SAND2015-4738C

Sandia National Laboratories

Exceptional service in the national interest

Evolving the message-passing model via an object-oriented, fault-tolerant transport layer

Jeremiah Wilke, Janine Bennett, Hemanth Kolla, Nicole Slattengren, Keita Teranishi, David Hollman

FTXS Workshop at HPDC
Portland, OR
June 15, 2015

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Applications should focus on algorithm-specific fault-tolerance problems

Sandia National Laboratories

- Fail-stop fault tolerance has a ``universal'' solution, once checkpoint interface is defined
  - LFLR (local-failure, local-recovery; Sandia)
  - FMI (fault-management interface; LLNL)
  - Fenix (Rutgers)
- Should silent data corruption be focus of algorithm-specific approaches?
- Two choices (either may actually be end being correct)
  - Ad hoc solutions engineered for specific problems
  - General solutions that are broadly applicable
- Fault-tolerance and performance are a *programming productivity* problem

Goals of the talk/position paper

Sandia National Laboratories

1) Sit down as quickly as possible so smarter people can give feedback/criticism

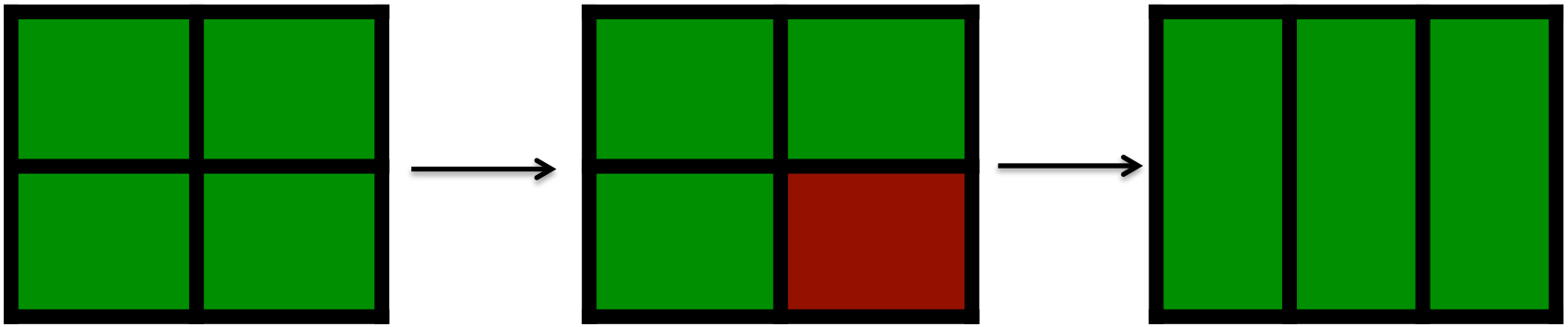
2) Complimentary to, not critical of ULFM

3) Engage current state of the art to avoid repeating efforts

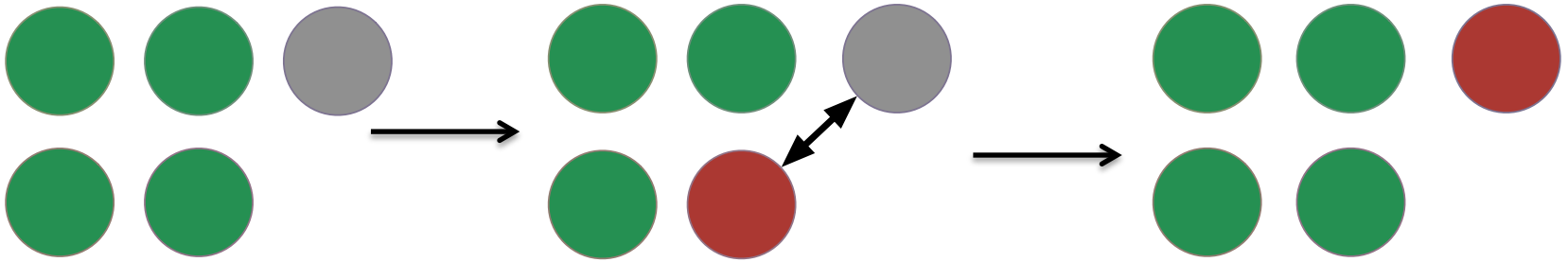
Shrinking model is difficult, app-specific
Non-shrinking model is universal, runtime-level

Sandia National Laboratories

Shrinking model potentially requires complete repartitioning



Non-shrinking model transparent – might be network topology consequences



MPI return codes create messy logic if MPI functions can``fail''

Sandia National Laboratories

```
int rc = MPI_Send(...);
if (rc == FAILED){
  //what to do here
}

rc = MPI_Recv(...);
if (rc == FAILED){
  //what to do here
}

rc = MPI_Wait(...);
if (rc == FAILED){
  //what to do here
}
```

If statements are bad!

Our position is that we want to know other people's positions

Sandia National Laboratories

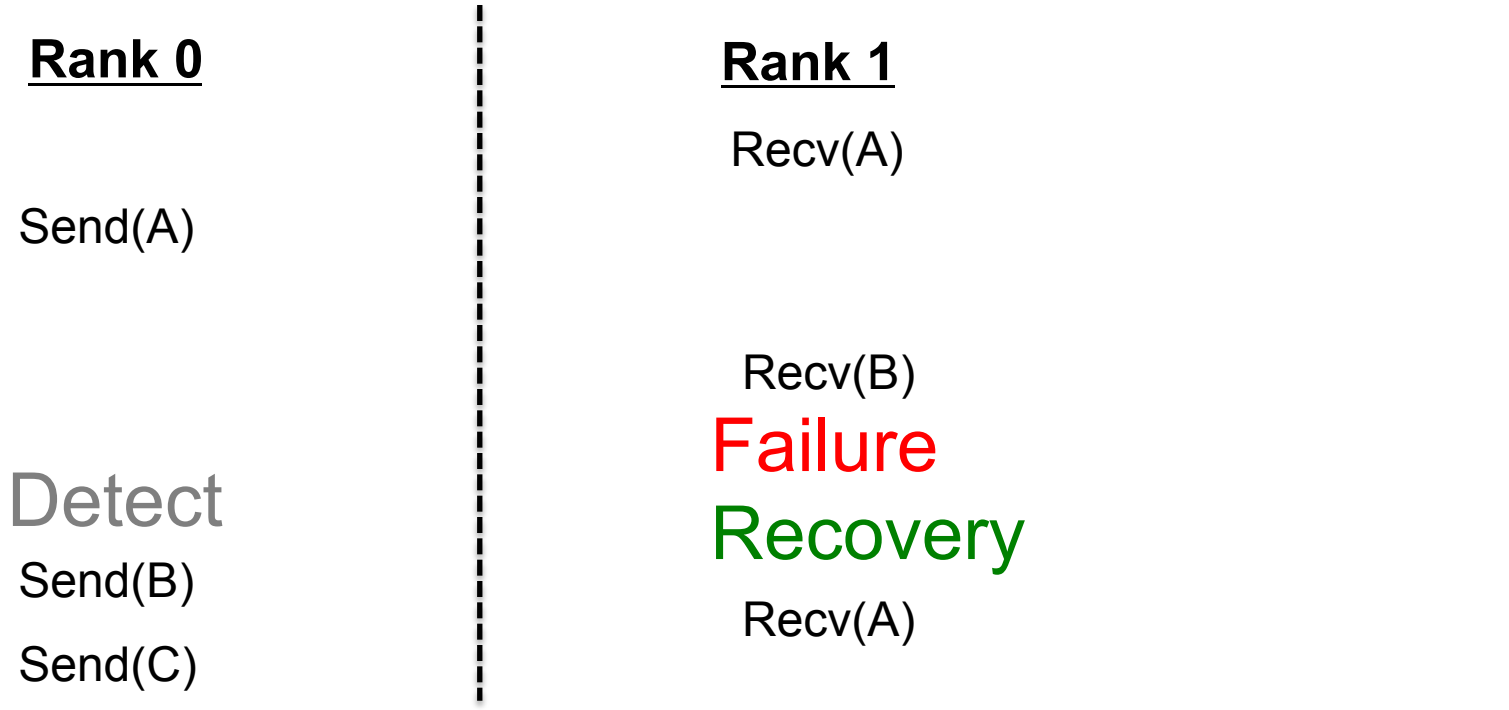
1. Our group at Sandia has been looking at fault-tolerance with task-based models
   -Fault-tolerance *seems* well-defined and straightforward in systems like TASCEL
2. Interactions with LFLR, Fenix projects
   -Fault-tolerance *seems* well-defined and "straightforward" for non-shrinking MPI model
3. Can we combine some basic aspects of many-task models into message passing (communicating sequential processes) to provide a performant and general-purpose fault-tolerance tool?

Local recovery can't ever really be strictly local

Sandia National Laboratories

Rank 0	**Rank 1**
	Recv(A)
Send(A)	
	Recv(B)
	Failure
Detect	Recovery
Send(B)	
Send(C)	Recv(A)

Local recovery can't ever really be strictly local



How does Rank 1 get Message A to be resent?

1) Rank 0 rolls back
2) Rank 0 detects error and resends all messages from a log
3) Rank 0 carries on, Rank 1 requests messages as needed

Local recovery can't ever really be strictly local

Sandia National Laboratories

Rank 0	Rank 1
	Recv(A)
Send(A)	
	Recv(B)
	Failure
Detect	Recover
Send(B)	
	Recv(A)
Send(C)	

How does Rank 1 get Message A to be resent?

1) Rank 0 rolls back
2) Rank 0 detects error and resends all messages from a log
3) Rank 0 carries on, Rank 1 requests messages as needed

Question for the audience: Did I miss any options?

Matrix of choices and tradeoffs for "transparent" fault-tolerance in MPI

Sandia National Laboratories

	Explicitly Coordinated Checkpoint Restart	Implicitly Coordinated Checkpoint Restart	Fully Uncoordinated Checkpoint Restart
All Processes Rollback	No special bookkeeping, Eager protocols preserved	Synchronous rollback	Not viable, Domino effect
Only Failed Process Rolls Back	Message logging, Rendezvous protocol required	Message logging, Garbage collection	Message logging, More complicated garbage collection

Matrix of choices and tradeoffs for "transparent" fault-tolerance in MPI

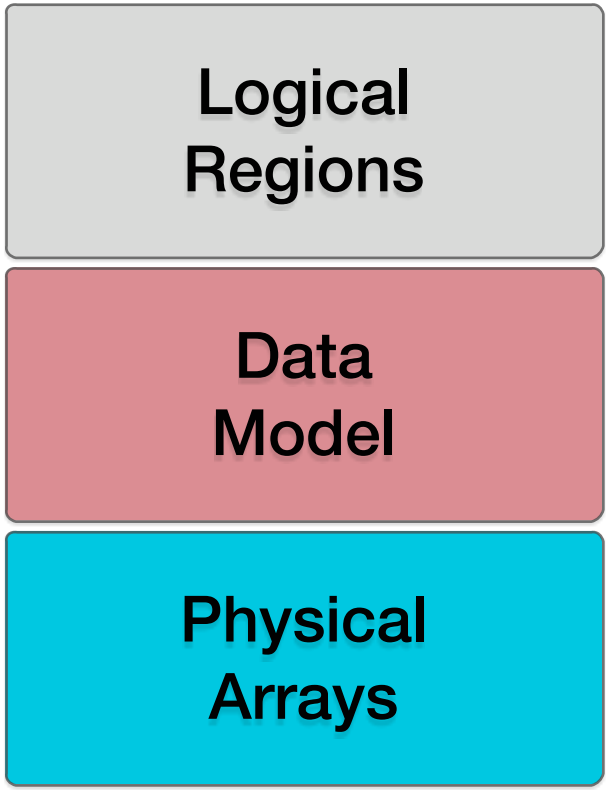
Sandia National Laboratories

	Explicitly Coordinated Checkpoint Restart	Implicitly Coordinated Checkpoint Restart	Fully Uncoordinated Checkpoint Restart
All			ct
Only Failed Process Rolls Back	Message logging, Rendezvous protocol required	Message logging, Garbage collection	Message logging, More complicated garbage collection

Is there a way to enhance recovery strategies when only the failed process rolls back and restarts?

Every application has logical regions, a data model, and physical mapping

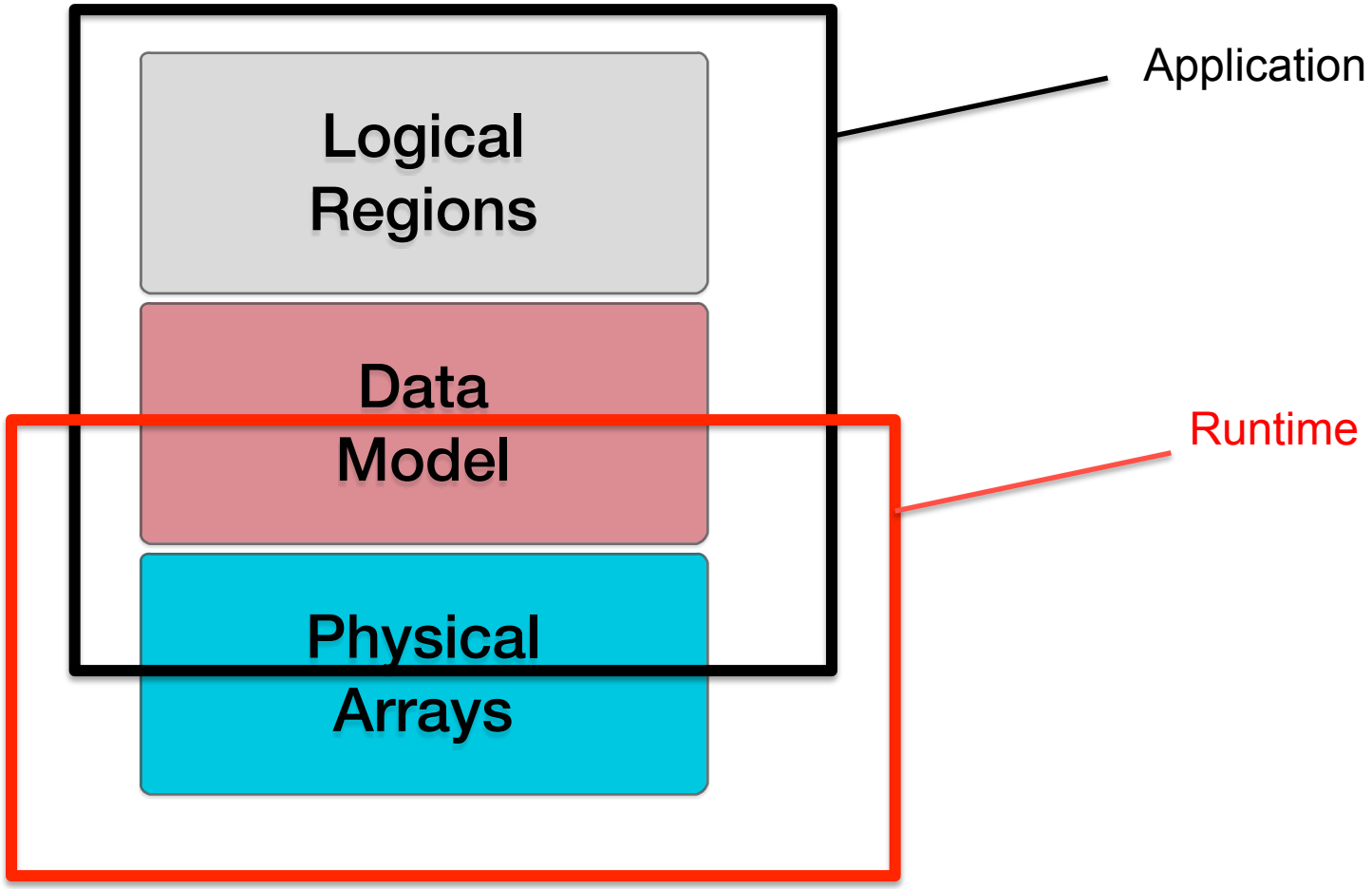
Sandia National Laboratories



Every application has logical regions, a data model, and physical mapping

Sandia National Laboratories

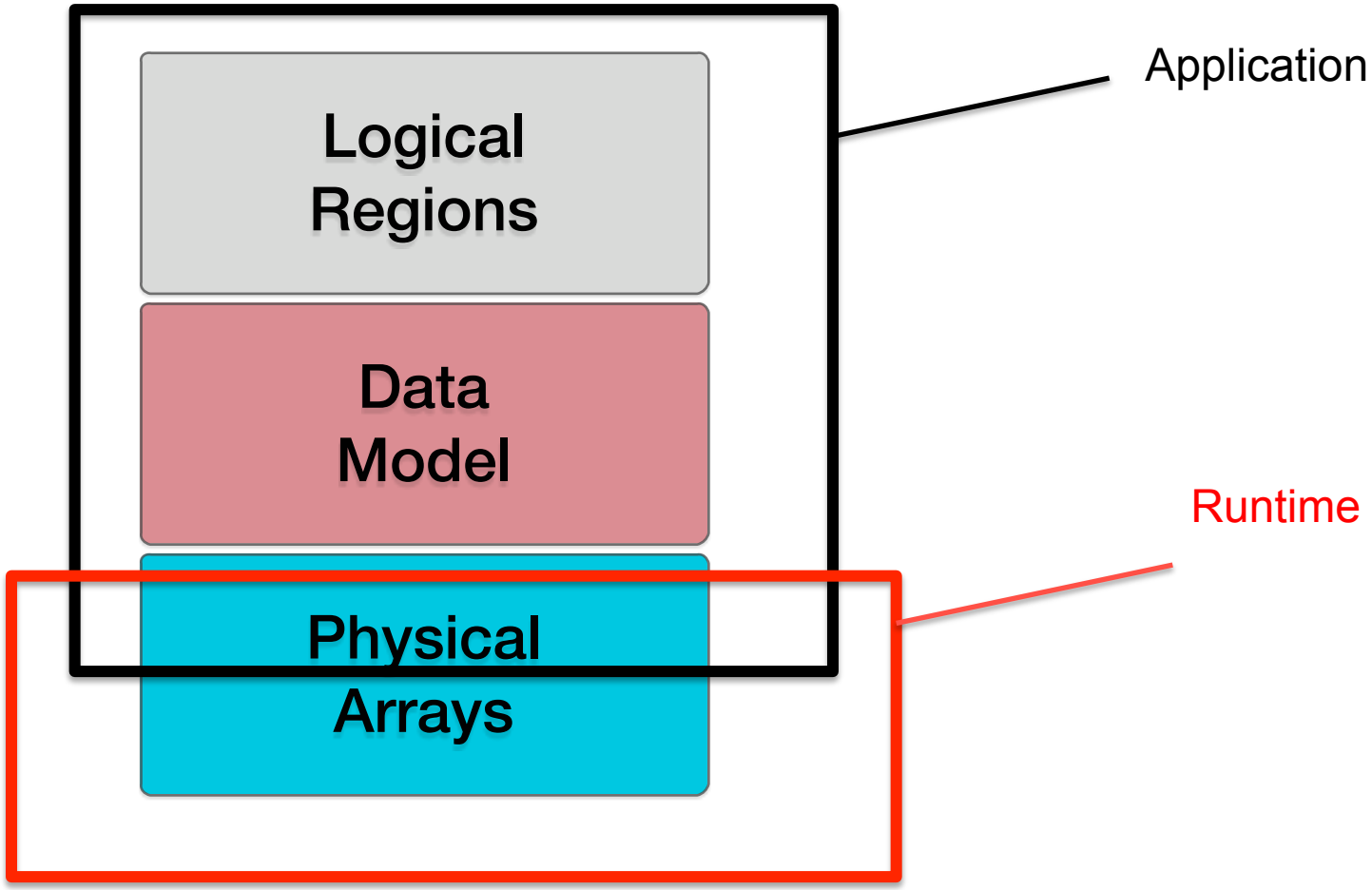
MPI: What apps and runtime interact with



Every application has logical regions, a data model, and physical mapping

Sandia National Laboratories

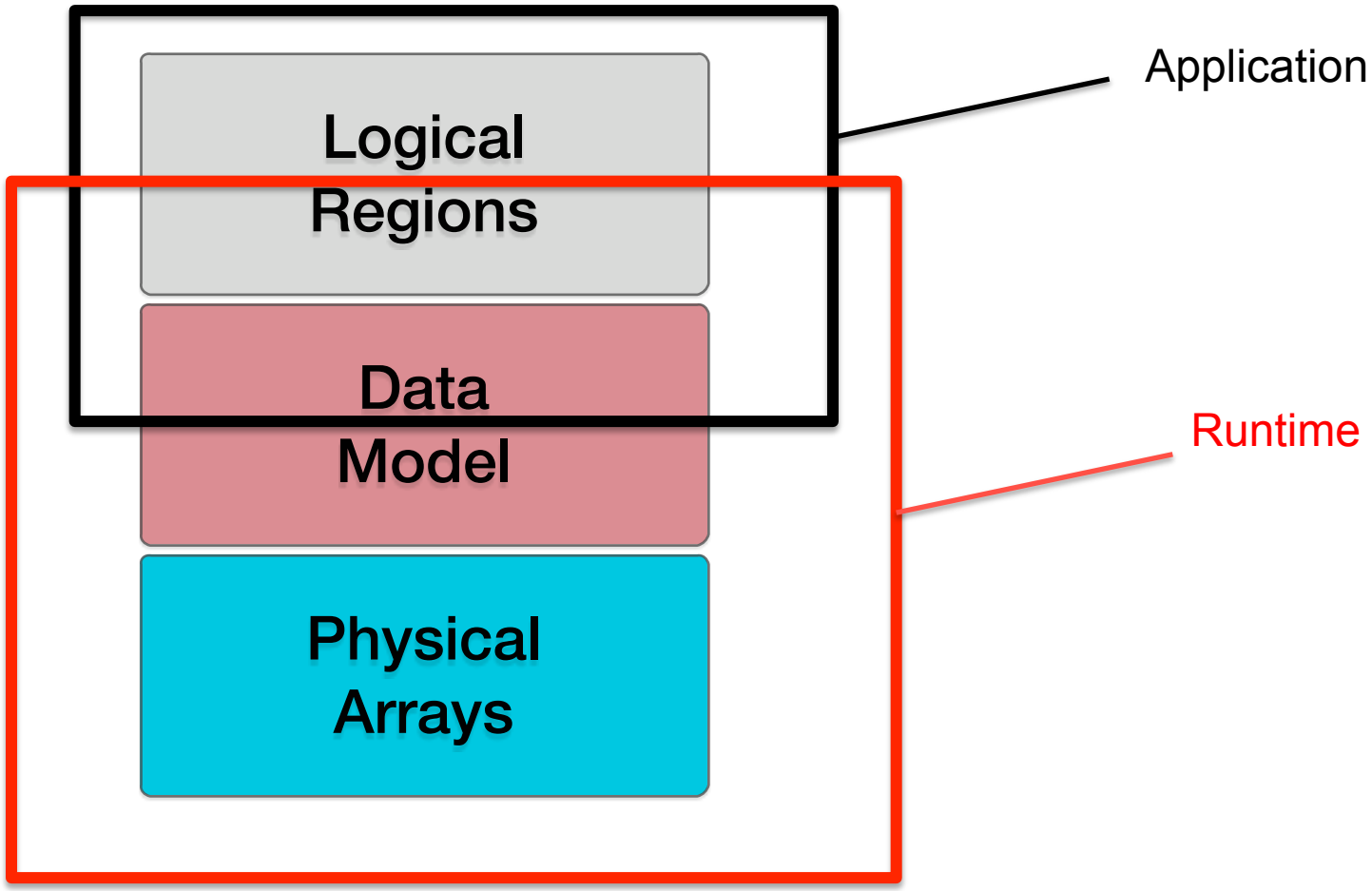
PGAS: What apps and runtime interact with



Every application has logical regions, a data model, and physical mapping

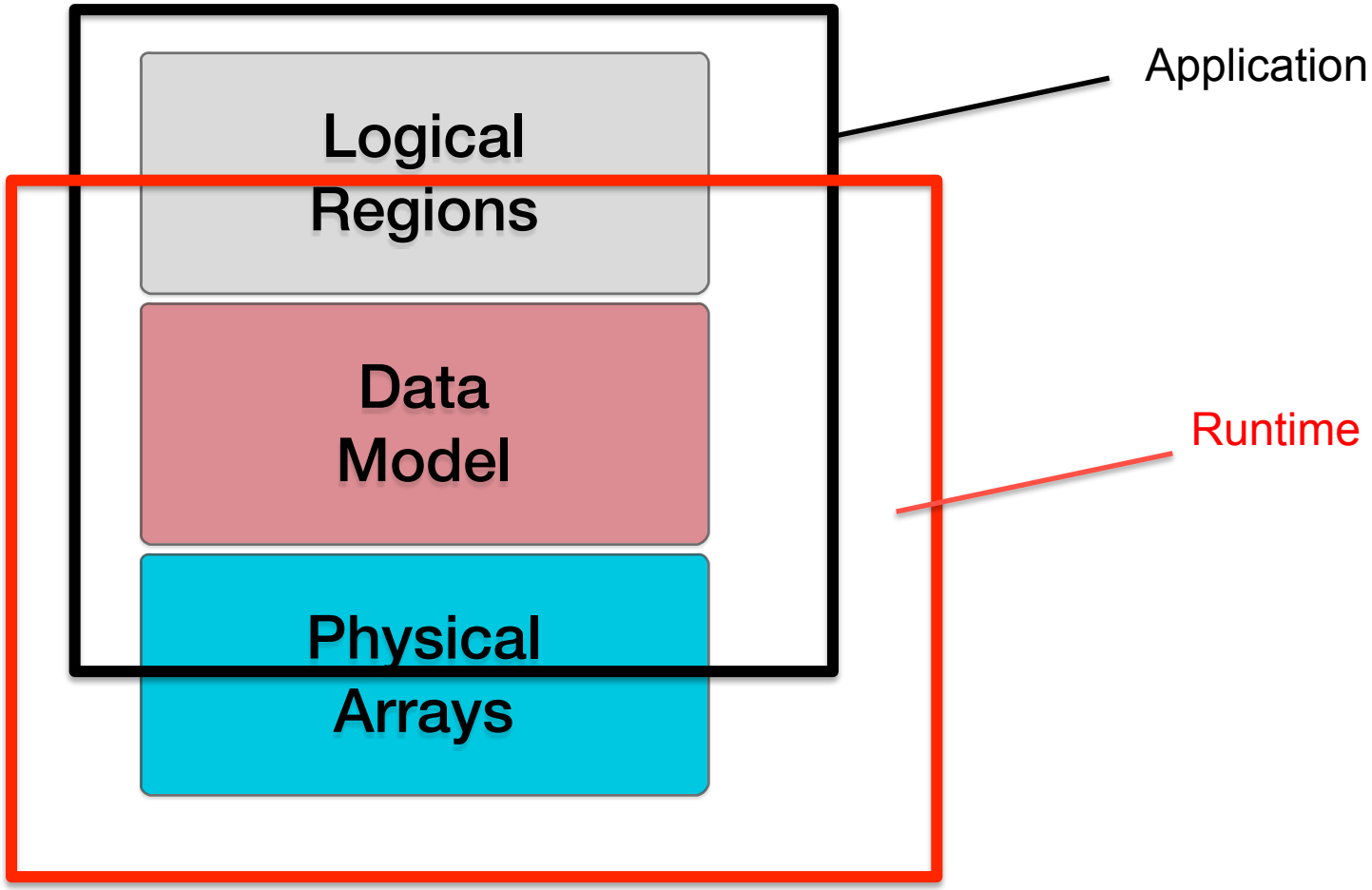
Sandia National Laboratories

(Some) many-task models



Every application has logical regions, a data model, and physical mapping

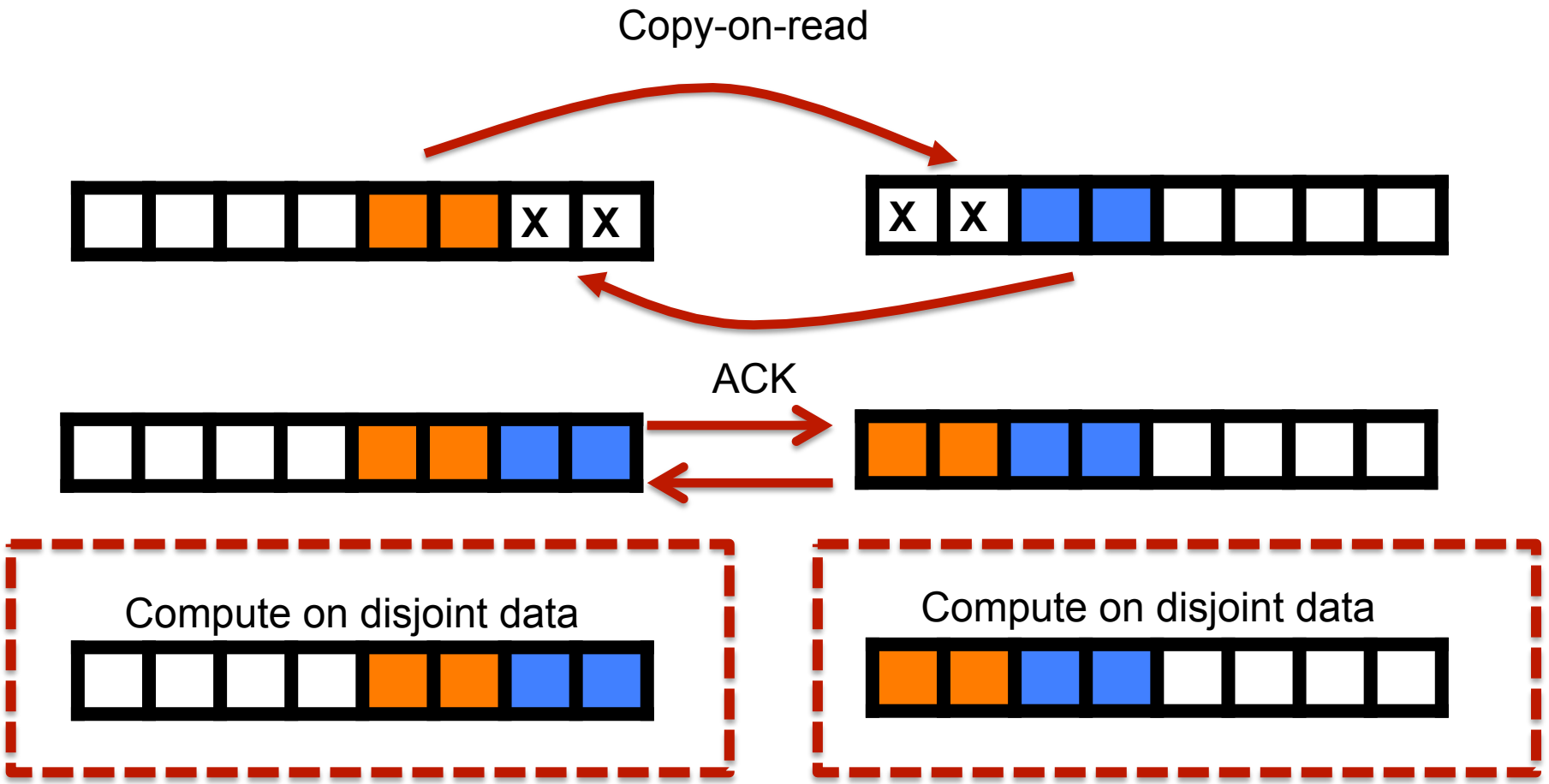
MPI: Is there any value in having the runtime know logical identity of data?



Sandia National Laboratories

Originally proposed publish/subscribe functions... let's start with MPI tags...

Message passing (CSPs) means (usually) two-sided, private address spaces, copy-on-read



Every message in MPI is ``logically'' identified Swap message order for logical identifiers

Sandia National Laboratories

- MPI message matching is "transparent" to the application, based on in-order message delivery

```
MPI_Send(buffer, count, type, dest, tag, comm);
   ->implicit order number
Tuple<dest,tag,comm,order> -> unique identifier
```

- MPIX + key-value extensions would logically identify all data sent with unique tag, checkpoint as you go

```
MPIX_KV_Tag tag(matrixBlock, 0, 0);
MPI_Send(buffer, count, type, dest, tag, comm);
```

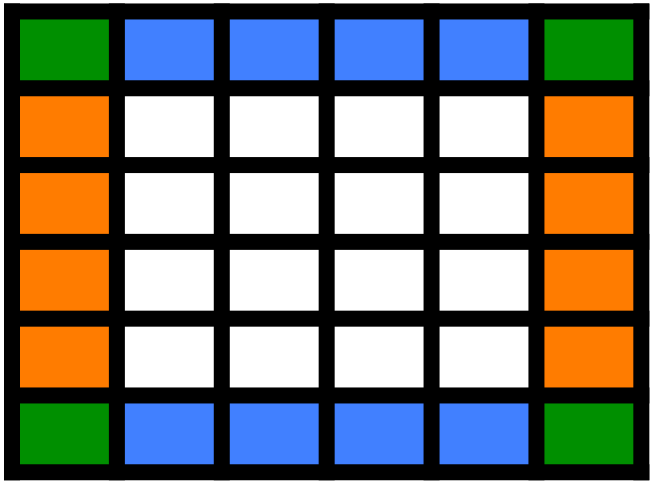
- Register buffer with checkpoint beforehand

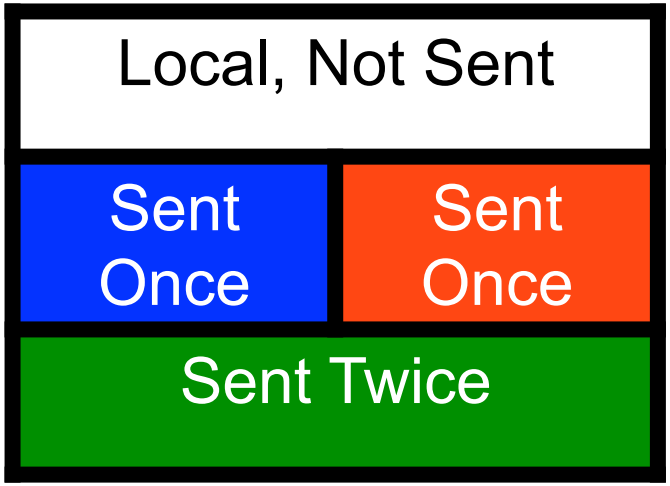
```
MPIX_KV_Tag tag(mesh, 0, 0, 0);
MPIX_Checkpoint(tag, buffer);
MPI_Datatype subsetType = …;
MPIX_Send(buffer, count, subsetType, dest, subtag, comm);
```

Logical tags are a general-purpose, application-specific solution

Sandia National Laboratories

Ghost exchange application
Optimum checkpoint interval = 10 iterations
Small stencil = N(ghost) << N(local)



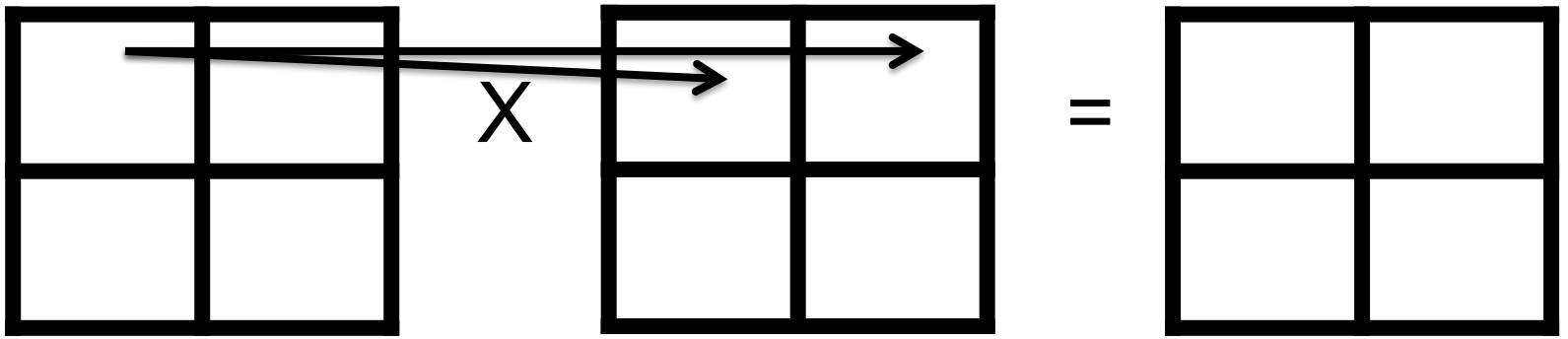


Logical data model provides little benefit beyond pessimistic message logging

Logical tags are a general-purpose, application-specific solution

Sandia National Laboratories

Tensors dominant part of electronic structure codes
Matrix-multiplication might send the same block many times

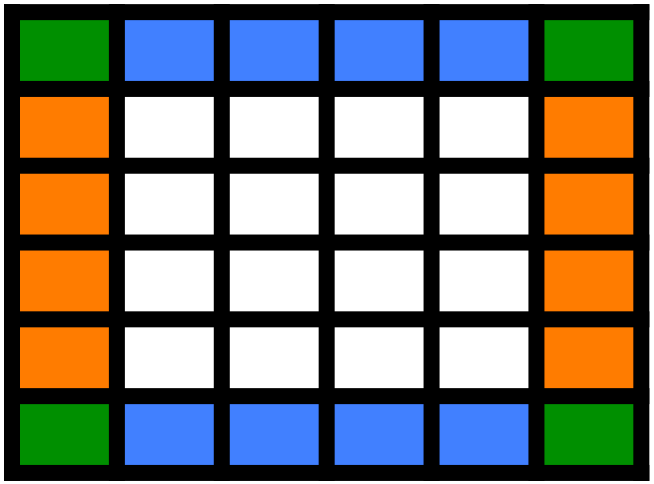


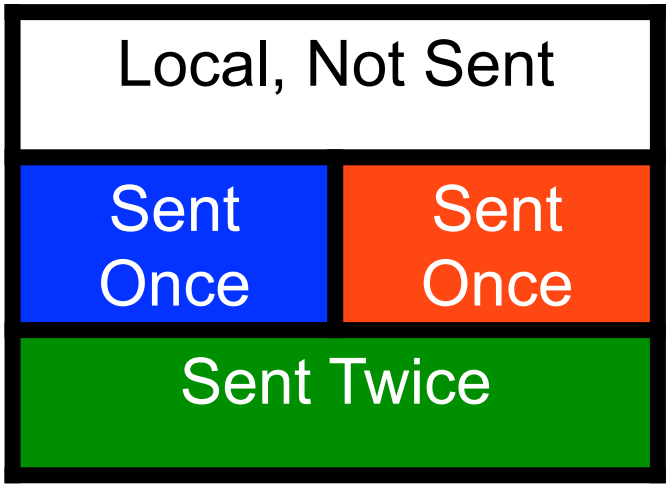
Logical data model cheaper than message logging when same data is sent multiple times

Logical tags are a general-purpose, application-specific solution

Sandia National Laboratories

Ghost exchange application
Previous iterations are kept for later analysis





Logical data model allows framework to avoid keeping message logs unnecessarily

Commercial break

Sandia National Laboratories

Failure Masking and Local Recovery for Stencil-based Applications at Extreme Scales

Marc Gamell (Rutgers University)

10:35 AM HPDC

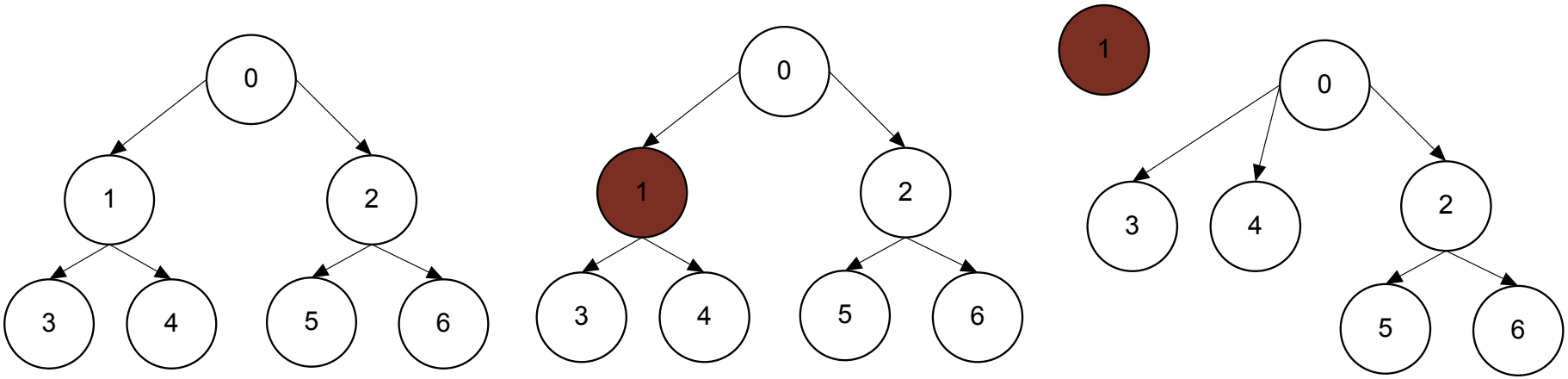
Simple object-oriented transport layer aims to expose underlying protocols, direct control of messages, for rapid prototyping

Sandia National Laboratories

```
class message : public ptr {
  typedef intrusive_ptr<message> ptr;
};

/** “Direct” functions – physical actions */
void smsg_send(message::ptr, int dst, …);
void rdma_put(message::ptr, int dst, …);
void rdma_get(message::ptr, int dst, …);

/** ``Indirect” function, runtime chooses appropriately */
void send(message::ptr msg, int dst);

/** Non-blocking collectives */
allreduce(…); -> returns collective_message::ptr to poll() function

/** Polling functions */
message::ptr blocking_poll();
message::ptr nonblocking_poll();
```

Log-scaling agreement algorithm

Everyone returns with same set of failed processes

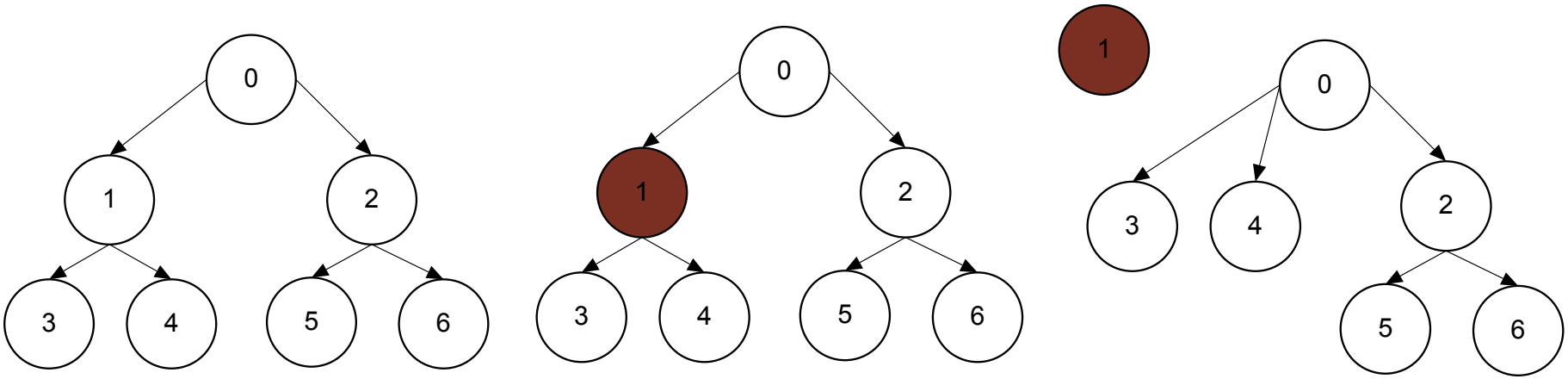



```
class VoteFunctor {
  operator()(void* newData);
}

class collective_message {
  std::set<int> failedProcs;
  …
}
```

```
void vote(VoteFunctor* fxn);

message::ptr msg = blocking_poll();
If (msg->cls() == collective){
  handleCollective(msg);
}
```

Log-scaling agreement algorithm

Simulated failures – RDMA get (ping) returns enum
API for “failing” nodes




```
class VoteFunctor {
  operator()(void* newData);
}

class collective_message {
  std::set<int> failedProcs;
  …
}
```

```
void vote(VoteFunctor* fxn);

message::ptr msg = blocking_poll();
If (msg->cls() == collective){
  handleCollective(msg);
}
```

Transport layer aims to provide well-defined, simple semantics for reliable message delivery

Sandia National Laboratories

send(M{type=payload})

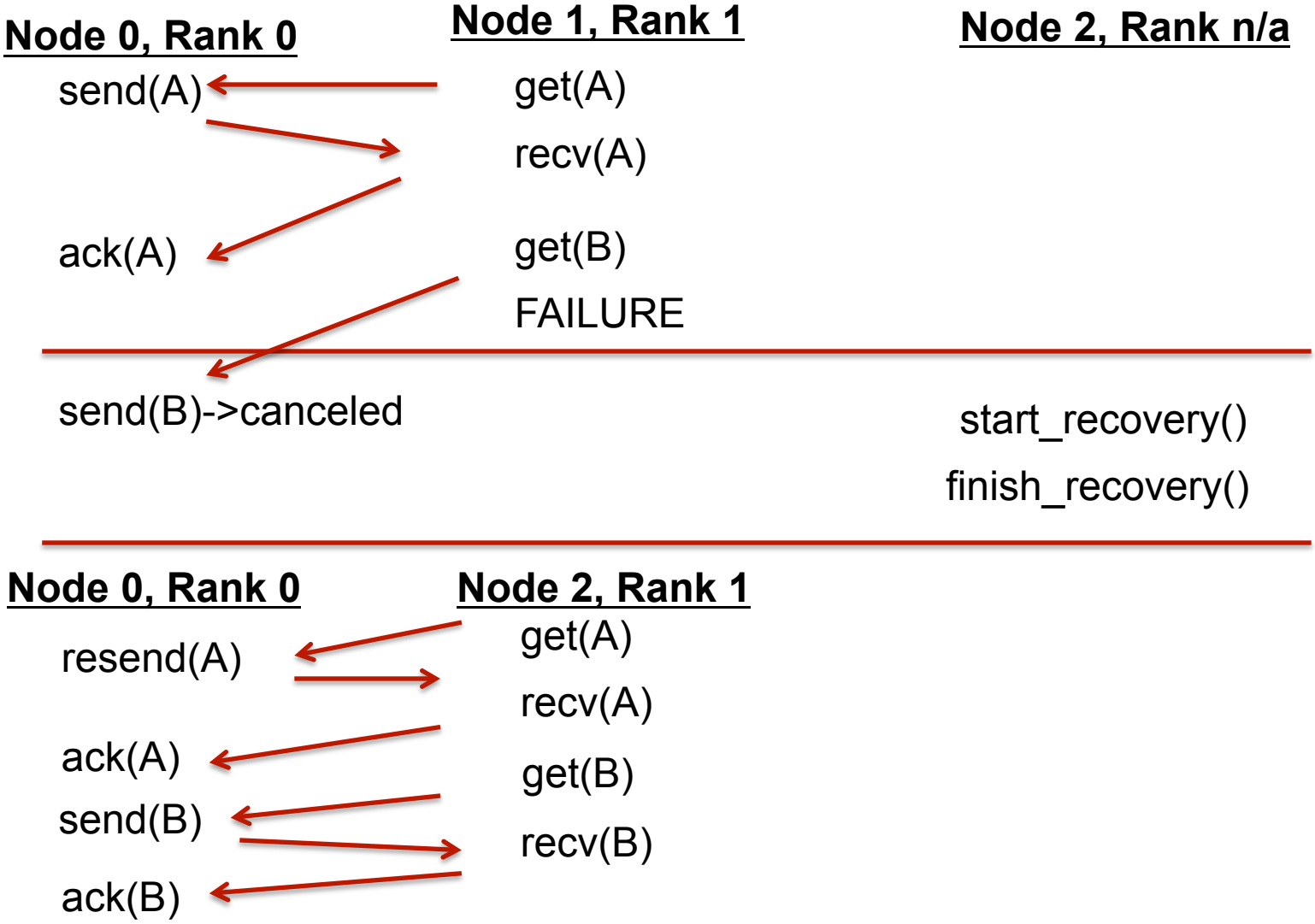
Bytes = serialize(M)
TID = newTransaction()
outgoing[TID] = M
M->type(payload)

M = deserialize(bytes)
blocking_poll()
->M{type=payload}

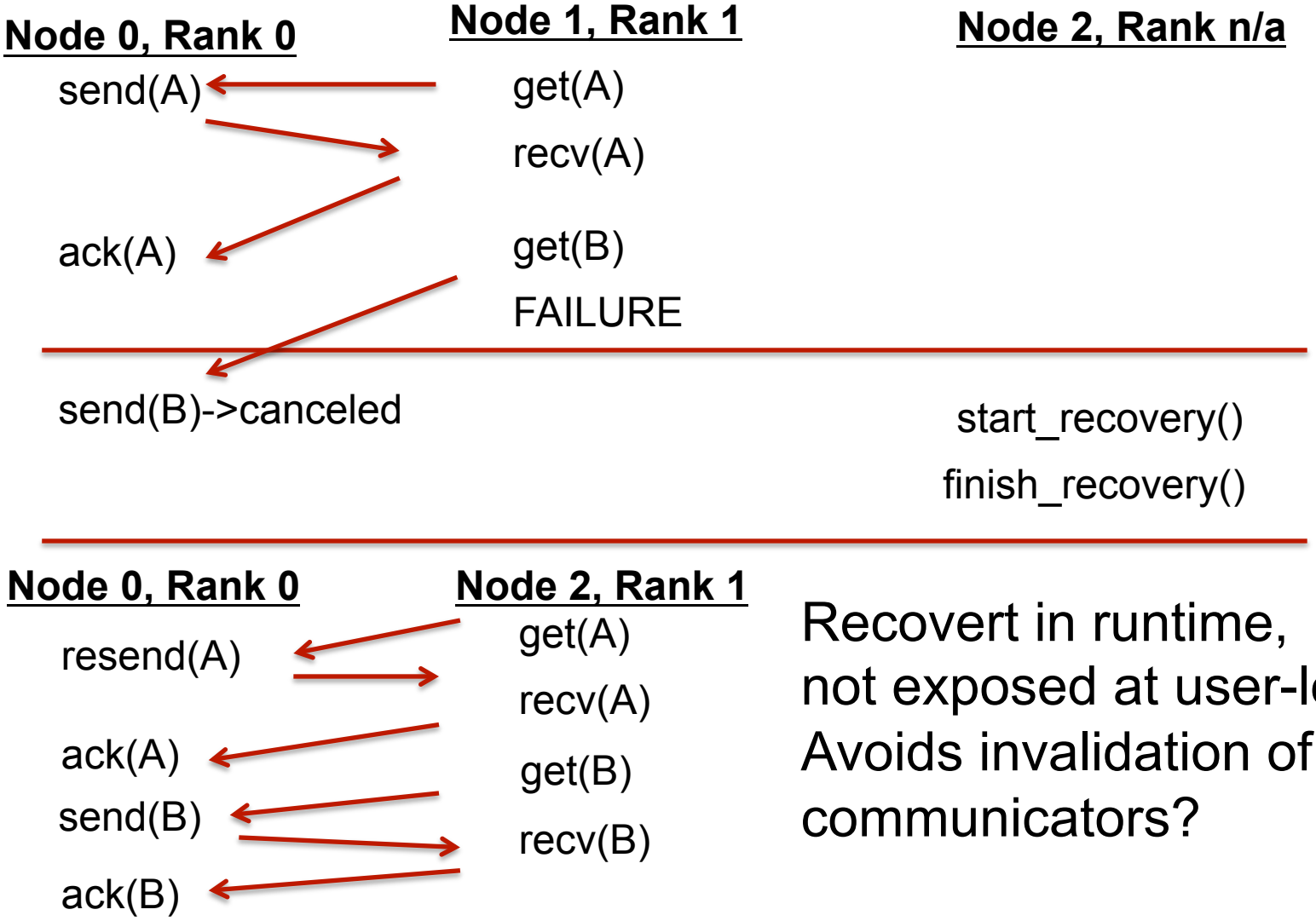
M = outgoing[TID]
M->type(ack)

blocking_poll()
->M{type=ack}

Transport layer aims to provide research tool for fail-stop fault tolerance studies

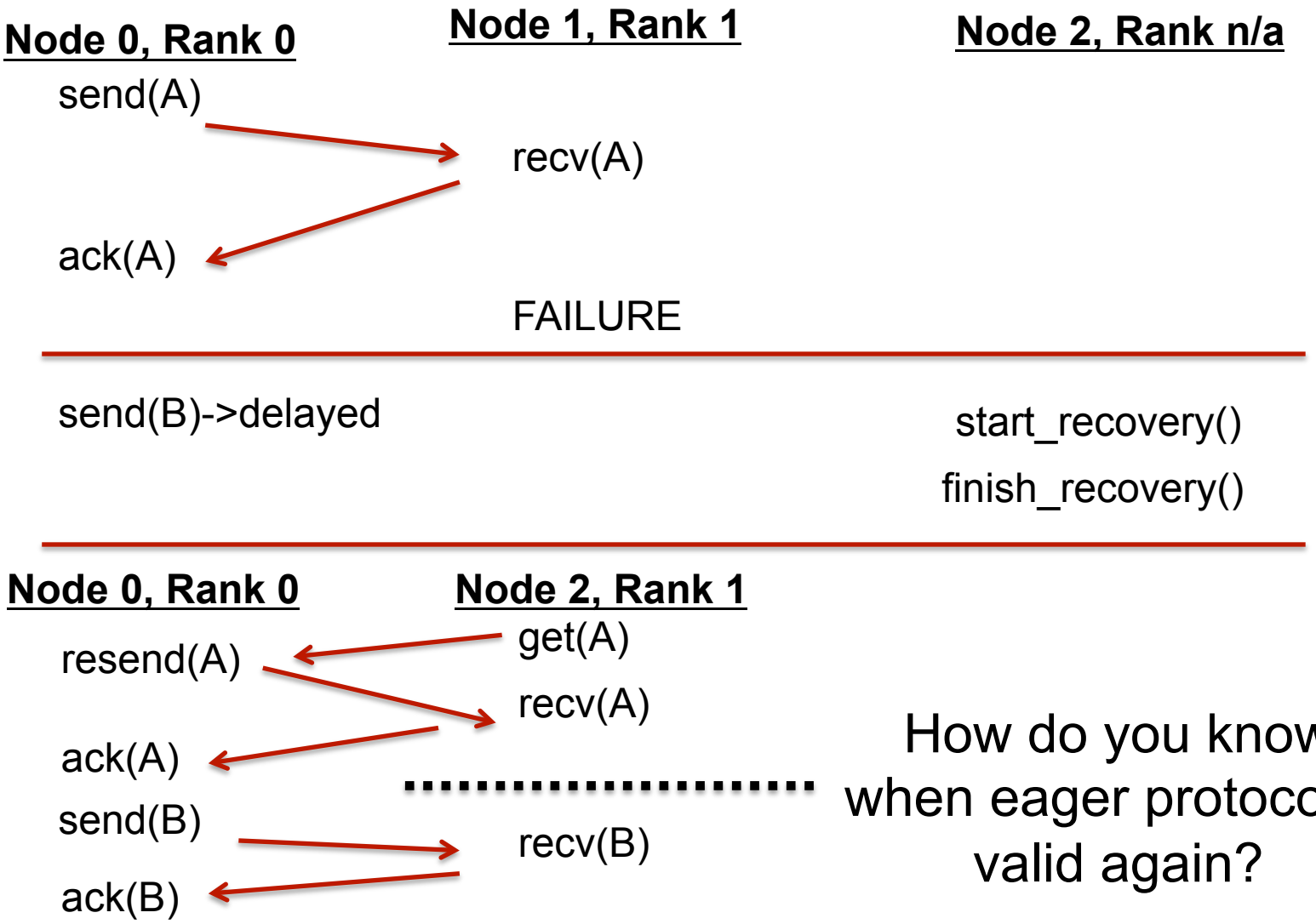


Transport layer aims to provide research tool for fail-stop fault tolerance studies



Recovert in runtime, not exposed at user-level
Avoids invalidation of communicators?

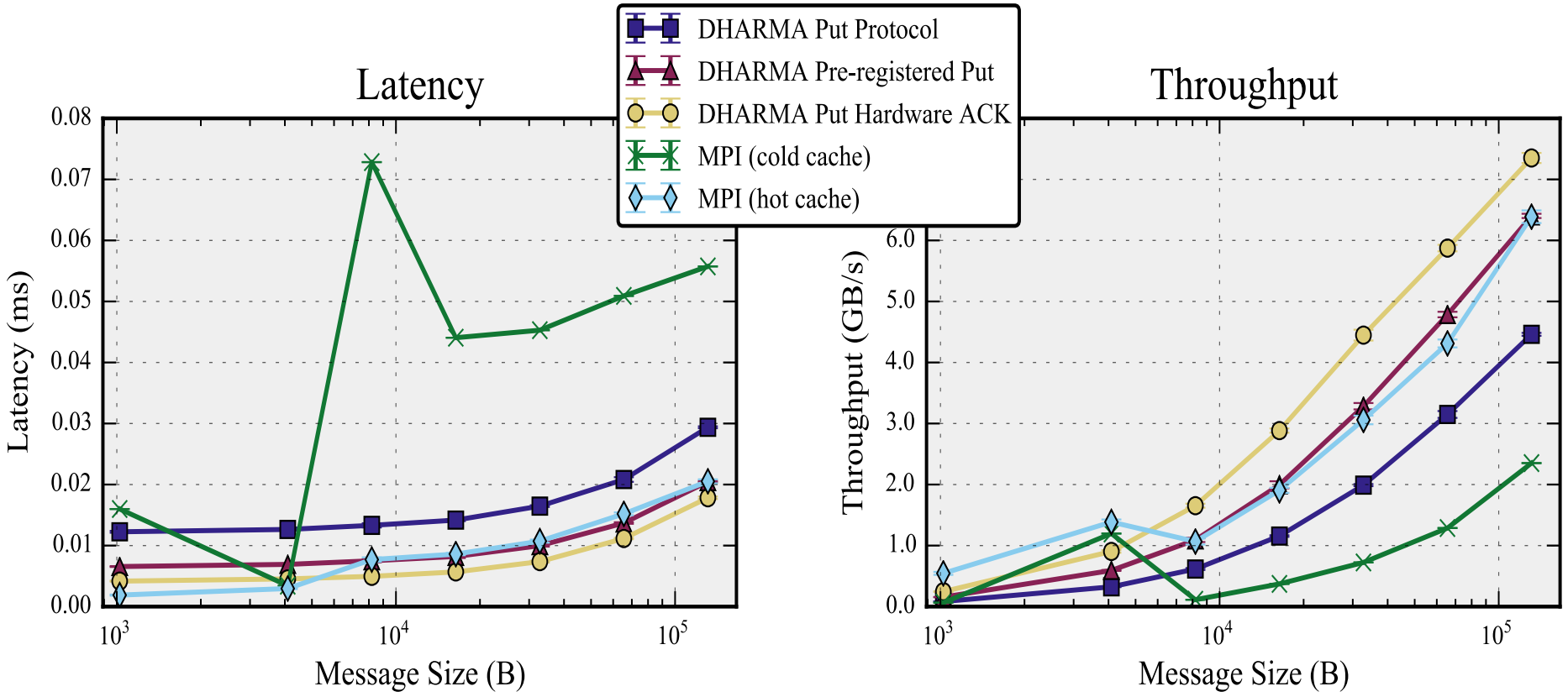
TNSAAFL: Eager protocols now complicated



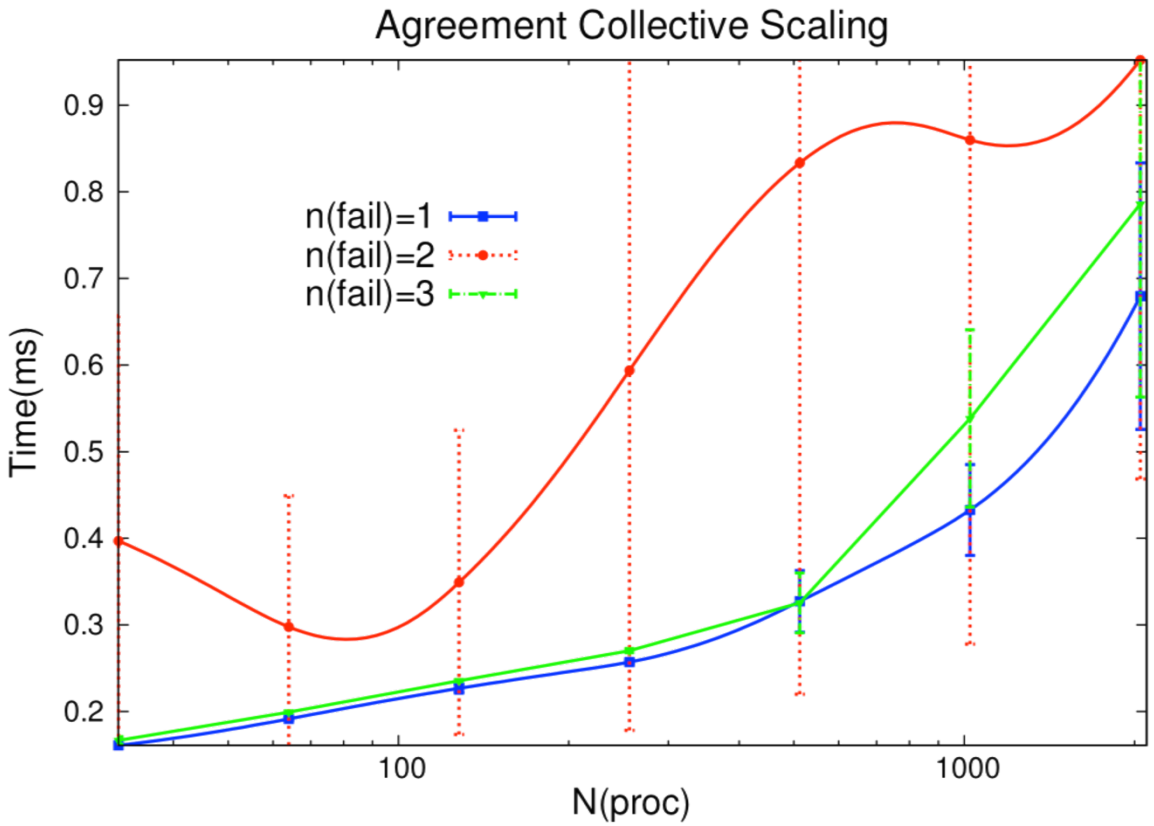
How do you know when eager protocol is valid again?

Key-value store overheads are small compared to network overheads

Sandia National Laboratories

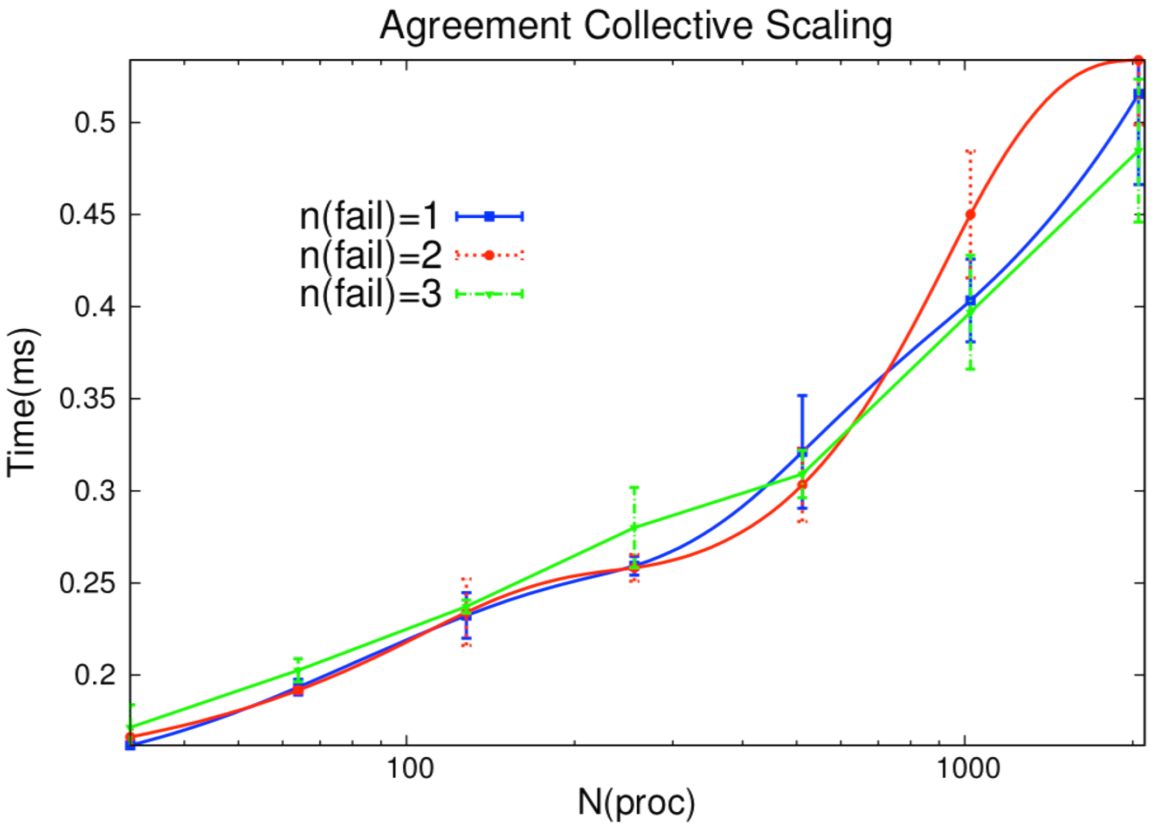


System noise makes scaling studies hard



Edison Cray XC30, 4 procs/node

Ignoring “outliers”, mostly log scaling



Edison Cray XC30, 4 procs/node

Publish/subscribe is extension to 1-sided

Sandia National Laboratories

```
MPIX_KV_Tag tag("mesh", 0, 0, 0);
MPIX_Publish(tag, buffer);

MPIX_Subscribe(tag, buffer);

MPIX_Delivery_fence();
MPIX_Delivery_fence(tag);

MPIX_Wait_delivery(tag);
```

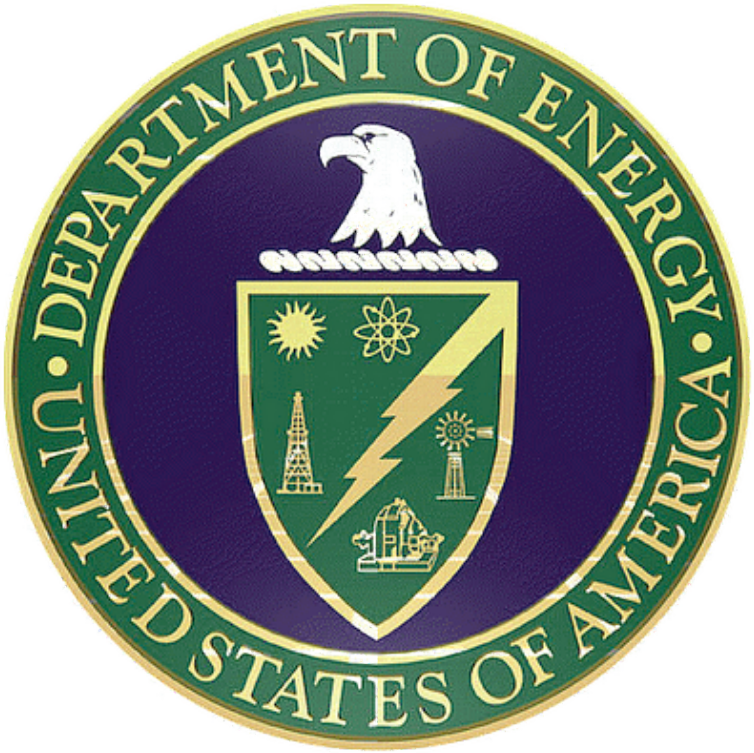
Conclusions (and questions)

Sandia National Laboratories

- Is there a right/wrong way to do “transparent” fail-stop fault-tolerance? Non-shrinking model with no error codes *seems* so much easier...
- Should the standard be no standard? Same core API/ programming model with multiple implementations underneath for application
- General-purpose, application-specific solutions!

Acknowledgments

This work was supported by the U.S. Department of Energy (DOE) National Nuclear Security Administration (NNSA) Advanced Simulation and Computing program and the DOE Office of Advanced Scientific Computing Research. SNL is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the DOE NNSA under contract DE-AC04-94AL85000.



National Nuclear Security Administration

Sandia National Laboratories

Questions?

Sandia National Laboratories