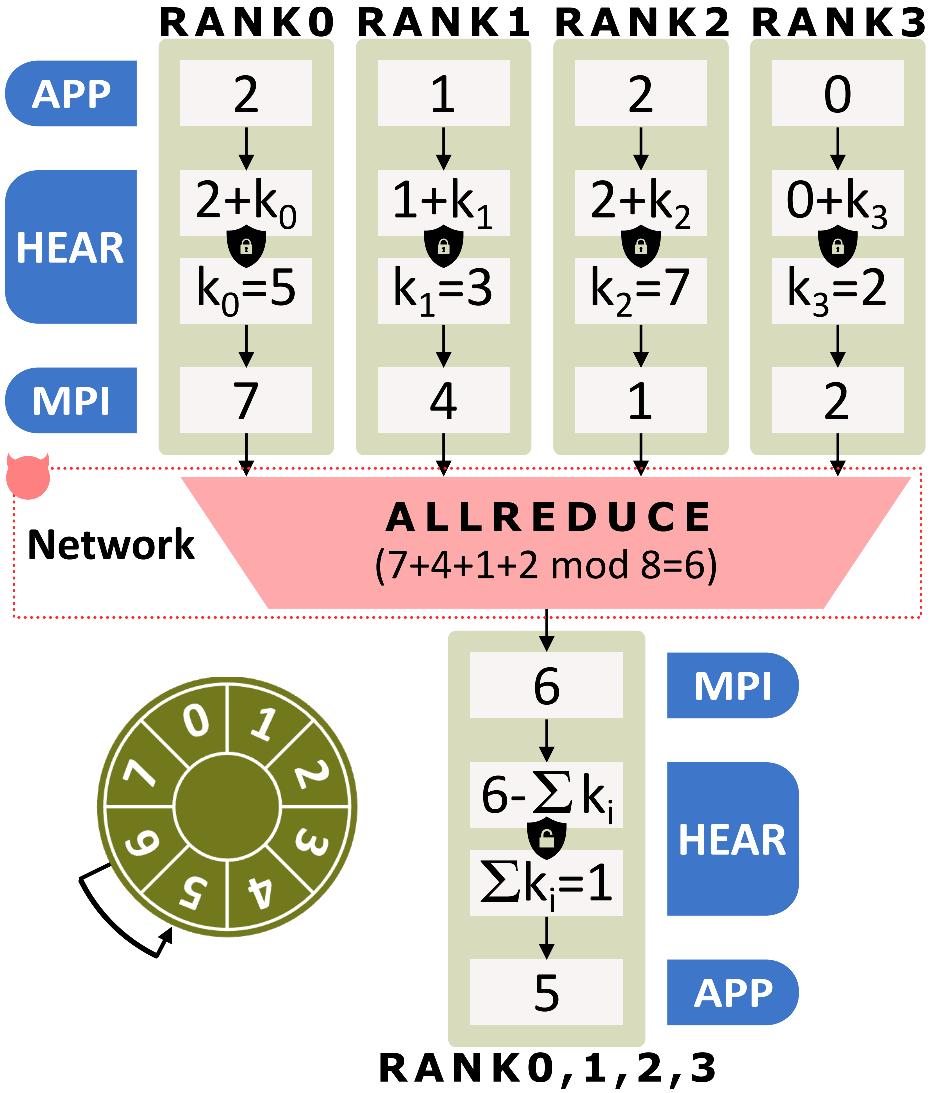
Reduction happens without any changes to the hardware

The operations are performant

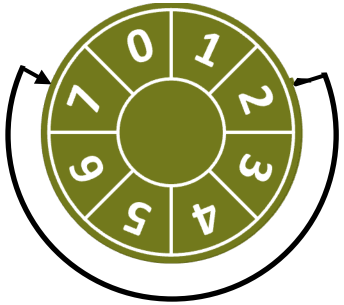
No increased bandwidth usage

No loss of information

Example integer summation



The adversary does not know where on the ring we are



Data with noise

All ranks need to know the keys of other ranks

This means N^2 communication and storing N keys. Can we do better?

Key generation

Scalable O(1) state.

Step 1

Rank 0

1. generates a compound key k_c
2. shares them **securely** with all other ranks (end-to-end encryption).

Step 2

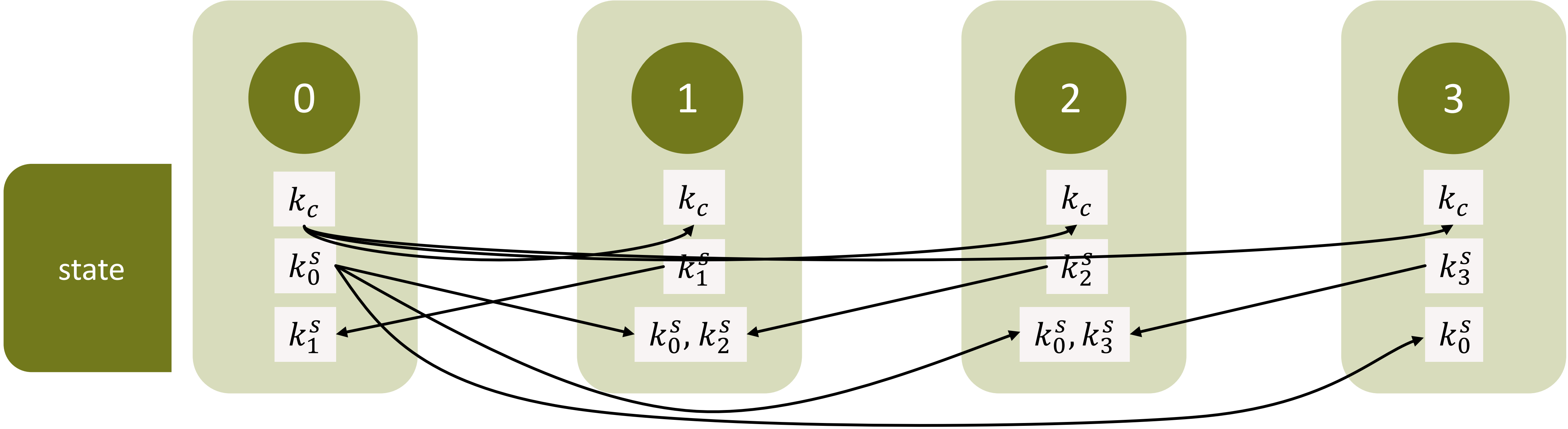
Each rank i

1. generates local starting key k_i^s
2. **securely** obtains the starting key of ranks 0 and the next rank.

Step 3

All ranks

Agree upon a pseudorandom function (PRF) $F_k(x)$ such as AES.



Encryption

Encryption in two PRF executions and two primitive operations.

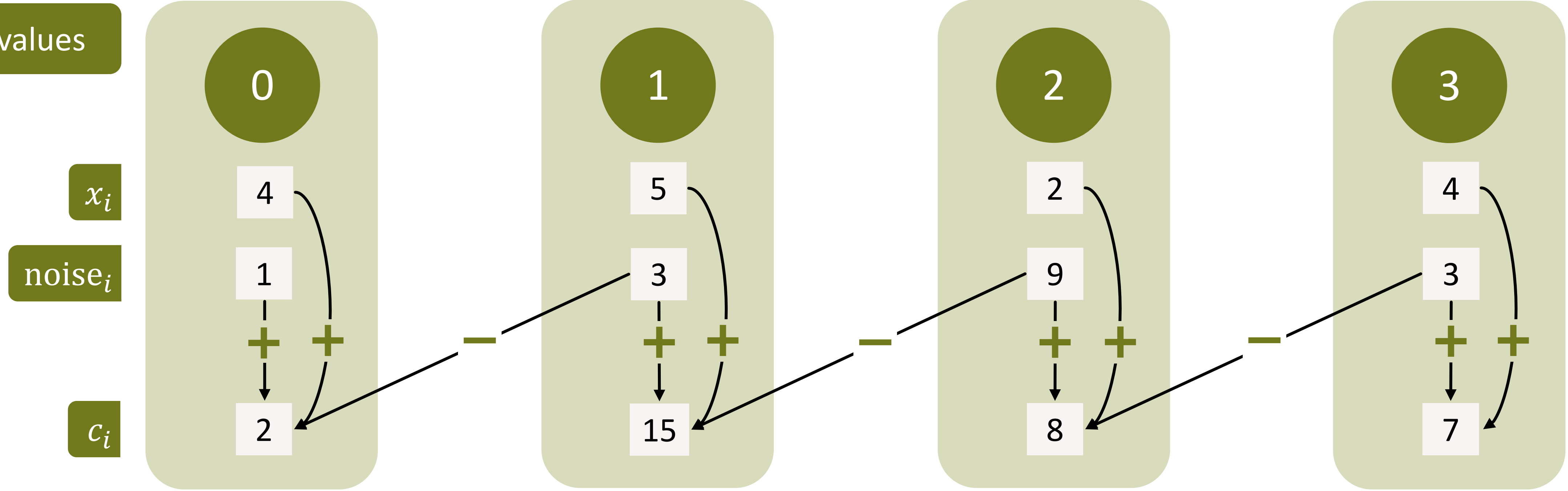
Step 4

Encrypt each element j of vector x_i to ciphertext c_i using one of the schemes HEAR defines. E.g., for integers:

$$c_i[j] = \begin{cases} x_i[j] + \mathrm{noise}_i[j], & i = P - 1 \\ x_i[j] + \mathrm{noise}_i[j] - \mathrm{noise}_{i+1}[j], & \text{otherwise} \end{cases}$$

noise_i is a PRF evaluation with node's i starting key and the compound key as inputs

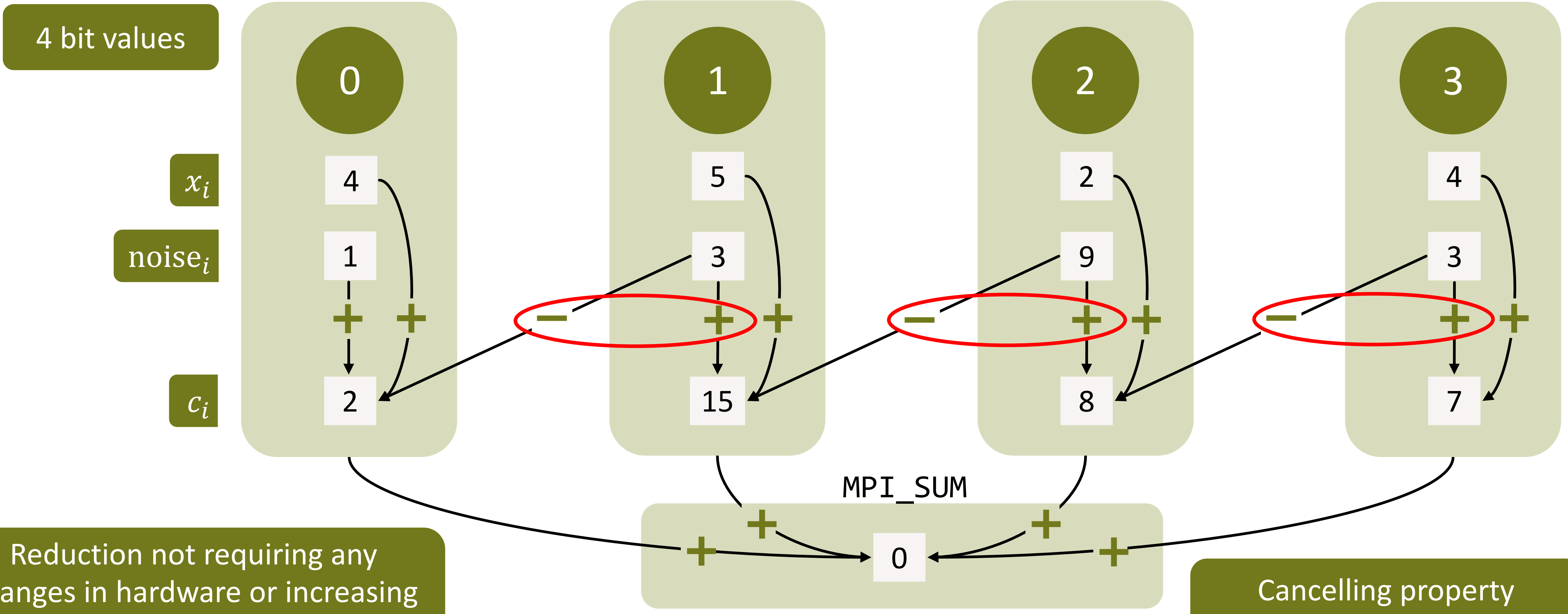
4 bit values



Reduction

Step 5

Reduce c_i in network. E.g., for integers: $c_r = \sum_{i=0}^{P-1} c_i$



Reduction not requiring any changes in hardware or increasing complexity.

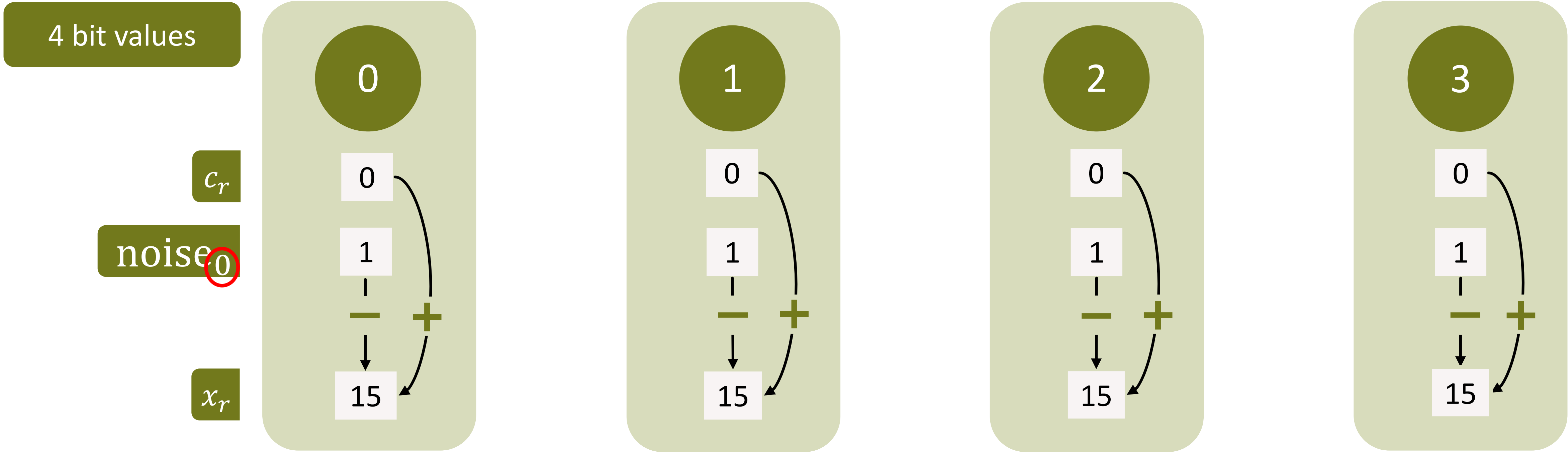
Cancelling property enabling faster decryption

Decryption

Step 6

Decrypt using simple primitives. E.g., for integers:

$$x_r[j] = c_r[j] - \text{noise}_0[j]$$



Decryption using one PRF evaluation, and one primitive operation.

LD_PRELOAD=libhear.so

Application

libhear (PMPI)

MPI implementation

- Small C++ middleware
- MPI implementation independent

In-network compute

Latency for small messages (e.g., 16B)

Throughput for large messages (e.g., 16MB)

Step 1

Get `libhear` and compile it

Step 2

`LD_PRELOAD` `libhear` while running the MPI job. E.g.,:

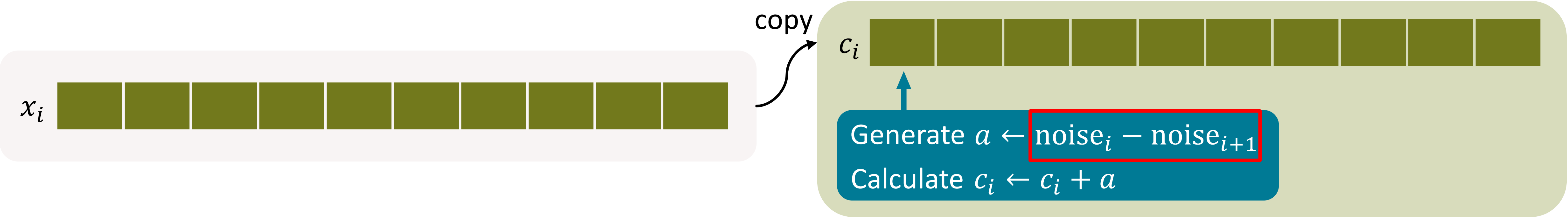
```
LD_PRELOAD=libhear.so mpirun -np 2 ./test
```

`libhear` is open-source

https://github.com/spcl/libhear

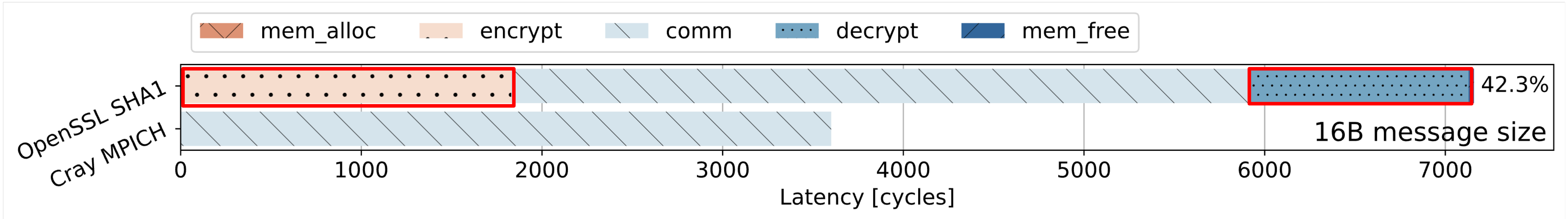
No recompilation or change of code.

Naïve implementation

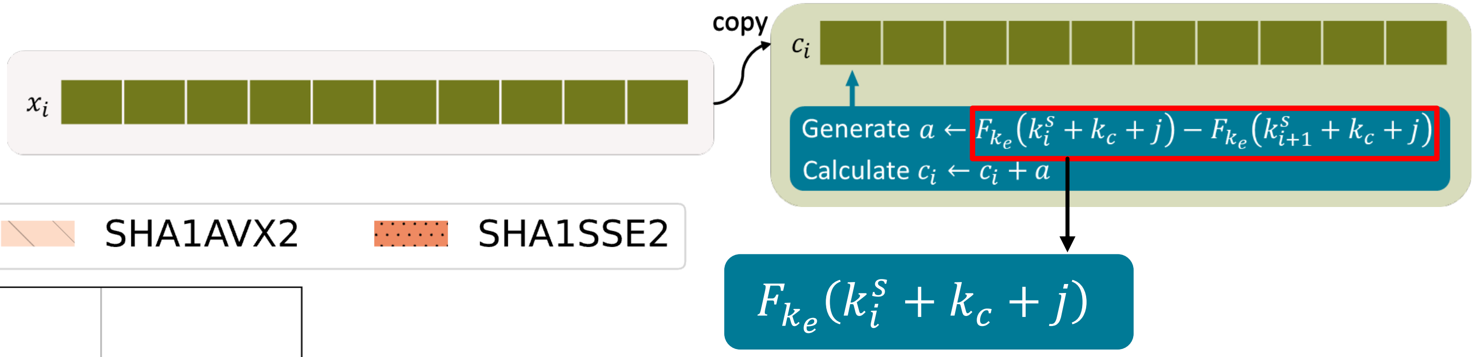


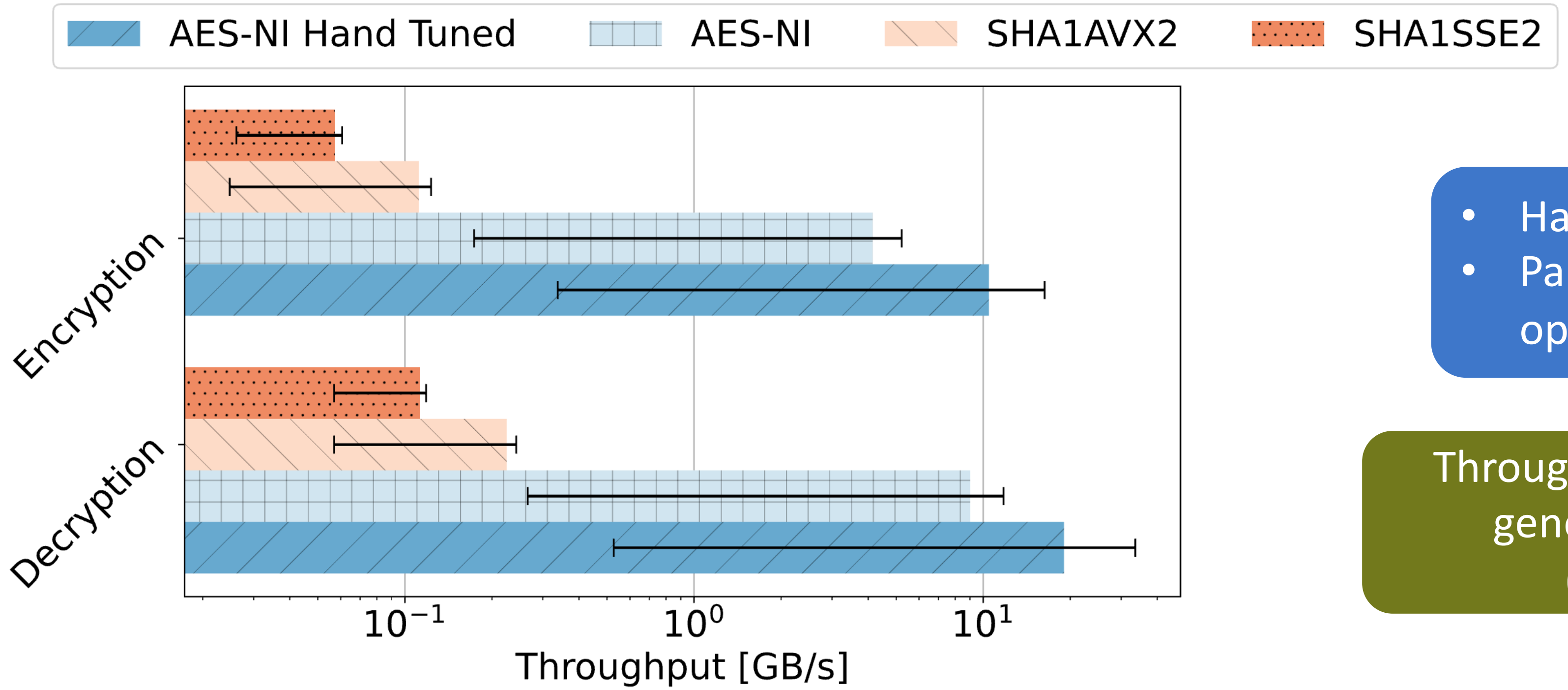
Integer summation

100Gbps Aries Interconnect on PizDaint



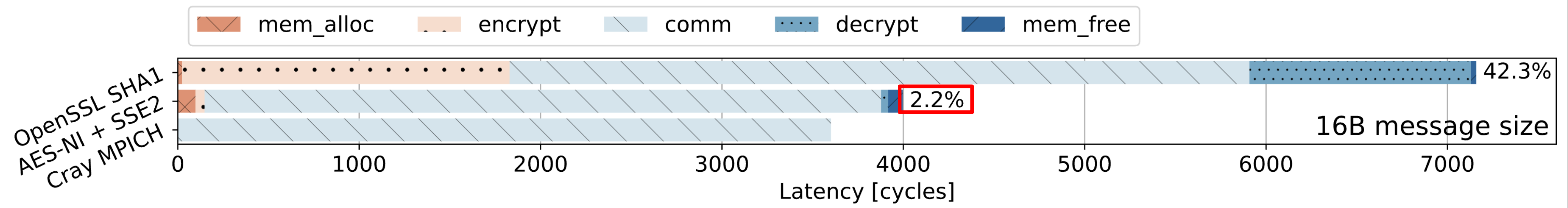
Performance optimizations



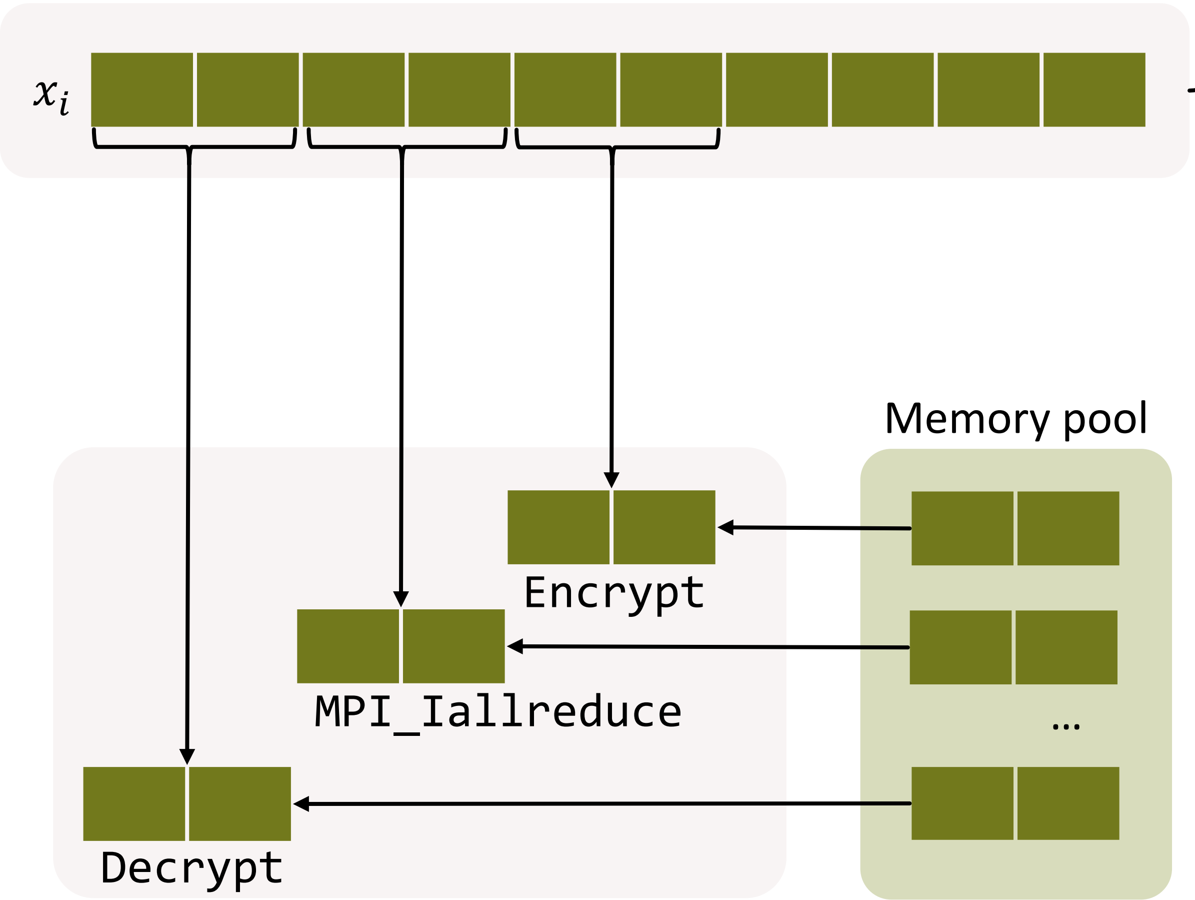


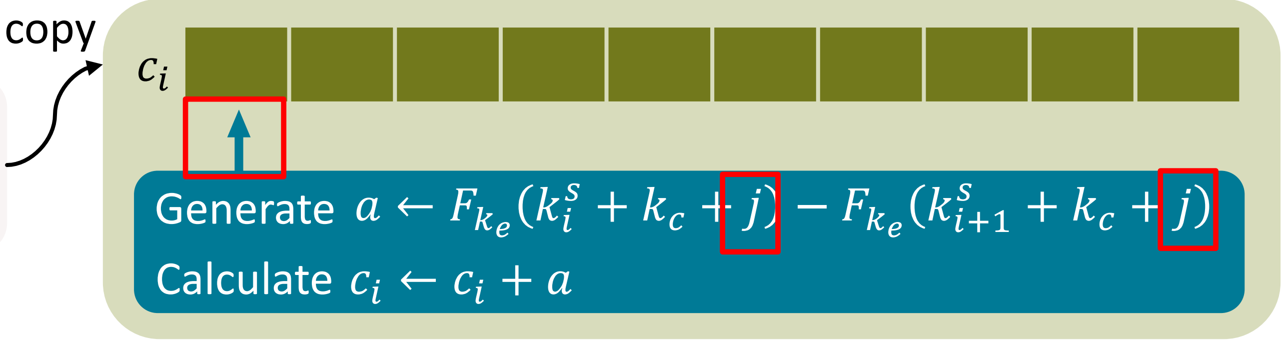
- Hash function evaluation
- Parallelizable primitive operations

Throughputs suitable for next generation of networks (400-800Gbit/s)

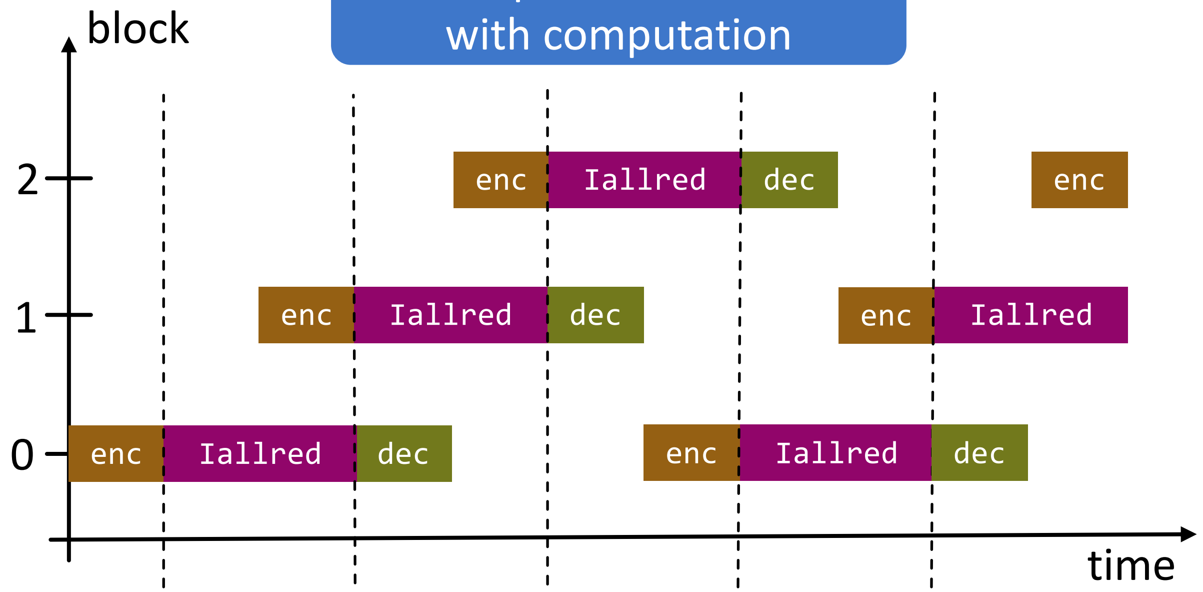


Further throughput optimizations



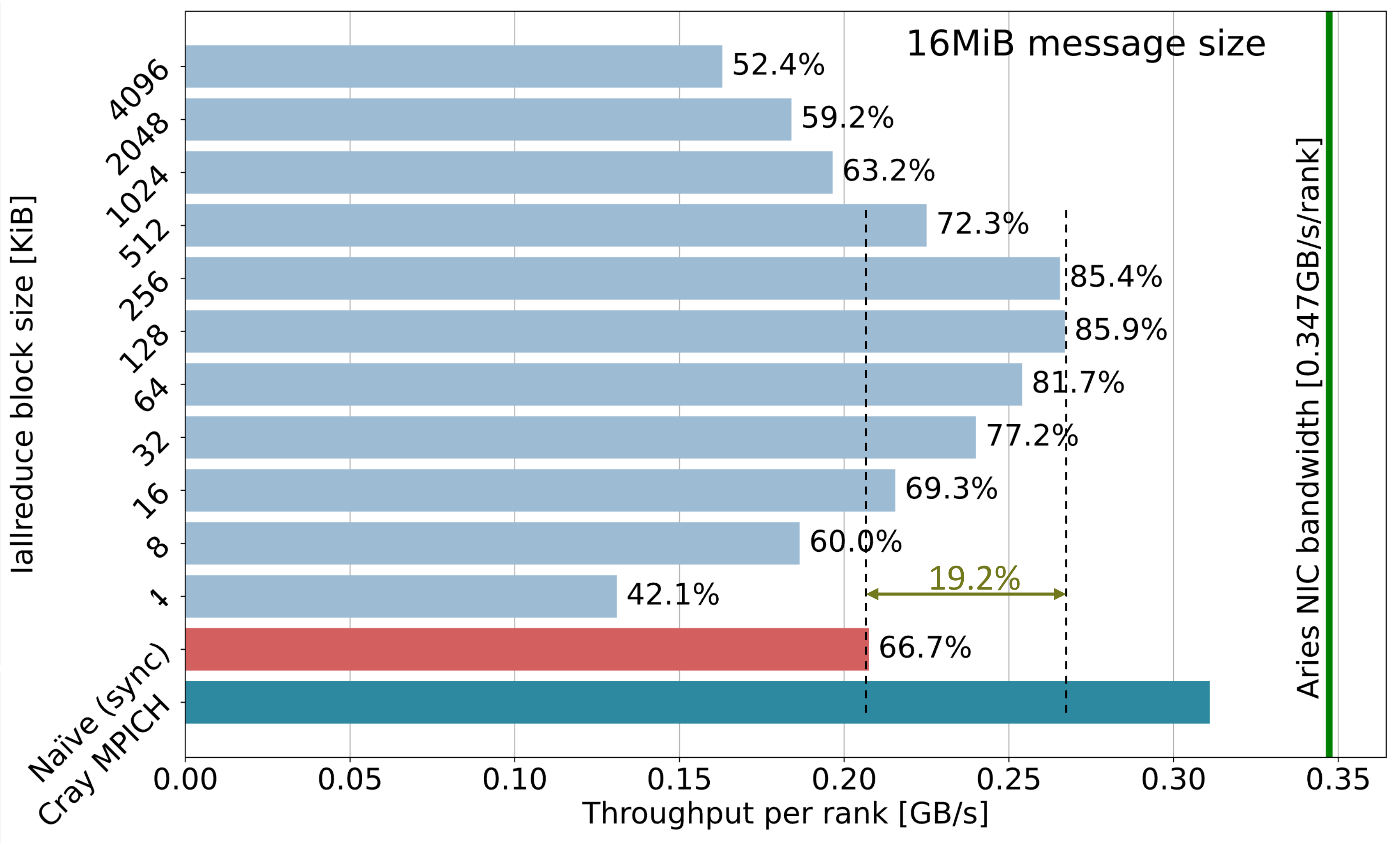


Overlap communication with computation

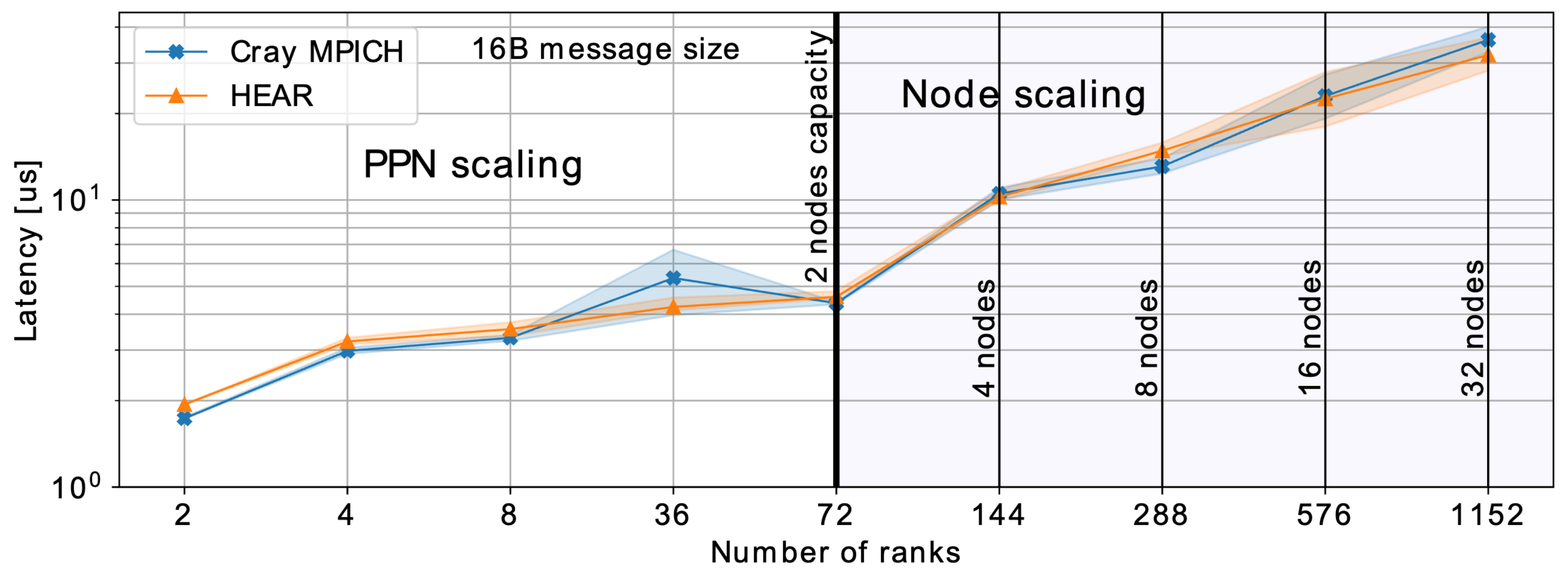


Memory pool avoids dynamic allocation using malloc and alleviates the cost of memory pinning for RDMA.

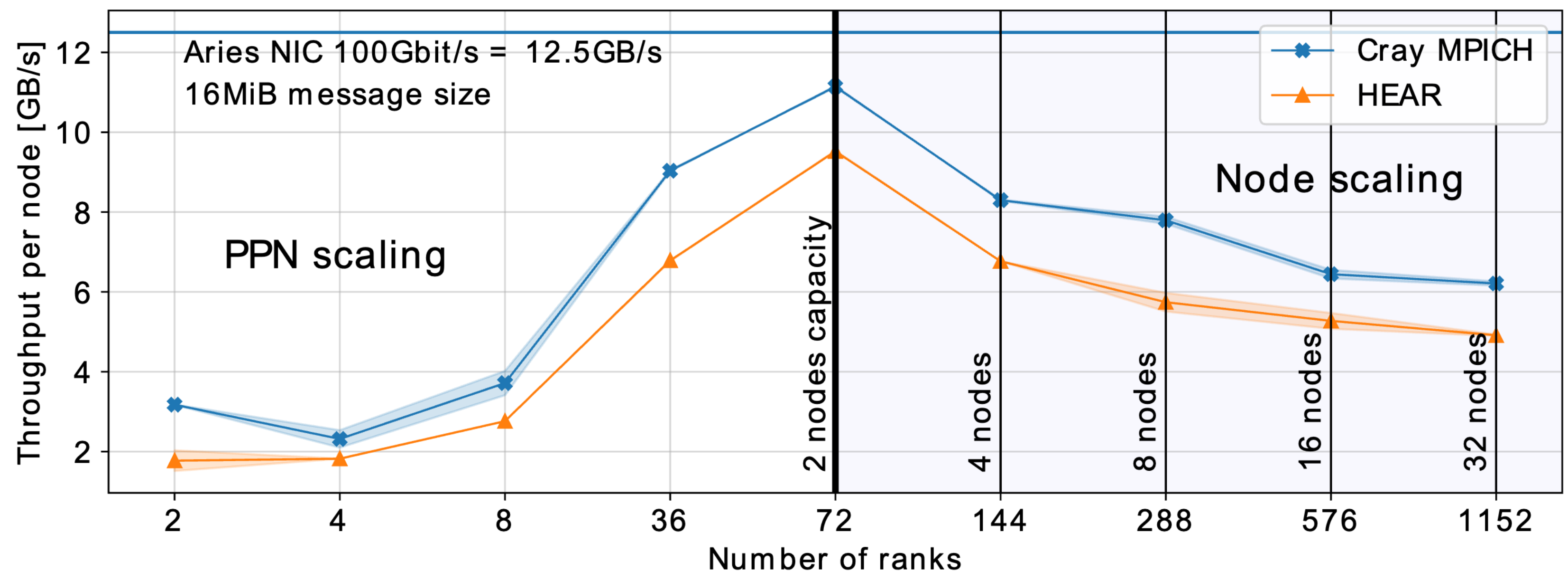
Optimal pipeline block size



Scalability



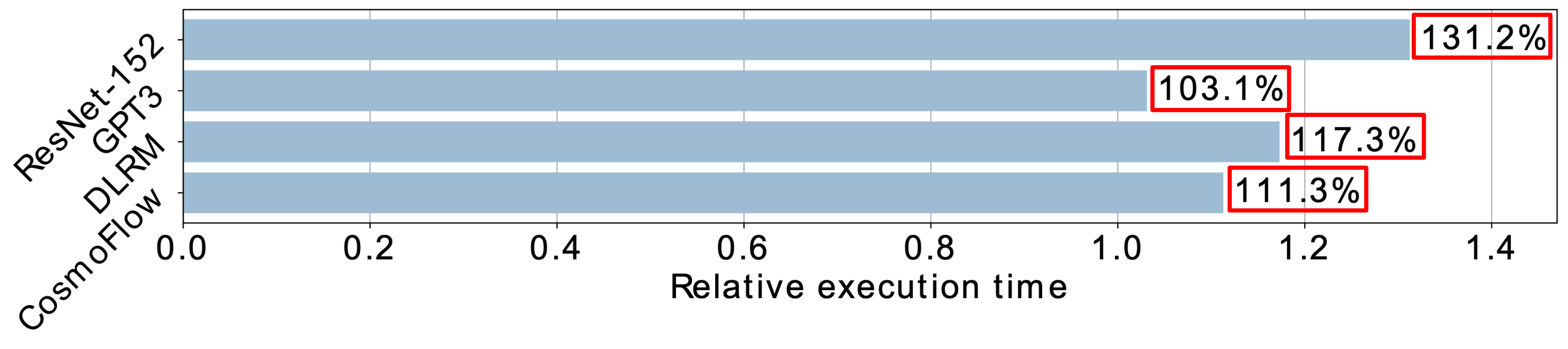
Consistently staying within ~10%



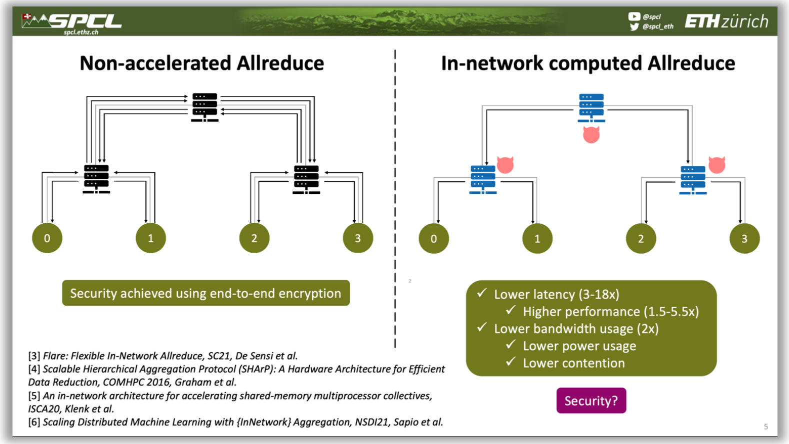
Consistently staying within ~25%

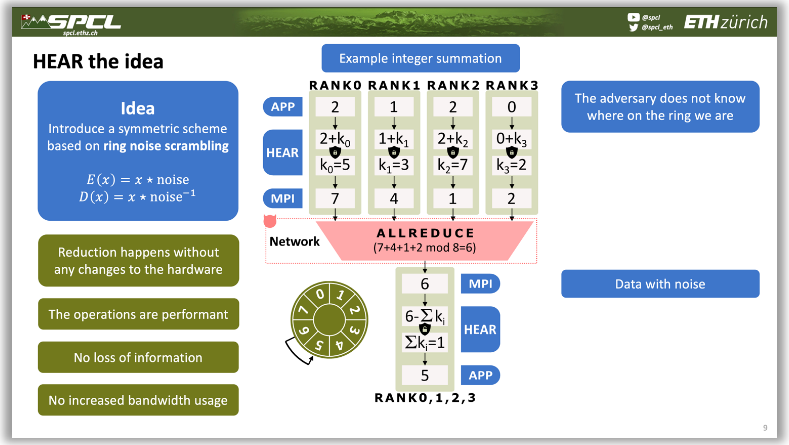
Applications

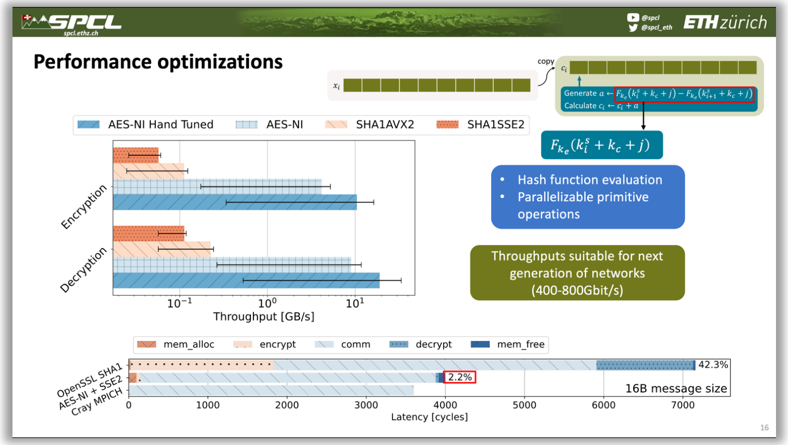
DNNs have the most challenging communication patterns.

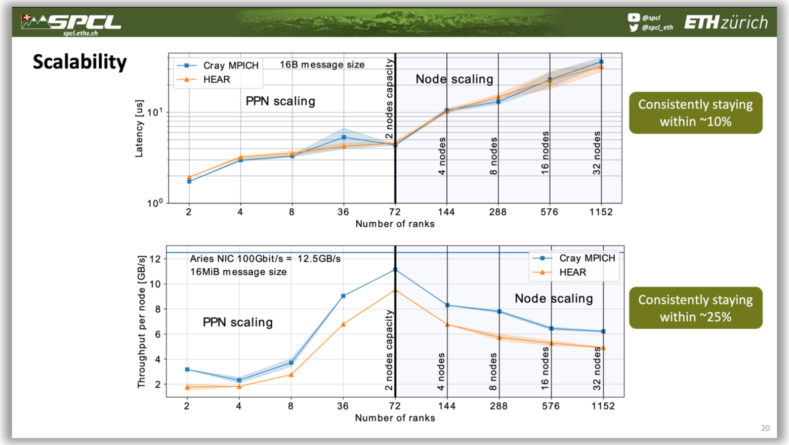


Conclusions









More of SPCL's research:

youtube.com/@spcl — 150+ Talks

twitter.com/spcl_eth — 1.2K+ Followers

github.com/spcl — 2K+ Stars

... or spcl.ethz.ch

https://github.com/spcl/libhear

Floating point operations

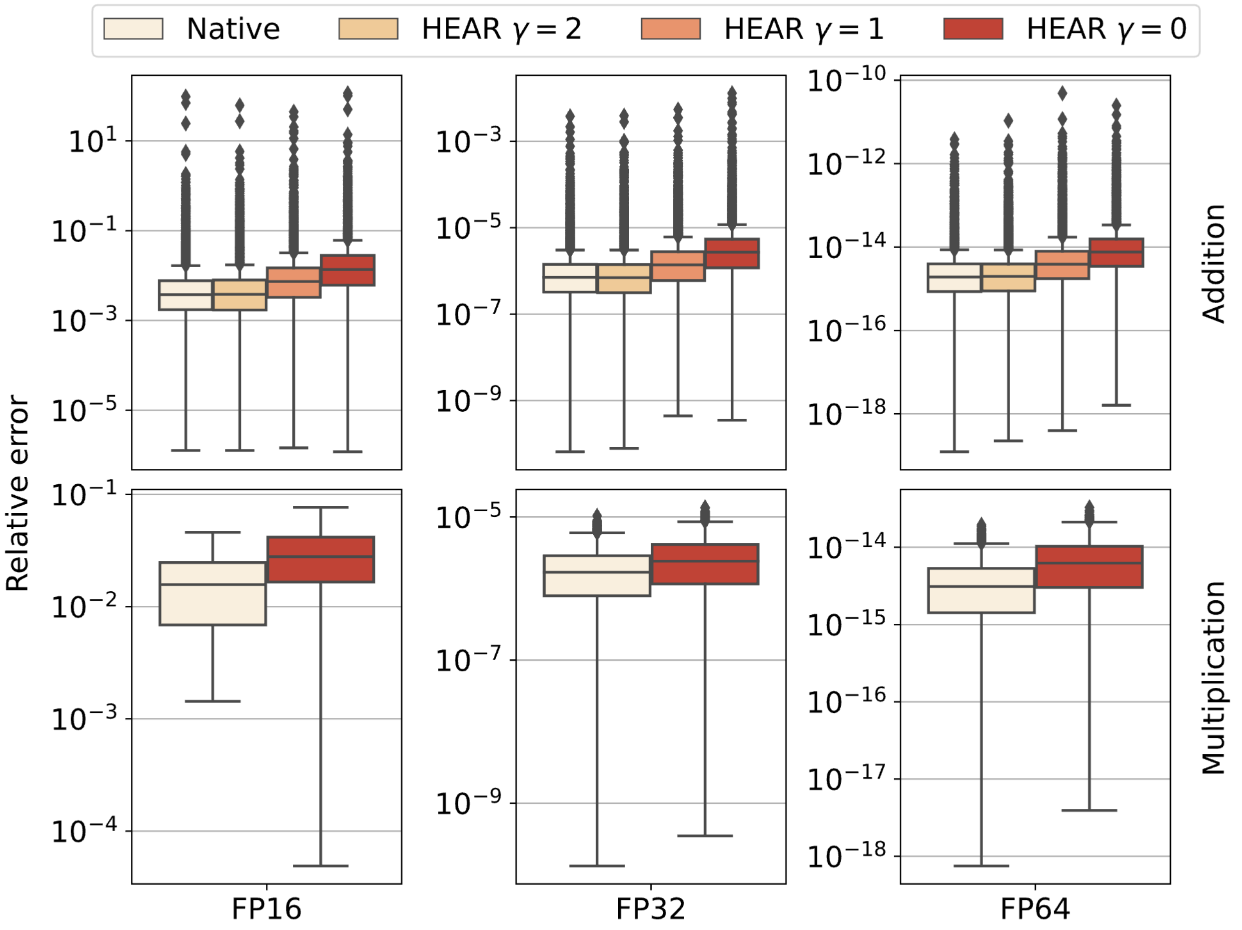
Assume noise f and the following format of a floating number:

$$x = (-1)^s \times m \times 2^e$$

$$c = x \otimes \mathrm{f} = (-1)^{s_x + s_f} \times (m_x \times m_f) \times 2^{e_x + e_f}$$

Create the ring of values on the exponent and introduce some noise to the mantissa via multiplication.

Average probability of a guess for FP32 is $3.57{\times}10^{-7}$ with reference probability of a guess equal to $2.38{\times}10^{-7}$ giving minor advantage to the attacker.



Security requirements

Trying to obtain keys

A set $\mathbb{S}$ with P processes storing a secure state

Untrusted network

The adversaries are stronger versions of passive attackers.

Each element is a separate plaintext we want to secure