# Fault Tolerance in Message Passing and in Action

## Experiments with Fault Tolerant Linear Algebra Algorithms

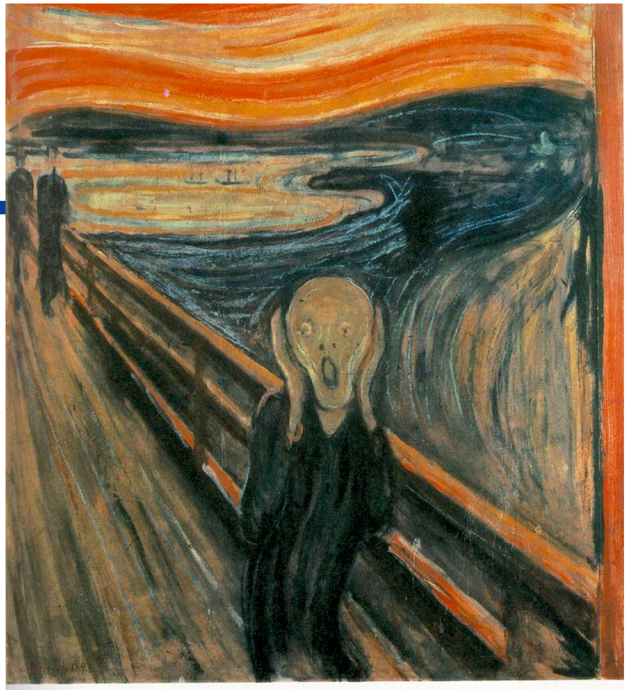
**Jack Dongarra**

**Julien Langou**

**Jeffery Chen**

http://lacsi.rice.edu/review/2004/slides/ft-mpi.pdf

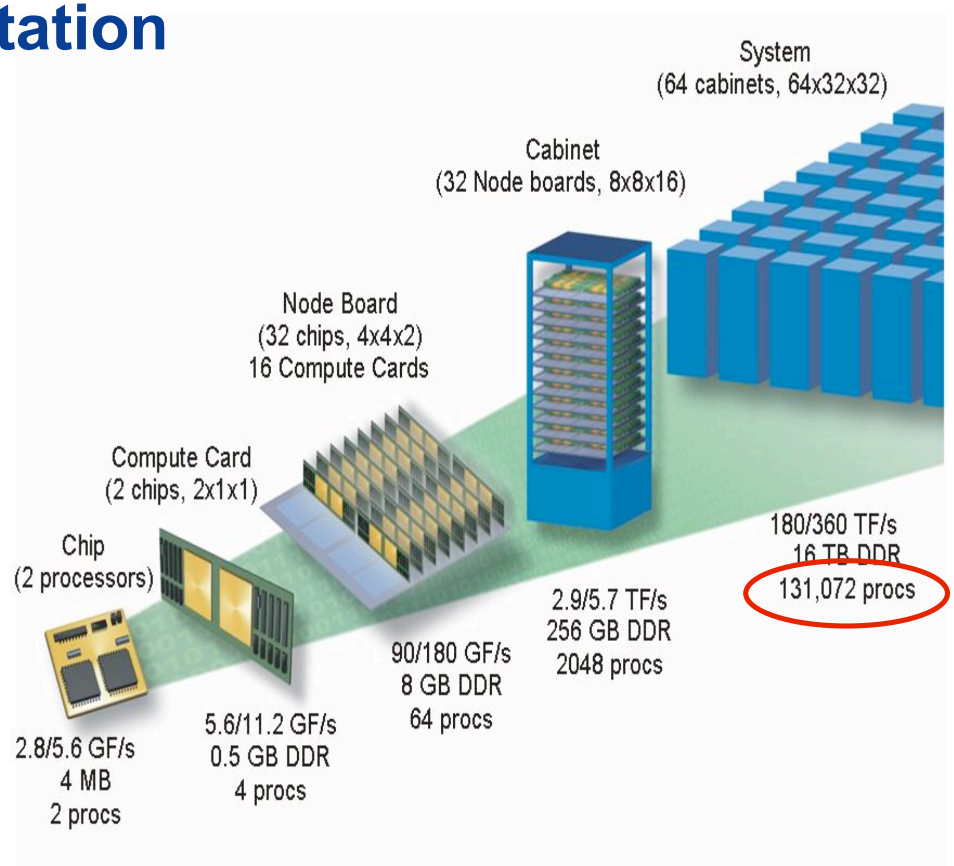
# Fault Tolerance: Motivation



- Trends in HPC:
  - High end systems with thousand of processors
- Increased probability of a system failure
  - Most nodes today are robust, 3 year life
  - Mean Time to Failure is growing shorter as systems grow and devices shrink.
- MPI widely accepted in scientific computing
  - Process faults not tolerated in MPI model
- Mismatch between hardware and (non fault-tolerant) programming paradigm of MPI.

# Fault Tolerance in the Computation

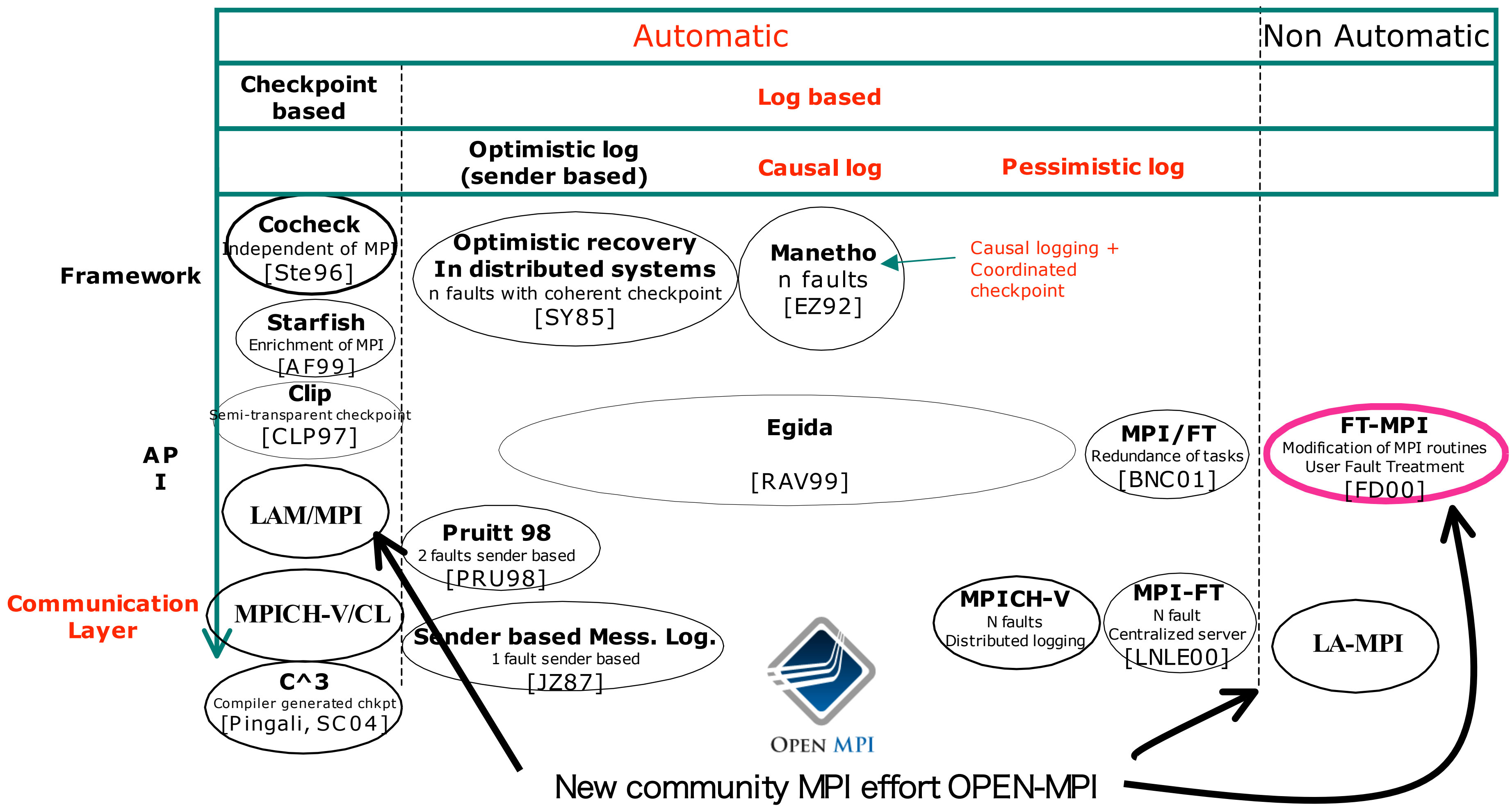
- Some next generation systems are being designed with 100K processors (IBM Blue Gene L)
- MTTF $10^5$ - $10^6$ hours for component
  - sounds like a lot until you divide by $10^5$!
  - Failures for such a system can be just a few hours, perhaps minutes away.
- Application checkpoint / restart is today's typical fault tolerance method.
- Problem with MPI, no recovery from faults in the standard




- Many cluster based on commodity parts don't have error correcting primary memory

# Related work

A classification of fault tolerant message passing environments considering
A) level in the software stack where fault tolerance is managed and
B) fault tolerance techniques.

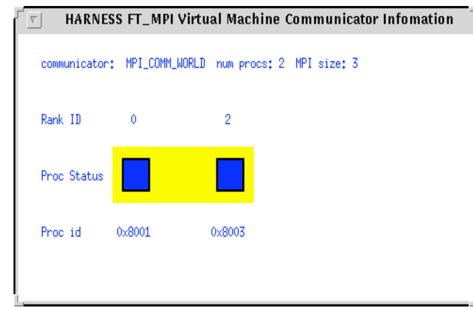


# FT-MPI http://icl.cs.utk.edu/ft-mpi/

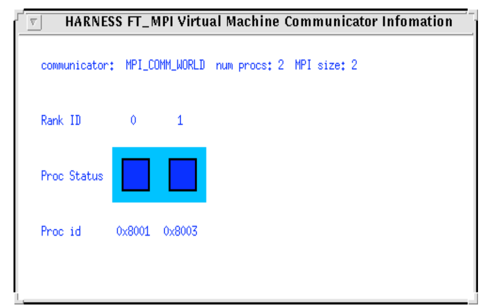
- Define the behavior of MPI in case an error occurs
- FT-MPI based on MPI 1.3 (plus some MPI 2 features) with a fault tolerant model similar to what was done in PVM.
  - Complete reimplementation, not based on other implementations
- Gives the application the possibility to recover from a node-failure
- A regular, non fault-tolerant MPI program will run using FT-MPI
- What FT-MPI does not do:
  - Recover user data (e.g. automatic check-pointing)
  - Provide transparent fault-tolerance

# FT-MPI Failure Recovery Modes

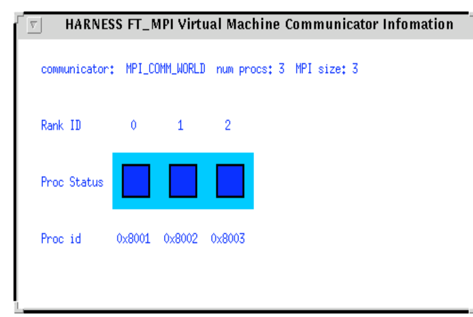
- ABORT: just do as other MPI implementations




- BLANK: leave hole




- SHRINK: re-order processes to make a contiguous communicator
  - Some ranks change




- REBUILD: re-spawn lost processes and add them to MPI_COMM_WORLD

LACSI

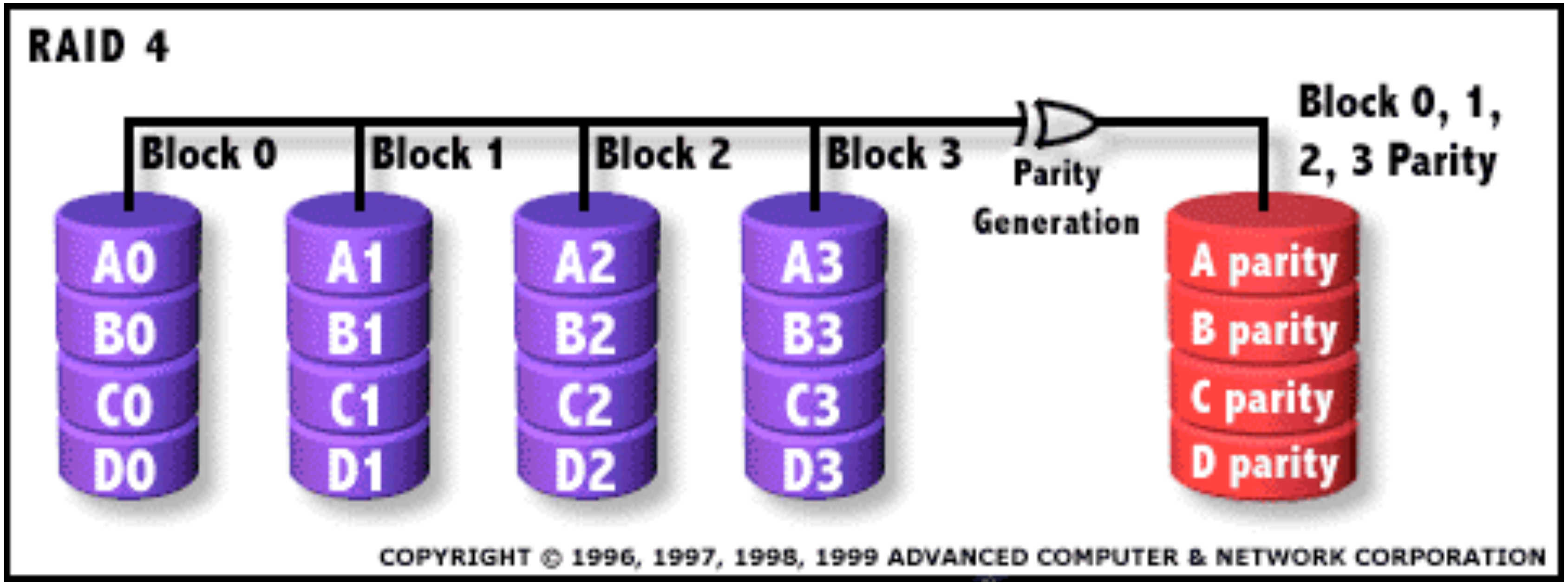
# Fault Tolerance - Diskless Checkpointing Built into Software

- Checkpointing to disk is slow
  - May not have any disks on the system
- Have extra checkpointing processors
- Use RAID like checkpointing to processor
- Maintain a system checkpoint in memory
  - All processors may be rolled back if necessary
  - Use k extra processors to encode checkpoints so that if up to k processors fail, their checkpoints may be restored (Reed-Solomon encoding)
- Idea to build into library routines
  - We are looking at iterative solvers
  - Not transparent, has to be built into the algorithm

LACSI

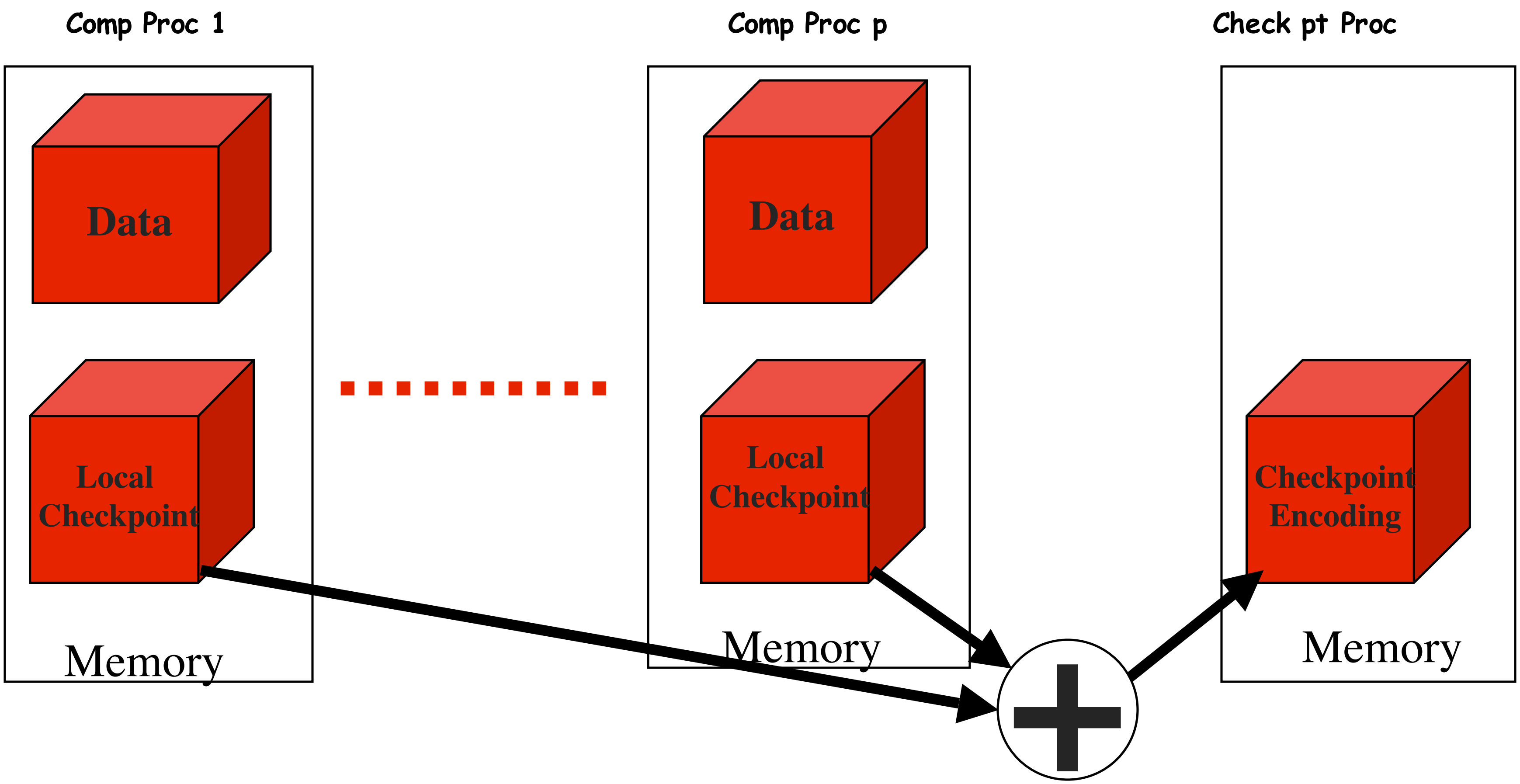
# How Raid for a Disk System Works

- Similar to RAID for disks.




- If X = A XOR B then this is true:
  X XOR B = A
  A XOR X = B

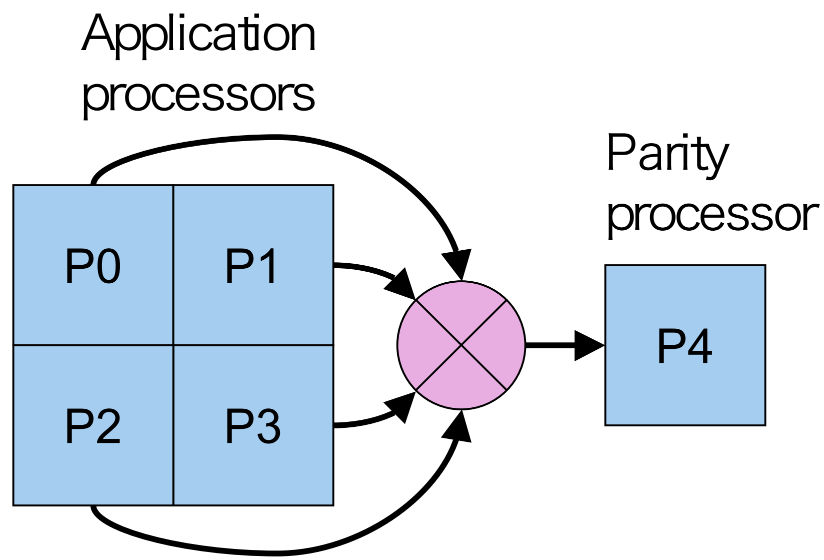
# How Diskless Checkpointing Works




The encoding establishes an equality: $C_1 + C_2 + \ldots C_p = C_{p+1}$
If one of the processor failed, the above equality becomes a linear equation with only one unknown, therefore, lost data can be solved from the equation

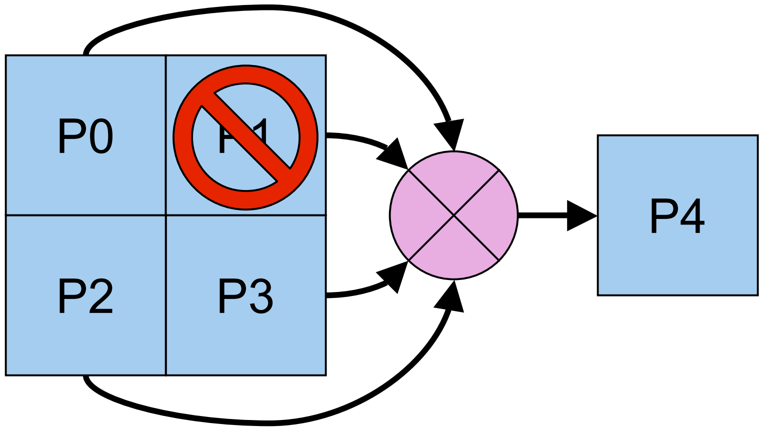
# Diskless Checkpointing

- The N application processors (4 in this case) each maintain their own checkpoints locally.
- K extra processors maintain coding information so that if 1 or more processors fail, they can be replaced.
- Will describe for k=1 (parity)
- If a single processor fails, then its state may be restored from the remaining live processors



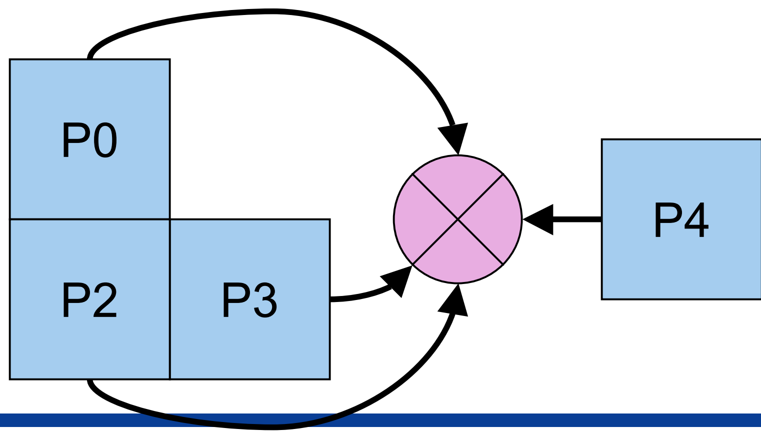

$P4 = P0 \otimes P1 \otimes P2 \otimes P3$

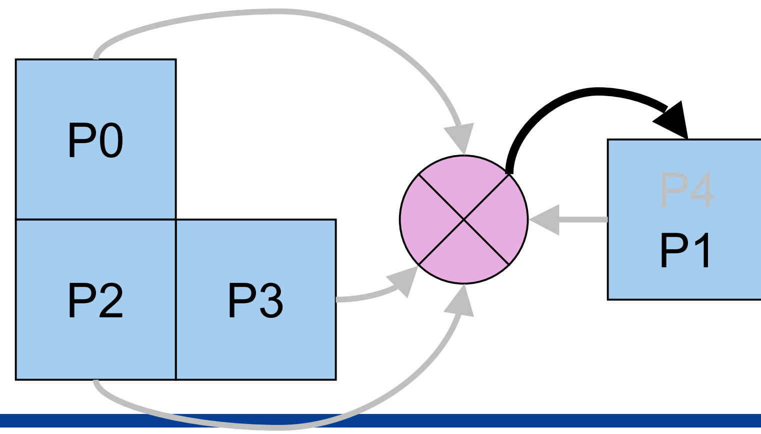
# Diskless Checkpointing




- When failure occurs:
  - control passes to user supplied handler
  - "XOR" performed to recover missing data
  - P4 takes on role of P1
  - Execution continue

P4 takes on the identity of P1
and the computation continues



# Application Scenario with FT-MPI

rc=MPI_Init (…)

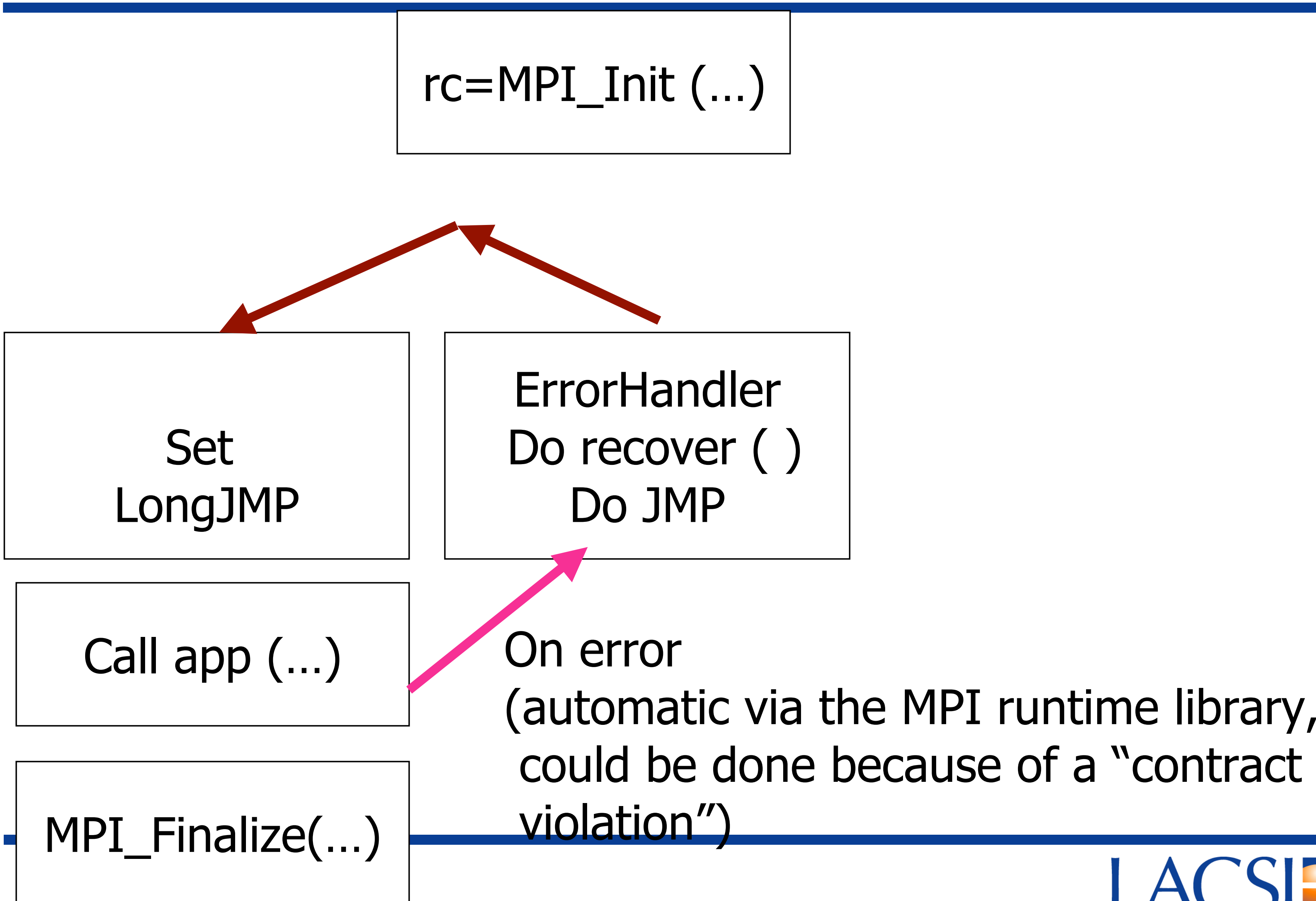
If normal startup

Install Error Handler & Set LongJMP

Call app (…)

MPI_Finalize(…)

# Application Scenario with FT-MPI



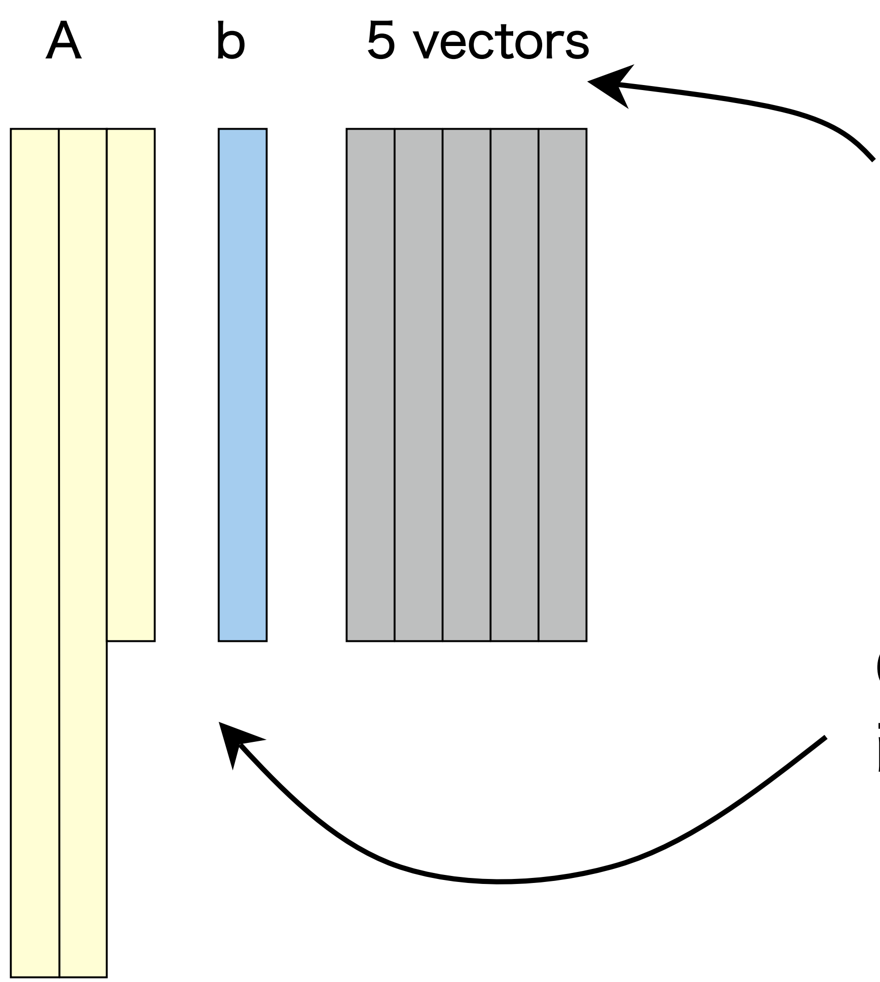

LACSI

# A Fault-Tolerant Parallel CG Solver

- Tightly coupled computation
- Do a "backup" (checkpoint) every *j* iterations for changing data
  - Requires each process to keep copy of iteration changing data from checkpoint
- First example can survive the failure of a single process
- Dedicate an additional process for holding data, which can be used during the recovery operation
- For surviving *k* process failures ($k \ll p$) you need *k* additional processes (second example)

# CG Data Storage

**Think of the data like this**

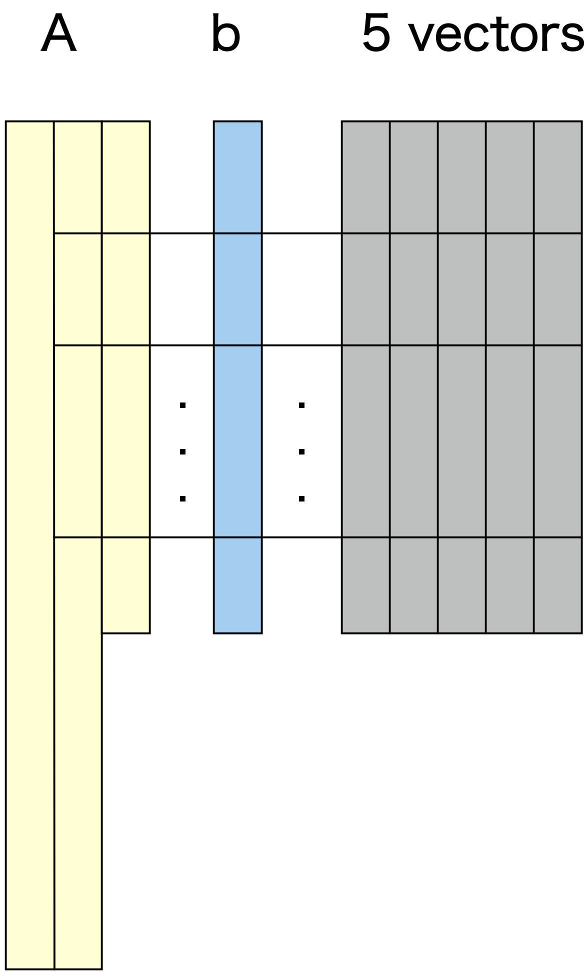



5 vectors change every iteration

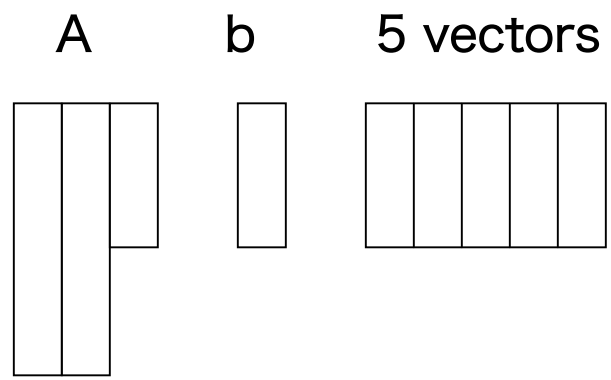
Checkpoint A and b initially

# Parallel Version

**Think of the data like this**



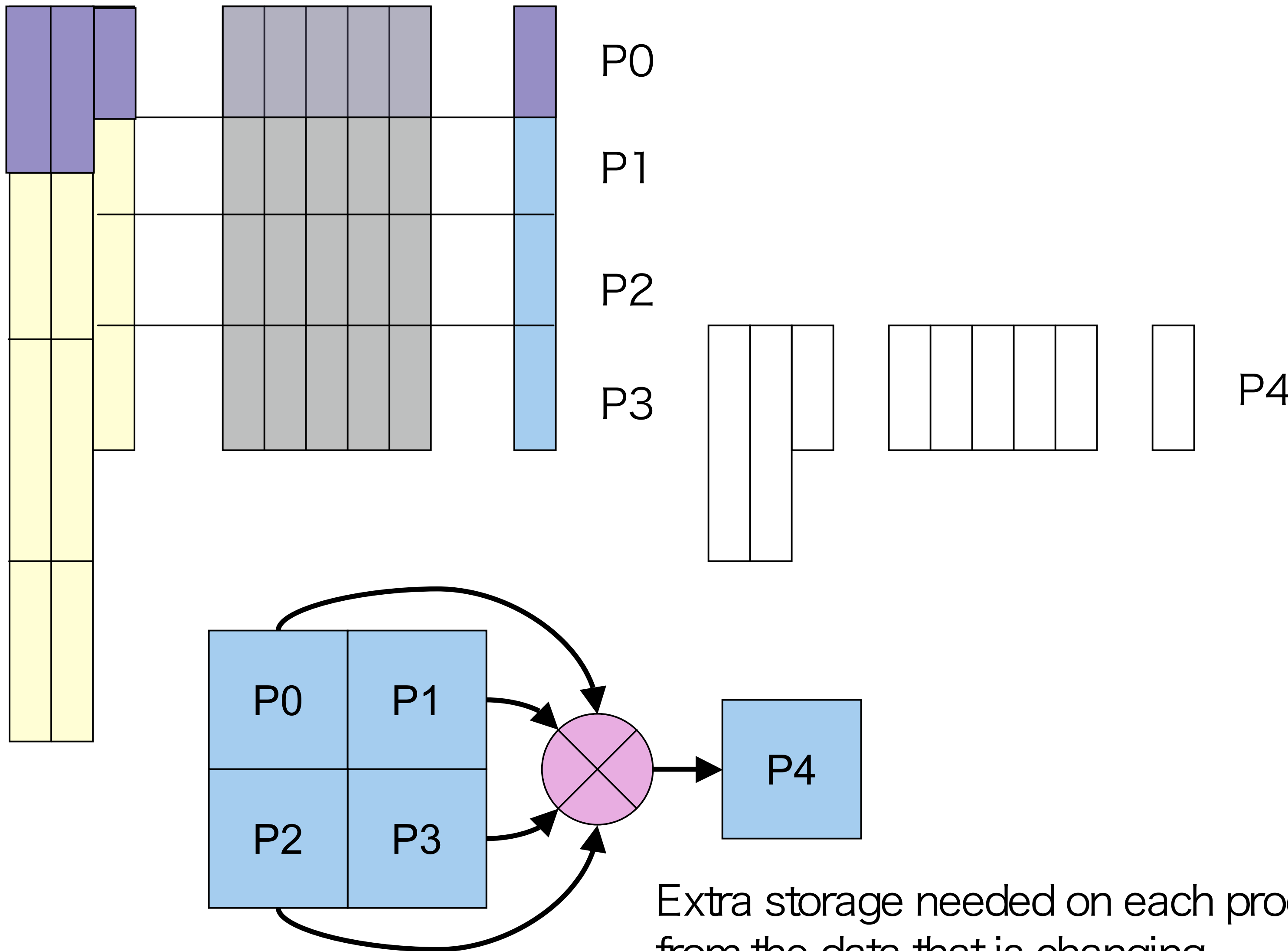

**Think of the data like this on each processor**




No need to checkpoint each iteration, say every *j* iterations.

Need a copy of the 5 vectors from checkpt in each proces

# Diskless Version




Extra storage needed on each process from the data that is changing.
Actually don't do XOR, add the information

# FT PCG Algorithm Analysis

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
**for** $i = 1, 2, \ldots$
  **solve** $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  **if** $i = 1$
    $p^{(1)} = z^{(0)}$
  **else**
    $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  **endif**
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1} / p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
**end**

Global Operations

Global operation in PCG: three dot product, one preconditioning, and one matrix vector multiplication.

Global operation in Checkpoint: encoding the local checkpoint.

LACSI

# FT PCG Algorithm Analysis

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

**for** $i = 1, 2, \ldots$

  **solve** $Mz^{(i-1)} = r^{(i-1)}$

  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$

  **if** $i = 1$

    $p^{(1)} = z^{(0)}$

  **else**

    $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

  **endif**

  $q^{(i)} = Ap^{(i)}$

  $\alpha_i = \rho_{i-1} / p^{(i)^T} q^{(i)}$

  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

  check convergence; continue if necessary

**end**

Checkpoint *x, r,* and *p* every k iterations

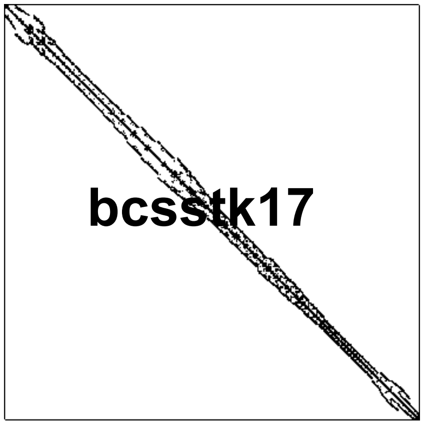
Global Operations

Global operation in PCG: three dot product, one preconditioning, and one matrix vector multiplication.

Global operation in Checkpoint: encoding the local checkpoint.

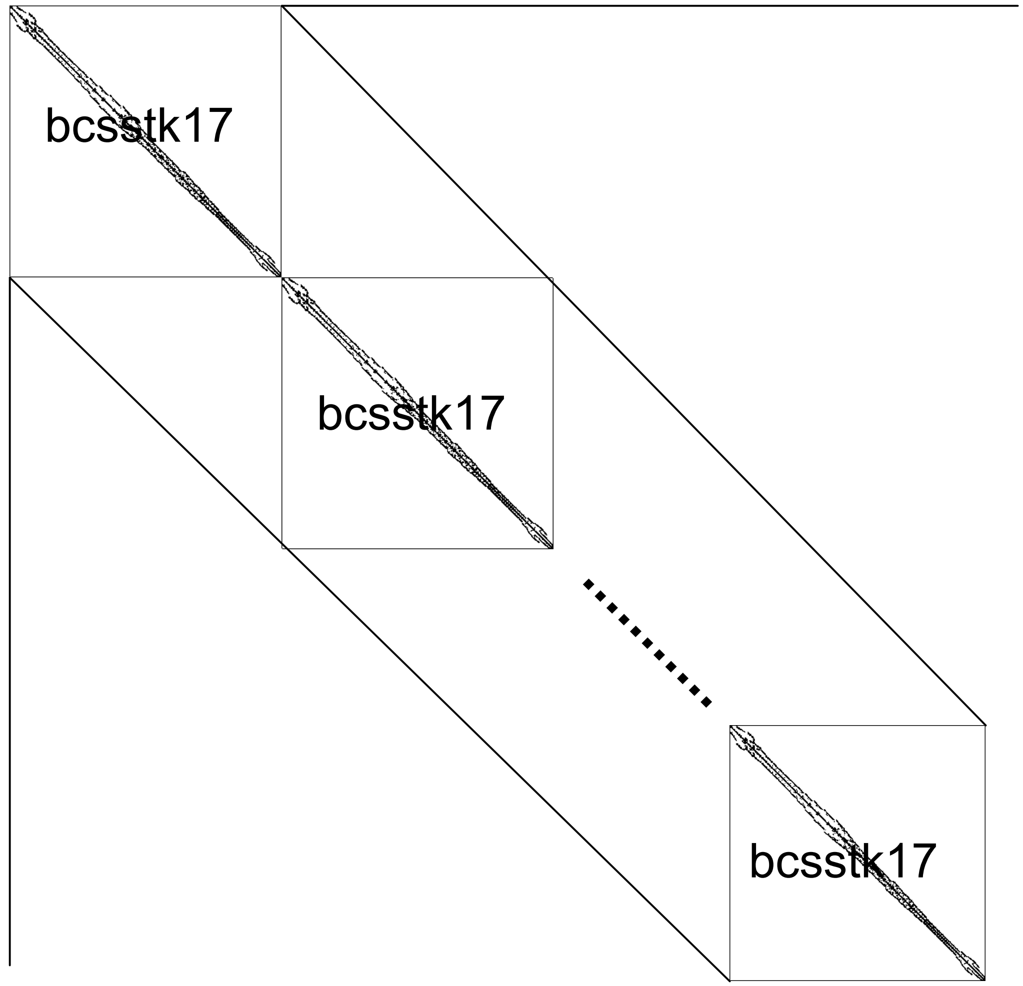
Global operation in checkpoint can be localized by sub-group.

# Test Matrices

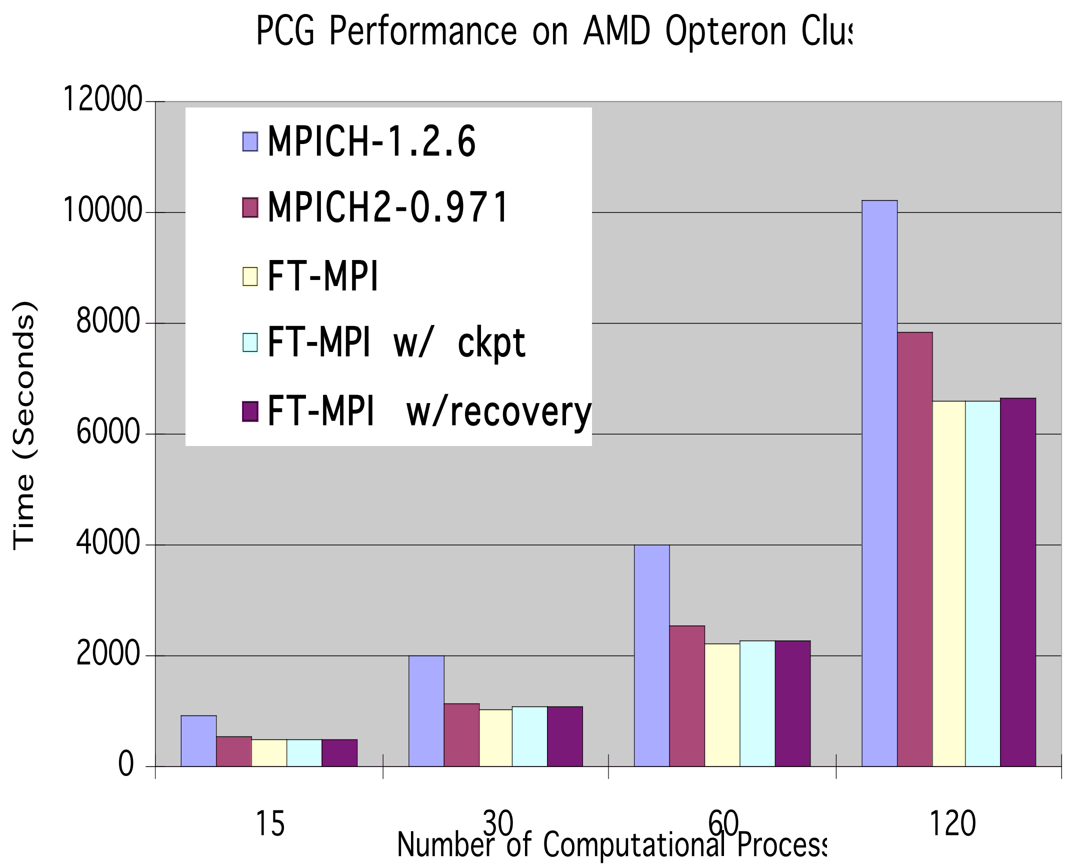



**Bcsstk17:**

- **The size is:**
  - 10974 x 10974
- **Non-zeros:**
  - 428650
- **Sparsity:**
  - 39 non-zeros per row on average
- **Source:**
  - Linear equation from elevated pressure vessel



# PCG Performance (Single Failure)




Test Matrices:
: Modified block diagonal matrices with each processor own a bcsstk17 plus something.

Sparsity:
: 43 non-zeros per row on average.

Total Number of Iterations:
: Run PCG for 2000 iterations.

Checkpoint Status:
: For MPICH runs, there is no checkpoint involved. For FT-MPI runs with checkpoint or recovery, dedicate one additional processor to do checkpoint and do checkpoint at every 100 iterations.

Number of Failures:
: For FT-MPI run with recovery, force one process to fail at the 1000th iterations

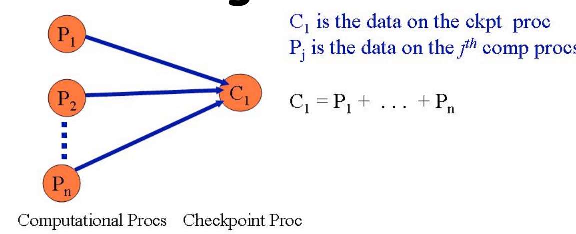
Timing:
: Report the maximum time on all processes. The timer is MPI_Wtime() whose resolutions are 0.003906 seconds for MPICH-1.2.6, 0.000001 for MPICH2-0.971, and 0.000100 for FT-MPI.

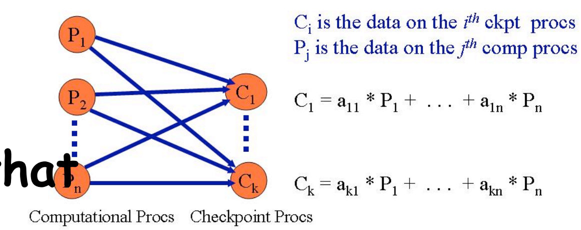
Platform:
: Linux cluster with 64 dual processor 1.4GHz AMD Opteron nodes and Gigabit Ethernet.

| Time (sec) | MPICH-1.2.6 | MPICH2-0.971 | FT-MPI | FT-MPI with checkpoint | FT-MPI with recovery | Checkpoint Ohead (%) | Recovery Ohead (%) |
|---|---|---|---|---|---|---|---|
| 15 proc | 916.2 | 544.0 | 480.3 | 482.7 (2.6) | 485.8 (3.2) | 0.53% | 0.66% |
| 30 proc | 1985.3 | 1120.0 | 1052.2 | 1055.1 (3.8) | 1061.3 (5.0) | 0.36% | 0.47% |
| 60 proc | 4006.8 | 2526.5 | 2241.8 | 2247.5 (5.5) | 2256.0 (8.7) | 0.24% | 0.39% |
| 120 proc | 10199.8 | 7857.4 | 6606.9 | 6614.5 (7.8) | 6634.0(18.2) | 0.11% | 0.27% |

# Protecting for More Than One Failure: Reed-Solomon (Checkpoint Encoding Matrices)

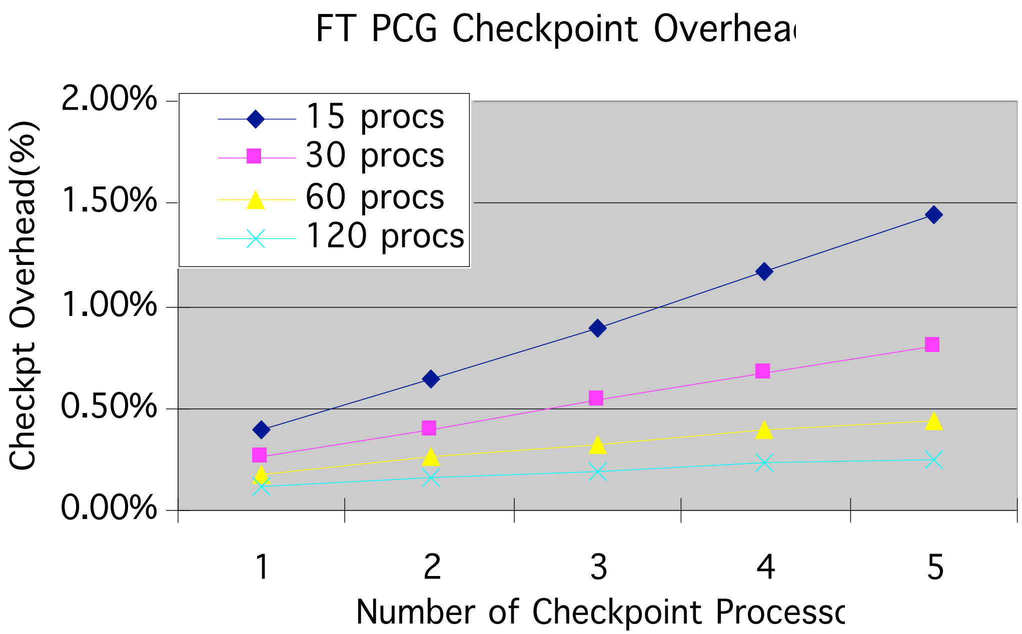
- In order to be able to recover from any k ( $\leq$ number of checkpoint processes ) failures, need a checkpoint encoding.
- With one checkpoint process we had:
  - P sets of data and a function A such that
  - $C=A*P$ where $P=(P_1,P_2,\ldots P_p)^T$;
    - C: Checkpoint data (C and $P_i$ same size)
    - With A = (1, 1, …, 1)
    - $C = a_1P_1 + a_2P_2 + \ldots + a_p P_p$
    - To recover $P_k$;
      solve $P_k = (C-a_1P_1-a_{k-1}P_{k-1}-a_{k+1}P_{k+1}-a_pP_p)/a_k$




- With k checkpoints we need a function A such that
  $C=A*P$ where $P=(P_1,P_2,\ldots P_p)^T$;
    - C: Checkpoint data $C = (C_1,C_2,\ldots C_k)^T$ ($C_i$ and $P_i$ same size)
    - A: Checkpoint-Encoding matrix A is k x p (k << p)
- When h failures occur, recover the data by taking the h x h submatrix of A, call it A', corresponding to the failed processes and solving A'P' = C'.
  - A' is the h x h submatrix
  - C' is made up of the surviving h checkpoints

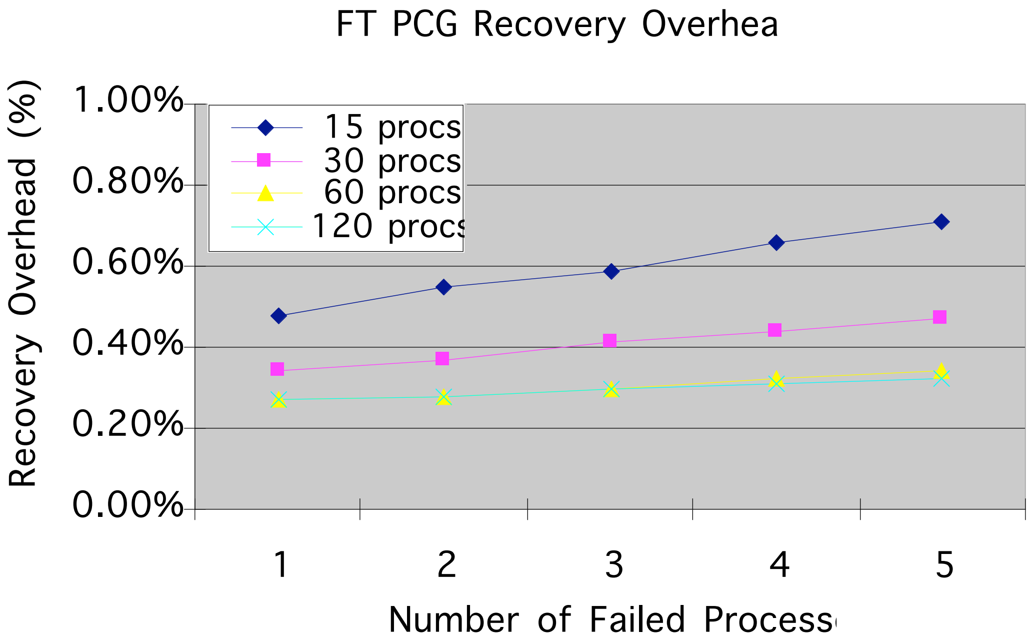
# FT PCG Checkpoint Overhead




Test Matrices:
Modified block diagonal matrices with each processor own a bcsstk17 plus something.
Sparsity:
43 non-zeros per row on average.
Total Number of Iterations:
Run FT PCG for 2000 iterations.
Checkpoint Interval:
Checkpoint at every 100 iterations.
Number of Failures:
There is no failure in this experiment.
Timing:
Maximum time on all processes. The timer is MPI_Wtime() whose resolution is 0.0001 second.
Platform:
Linux cluster with 64 dual processor 1.4GHz AMD Opteron nodes and Gigabit Ethernet.

| **Overhead (%)** | **0 ckpt** | **1 ckpt** | **2 ckpt** | **3 ckpt** | **4 ckpt** | **5 ckpt** |
|---|---|---|---|---|---|---|
| **15 procs** | **0.0%** | **0.39%** | **0.64%** | **0.89%** | **1.17%** | **1.45%** |
| **30 procs** | **0.0%** | **0.26%** | **0.39%** | **0.54%** | **0.67%** | **0.81%** |
| **60 procs** | **0.0%** | **0.17%** | **0.26%** | **0.32%** | **0.39%** | **0.44%** |
| **120 procs** | **0.0%** | **0.11%** | **0.16%** | **0.19%** | **0.23%** | **0.25%** |
| Time (seconds) | 0 ckpt | 1 ckpt | 2 ckpt | 3 ckpt | 4 ckpt | 5 ckpt |
| 15 procs | 662.4 | 666.7 (2.6) | 668.3 (4.4) | 671.0 (6.0) | 673.7 (7.9) | 674.6 (9.8) |
| 30 procs | 1463.2 | 1466.1 (3.8) | 1470.1 (5.8) | 1471.9 (7.9) | 1471.9 (9.9) | 1472.6 (11.9) |
| 60 procs | 3216.6 | 3220.8 (5.5) | 3222.9 (8.5) | 3225.7 (10.2) | 3227.9 (12.6) | 3232.2 (14.1) |
| 120 procs | 6606.9 | 6614.5 (7.8) | 6616.9 (10.6) | 6619.7(12.8) | 6622.3(15.0) | 6625.1(16.8) |

# FT PCG Recovery Overhead




Test Matrices:

Modified block diagonal matrices with each processor own a bcsstk17 plus something.

Total Number of Iterations:

Run FT PCG for 2000 iterations.

Checkpoint Interval:

Checkpoint at every 100 iterations.

Recovery Frequency:

One recovery. Force some processes to fail at the 300th iteration.

Timing:

Maximum time on all processes. The timer is MPI_Wtime() whose resolution is 0.0001 second.

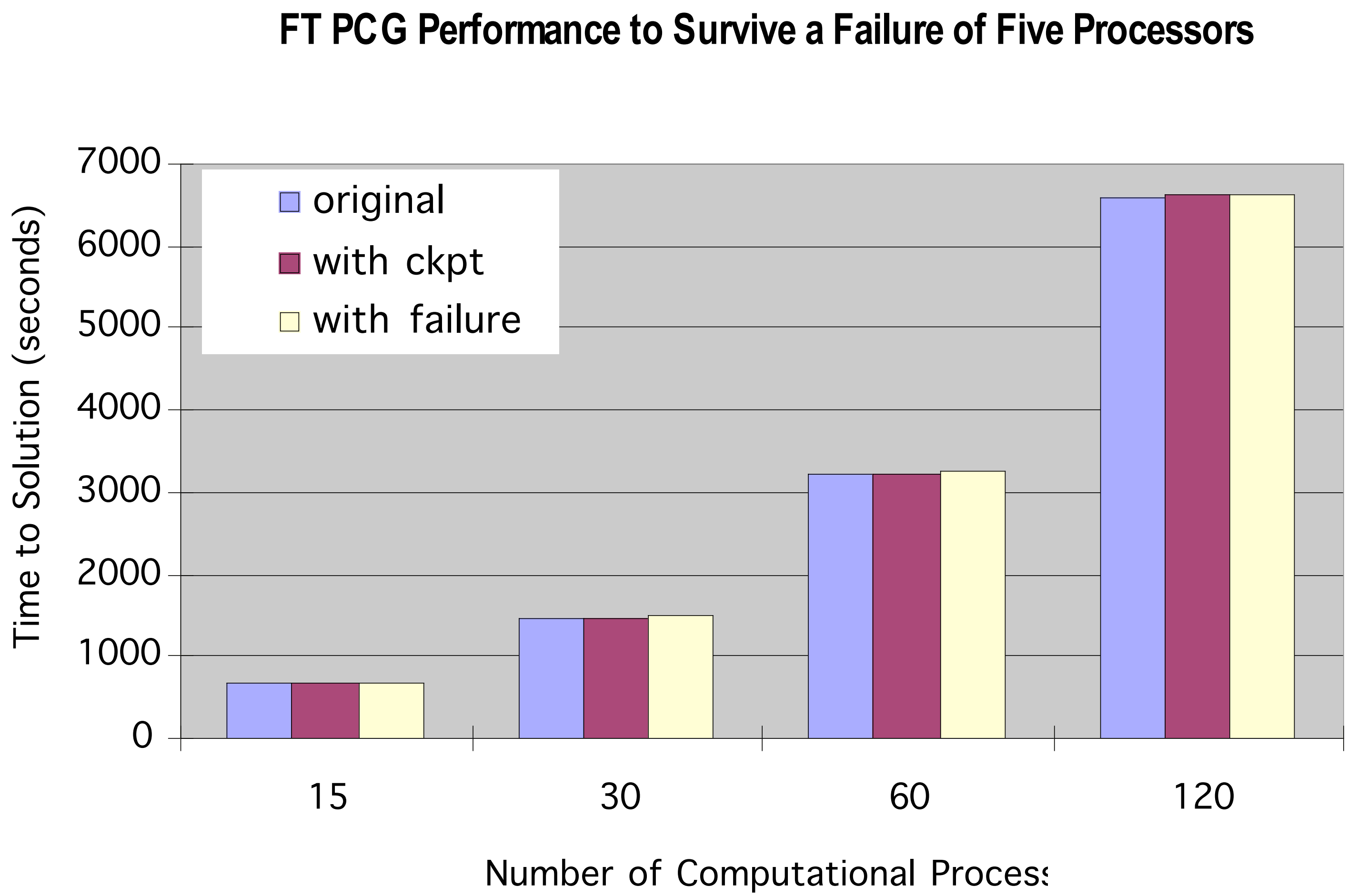
Platform:

Linux cluster with 64 dual processor 1.4GHz AMD Opteron nodes and Gigabit Ethernet.

| **Overhead (%)** | **0 proc-failures** | **1 proc-failures** | **2 proc-failures** | **3 proc-failures** | **4 proc-failures** | **5 proc-failures** |
|---|---|---|---|---|---|---|
| **15 procs** | **0.0%** | **0.48%** | **0.55%** | **0.59%** | **0.66%** | **0.71%** |
| **30 procs** | **0.0%** | **0.34%** | **0.37%** | **0.41%** | **0.44%** | **0.47%** |
| **60 procs** | **0.0%** | **0.27%** | **0.28%** | **0.30%** | **0.32%** | **0.34%** |
| **120 procs** | **0.0%** | **0.27%** | **0.28%** | **0.30%** | **0.31%** | **0.32%** |

| Time (seconds) | 0 proc-failures | 1 proc-failures | 2 proc- failures | 3 proc-failures | 4 proc-failures | 5 proc-failures |
|---|---|---|---|---|---|---|
| 15 procs | 662.4 | 670.0 (3.2) | 672.1 (3.7) | 676.0 (4.0) | 677.9 (4.5) | 679.8 (4.8) |
| 30 procs | 1463.2 | 1469.0 (5.0) | 1472.6 (5.5) | 1476.4 (6.0) | 1477.7 (6.5) | 1480.1 (7.0) |
| 60 procs | 3216.6 | 3230.1 (8.7) | 3231.1 (9.2) | 3235.1 (9.8) | 3237.0 (10.4) | 3239.1 (11.1) |
| 120 procs | 6606.9 | 6634.0(18.2) | 6633.5(18.8) | 6636.3 (20.0) | 6638.2 (20.9) | 6639.7 (21.5) |

# FT PCG Performance



# Next Steps

Investigate ideas for 1K to 10K processors, then to BG/L:

- Software to determine the checkpointing interval and number of checkpoint processors from the machine characteristics.
  - Perhaps use historical information
- Local checkpoint and restart algorithm.
  - Coordination of local checkpoints.
  - Processors hold backups of neighbors.
- Have the checkpoint processes participate in the computation and do data rearrangement when a failure occurs.
  - Use p processors for the computation and have k of them hold checkpoint.
- Generalize the ideas to provide a library of routines to do the diskless check pointing.
- Real problems
- Investigate Lossy algorithms