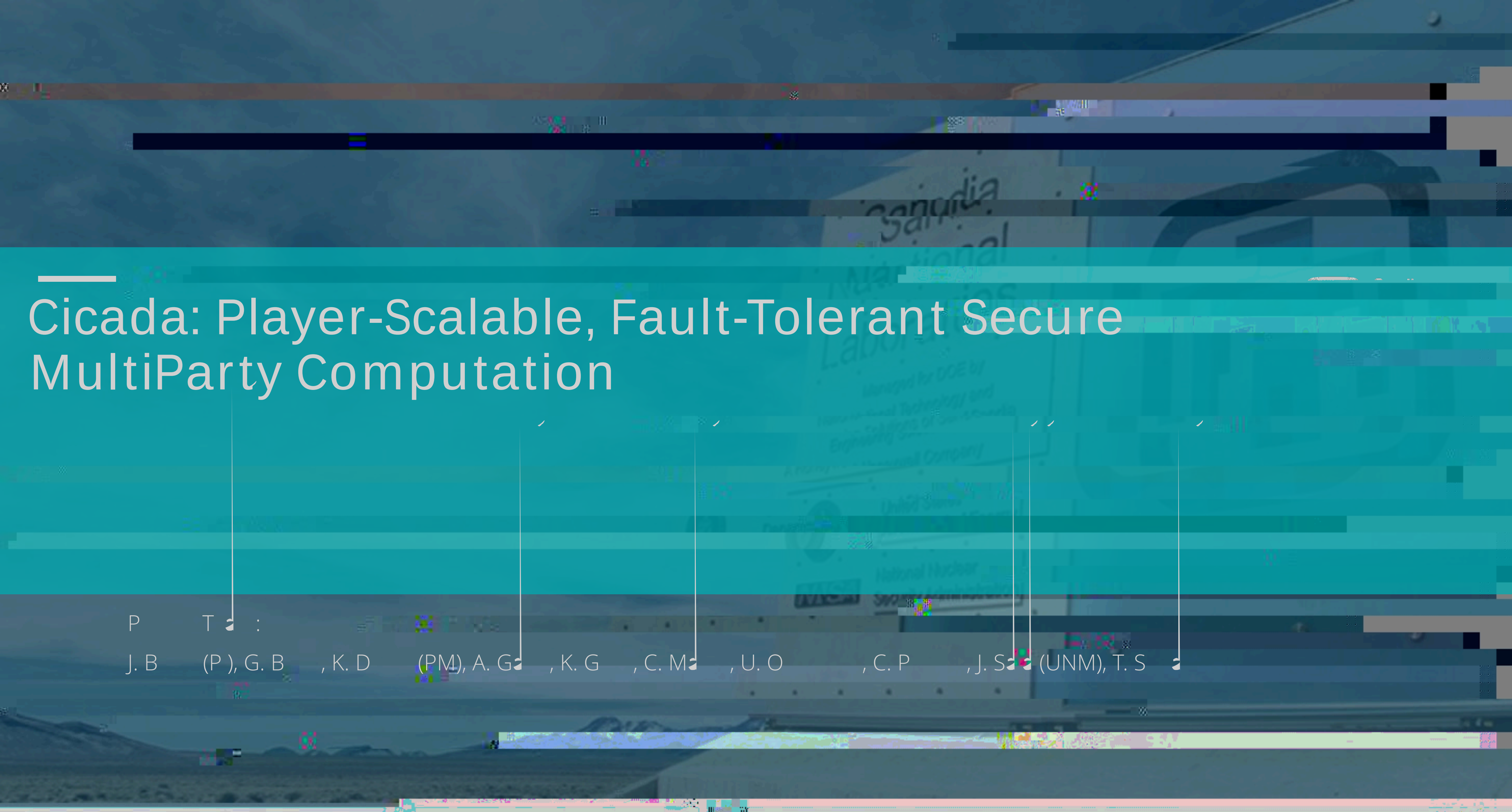
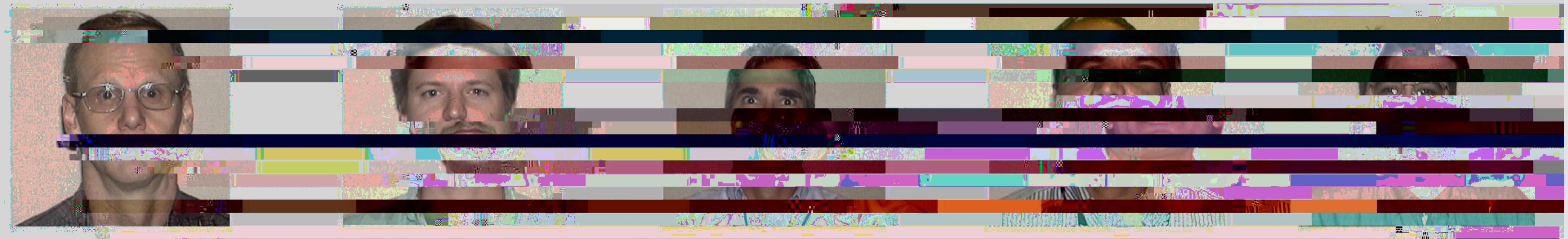


Cicada: Player-Scalable, Fault-Tolerant Secure MultiParty Computation

P T :

J. B (P), G. B , K. D (PM), A. G , K. G , C. M , U. O , C. P , J. S (UNM), T. S

Thanks to Our Multi-Disciplinary Research Team



Jon Berry (PI) Gabe Birch Kevin Dixon (PM) Anand Ganti Ken [illegible]

Carolyn Mayer Uzoma Onunkwo Cindy Phillips Jared Saia (UNM) Tim [illegible]

C -

Outline

- Application driver: Privacy-Preserving Machine Learning
- Algorithmic case study: dense matrix multiplication
- Software overview: Cicada-mpc (Fault-tolerant, open-source)

https://github.com/cicada-mpc/cicada-mpc/
https://cicada-mpc.readthedocs.io/

https://www.youtube.com/watch?v=GM_JuKrw4Ik

Motivation: MPC Linear Regression & Gradient Descent

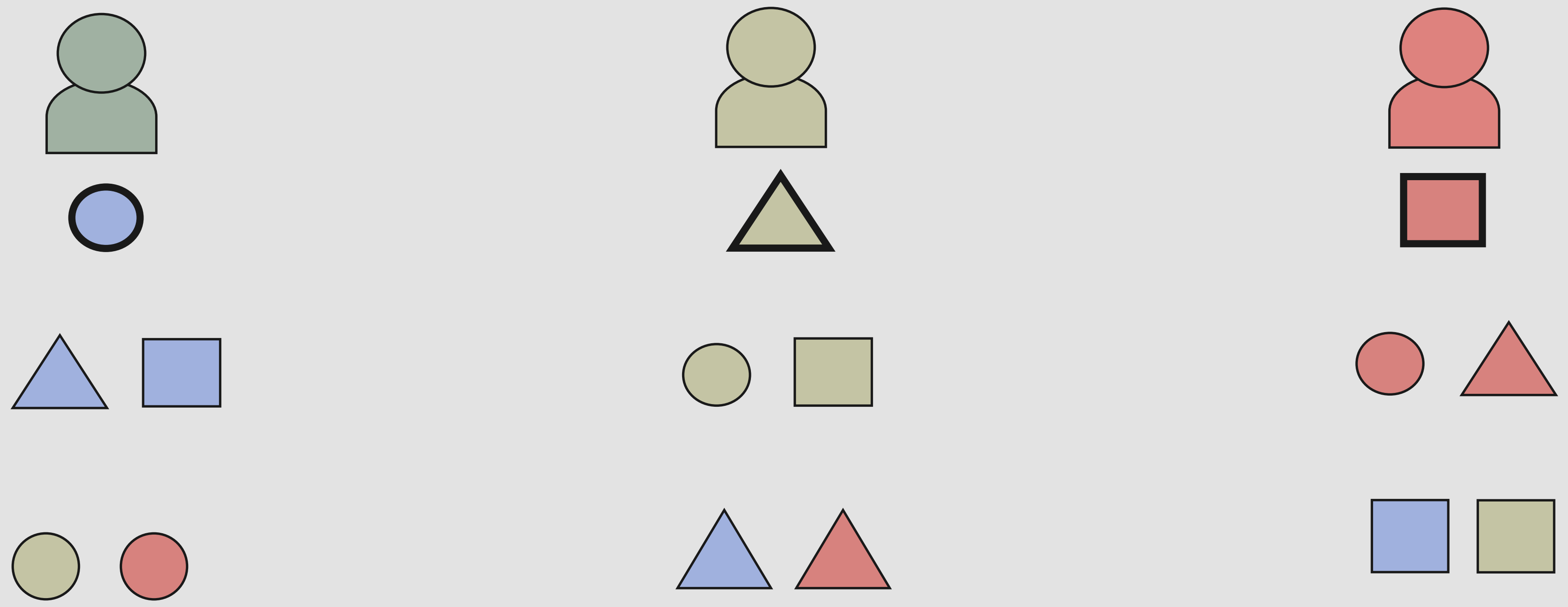
Gradient descent:

Model: vector $\boldsymbol{\beta}$.

Goal: Minimize a loss function

Local Gradient Matrices

Typical MPC Computation: Resharing Matrices

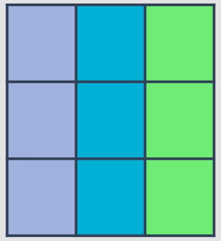


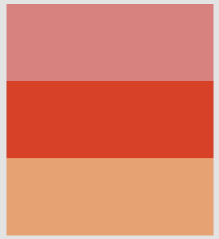
MMULT(A, B)

For each player :

1. $A_p{}' \leftarrow$ AGGREGATE$(A_p, \quad _p)$. # sum shares along columns
2. $\quad _p{}' \leftarrow$ AGGREGATE$(\quad _p, \quad p$

MMULT Example: 9 Players



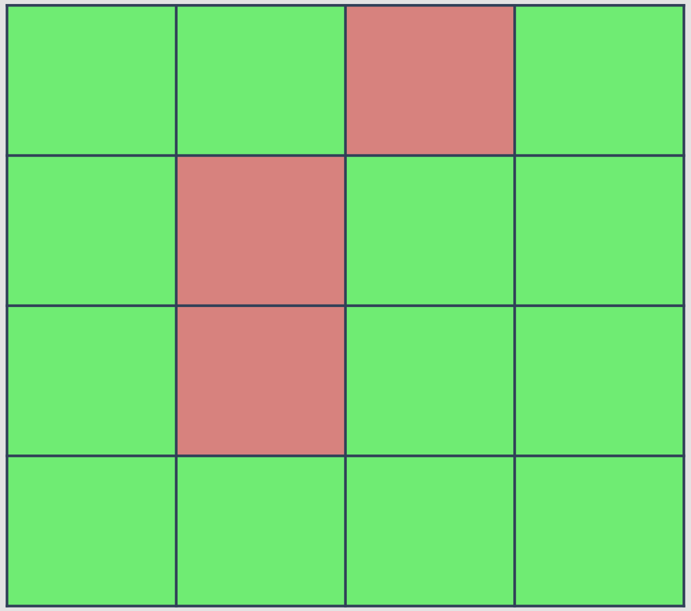


Global impact of MMULT:

Tolerating Fail-Stop Faults

Idea:

- Checkpoint row and column aggregated values.
- Use Cicada's built-in fault tolerance and Python exception handling



Based on three fundamental concepts

C ica

Network abstraction representing an unchanging group of players, and communication patterns to pass messages among them.

E c di g

Map between domain values and MPC-friendly integer field representations.

P c l S i e

Use communicators and encodings to implement curated collections of privacy-preserving protocols: secret sharing, addition, multiplication, logical comparison, etc.

Communication Patterns

Based on three fundamental concepts:

C ica

Network abstraction representing an unchanging group of players, and communication patterns to pass messages among them.

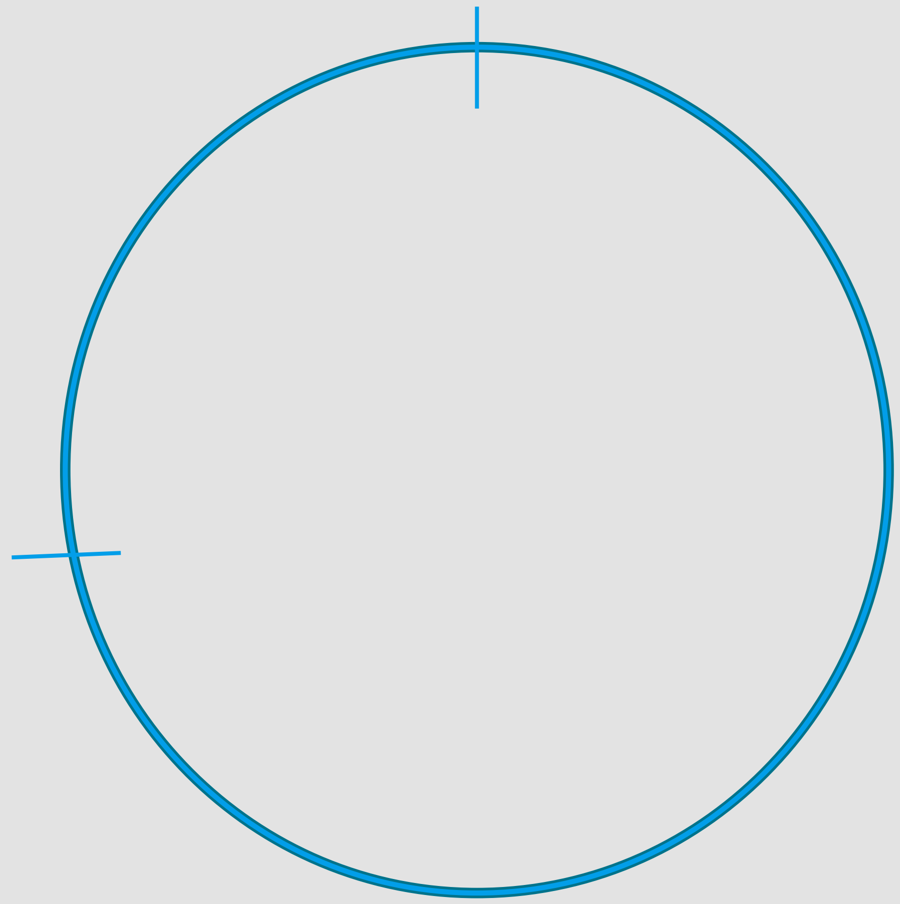
E c di g

Map between domain values and MPC-friendly integer field representations.

P c l S i e

Use communicators and encodings to implement curated collections of privacy-preserving protocols: secret sharing, addition, multiplication, logical comparison, etc.

Encoding Fixed Point Arithmetic into a Field



```
import numpy
```

Encodings

```
import numpy

from cicada.additive import AdditiveProtocolSuite
from cicada.communicator import SocketCommunicator
from cicada.encoding import Boolean
from cicada.interactive import secret_input

with SocketCommunicator.connect(startup_timeout=300) as communicator:
    protocol = AdditiveProtocolSuite(communicator)

    winner = None
    winning_share = protocol.share(src=0, secret=numpy.array(0), shape=())

    for rank in communicator.ranks:
        prompt = f"Player {communicator.rank} fortune: "
        fortune = secret_input(communicator=communicator, src=rank, prompt=prompt)
        fortune_share = protocol.share(src=rank, secret=fortune, shape=())
        less_share = protocol.less(fortune_share, winning_share)
        less = protocol.reveal(less_share, encoding=Boolean())
        if not less:
            winner = rank
            winning_share = fortune_share

    print(f"Winner: player {winner}")
```

Protocol Suites

```
import numpy

from cicada.additive import AdditiveProtocolSuite
from cicada.communicator import SocketCommunicator
from cicada.encoding import Boolean
from cicada.interactive import secret_input

with SocketCommunicator.connect(startup_timeout=300) as communicator:
    protocol = AdditiveProtocolSuite(communicator)

    winner = None
    winning_share = protocol.share(src=0, secret=numpy.array(0), shape=())

    for rank in communicator.ranks:
        prompt = f"Player {communicator.rank} fortune: "
        fortune = secret_input(communicator=communicator, src=rank, prompt=prompt)
        fortune_share = protocol.share(src=rank, secret=fortune, shape=())
        less_share = protocol.less(fortune_share, winning_share)
        less = protocol.reveal(less_share, encoding=Boolean())
        if not less:
            winner = rank
            winning_share = fortune_share

    print(f"Winner: player {winner}")
```

```
hostA $ cicada start --rank 0 millionaires.py

Player 0 fortune: 1230000
INFO:root:Winner: player 1
```

```
hostB $ cicada start --rank 1 millionaires.py

Player 1 fortune: 4560000
INFO:root:Winner: player 1
```

```
hostC $ cicada start --rank 2 millionaires.py

Player 2 fortune: 3400000
INFO:root:Winner: player 1
```

Fault Tolerance

All communication patterns have explicit, finite timeouts ...

… so failures cannot go unnoticed.

Communicators raise exceptions when failures occur …

… this is the part where other MPC tools just die.

```
7 features passed, 0 failed, 1 skipped
```

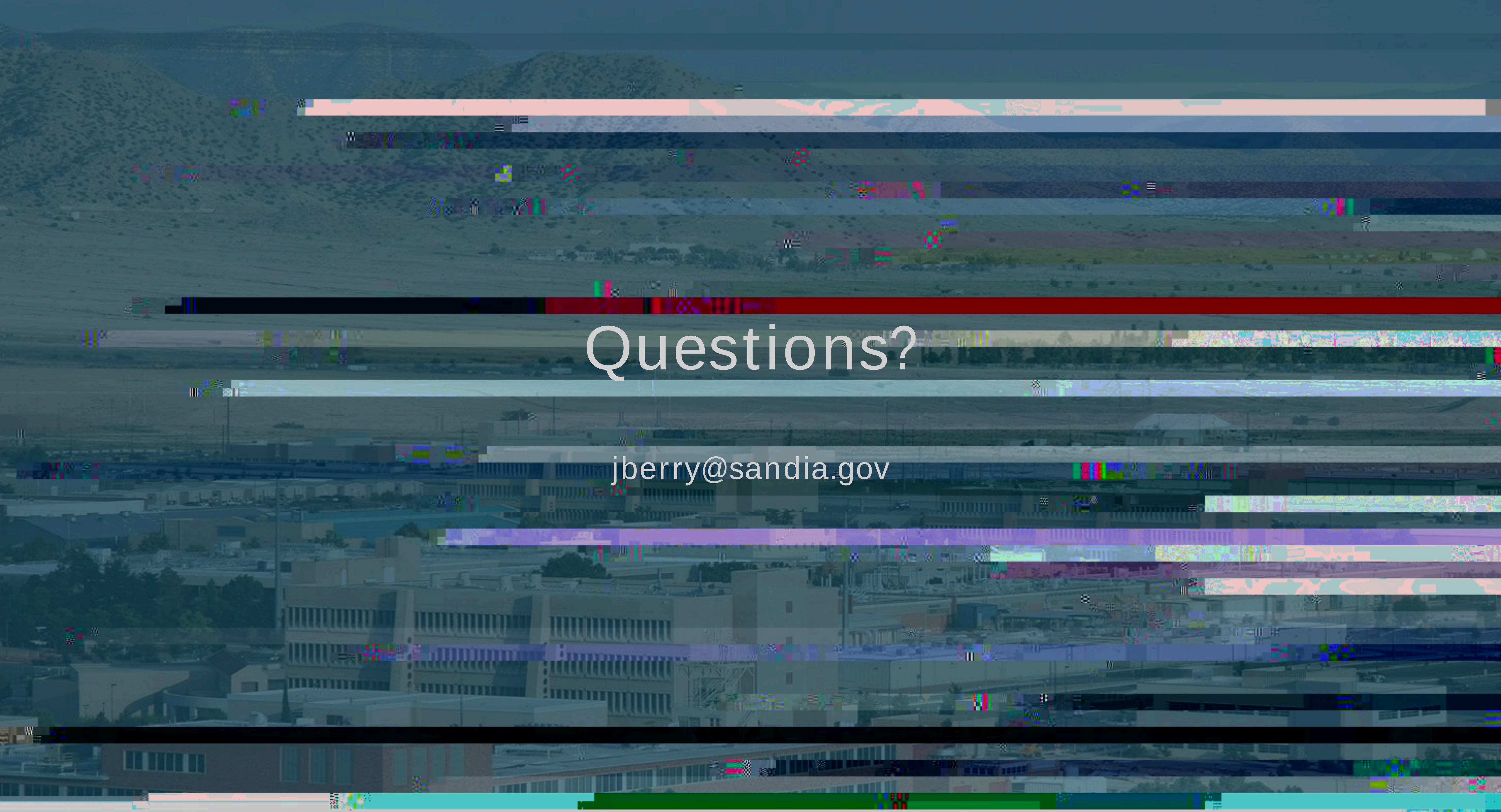
MPC Through 100 Players!

Conclusions, HPC Community Asks

- A A ( )?

! "#$$%&$'#(%')*+% , $%$-'./'0$1%2345 %#$6$#$78$%9 : ;.$: $70'09*7(%97%5<=>?%'71%
@;$75<=A %B$%91$70969$1%;#*C.$: (%(/8"%'(D

- >* : : /798'0*#%#$-*8'09*7% , '(7E0%1$0$80$1%C&%'..%#'7F(+%1$;$7197)%*7% , "98"% #'7F(%97909'0$1%0"$%#$-*8'09*7A
- G* : $%8*..$809-$%*;$#'09*7(%191%7*0%#'9($%09 : $*/0%$##*#(%$-$7% , "$7%(* : $% #'7F(% , $#$%1$'1A
- H$8'/($%2345 %"'(7E0%C$$7%'1*;0$1%C&%5<=+%0"$%<&0"*7% : ;9I;&%C97197)(% 1*7E0%(/;;*#0%2345 +%'71% , *#F97)% , 90"%;'08"$1%C97197)(%($-$#$.&%.9 : 90(%*/#% 'C9.90&%0*%19(0#9C/0$%*/#%(*60 , '#$A



Questions?

jberry@sandia.gov