LOAN DOCUMENT

PHOTOGRAPH THIS SHEET

DTIC ACCESSION NUMBER

LEVEL

INVENTORY

D

Software Fault Tolerance for Parallel Signal Processors

DOCUMENT IDENTIFICATION

19 Jul 2000

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

DISTRIBUTION STATEMENT

HANDLE WITH CARE

ACCESSION FOR

NTIS	GRA&I	☑
DTIC	TRAC	☐
UNANNOUNCED		☐
JUSTIFICATION		

BY

DISTRIBUTION/

AVAILABILITY CODES

DISTRIBUTION	AVAILABILITY AND/OR SPECIAL	
A-1		

DISTRIBUTION STAMP

DATE ACCESSIONED

DATE RETURNED

20000814 203

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NUMBER

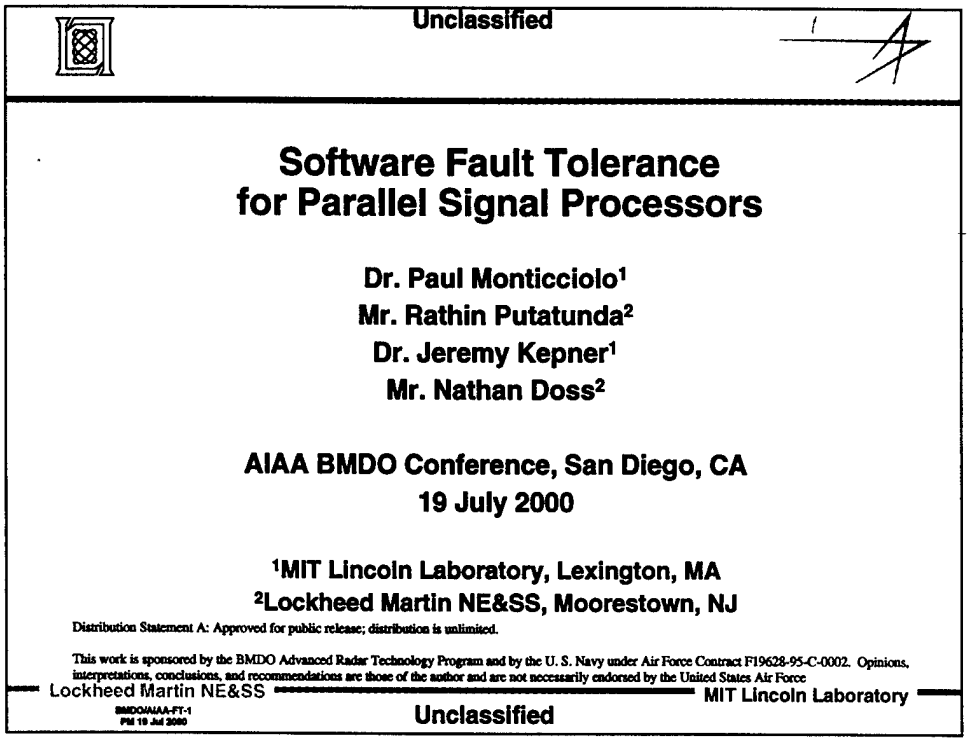
PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

DTIC FORM 70A JUN 90

DOCUMENT PROCESSING SHEET

PREVIOUS EDITIONS MAY BE USED UNTIL STOCK IS EXHAUSTED.

LOAN DOCUMENT

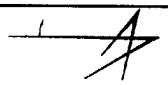


Abstract

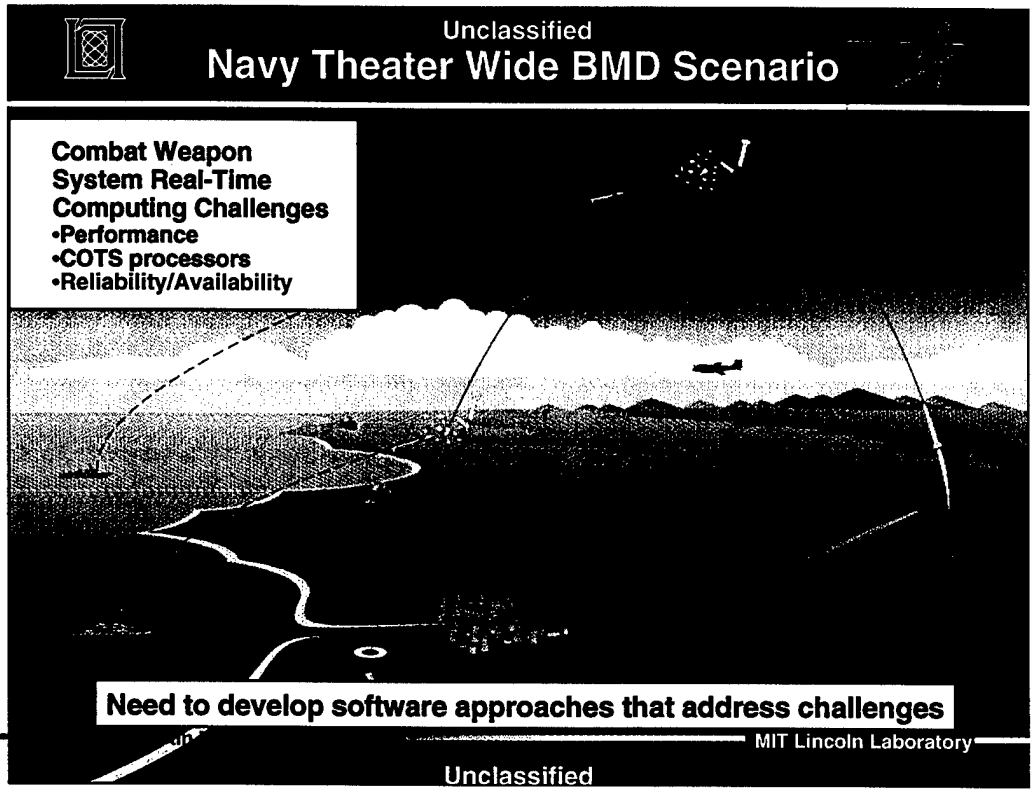
Military combat weapon systems are becoming increasingly reliant on advanced multiprocessor computer architectures to execute powerful real-time signal processing algorithms. These multiprocessor computer architectures must meet stringent system reliability, availability, and maintainability requirements. This paper addresses software-based fault tolerance techniques that can leverage inherent capabilities of multiprocessor computers to minimize overall system hardware redundancy requirements and provide quick, flexible fault detection, isolation, and recovery. An AEGIS SPY radar signal processing system example is used to illustrate fault tolerance approaches being jointly developed by MIT Lincoln Laboratory and Lockheed Martin Naval Electronic and Surface Surveillance.

Unclassified

Outline



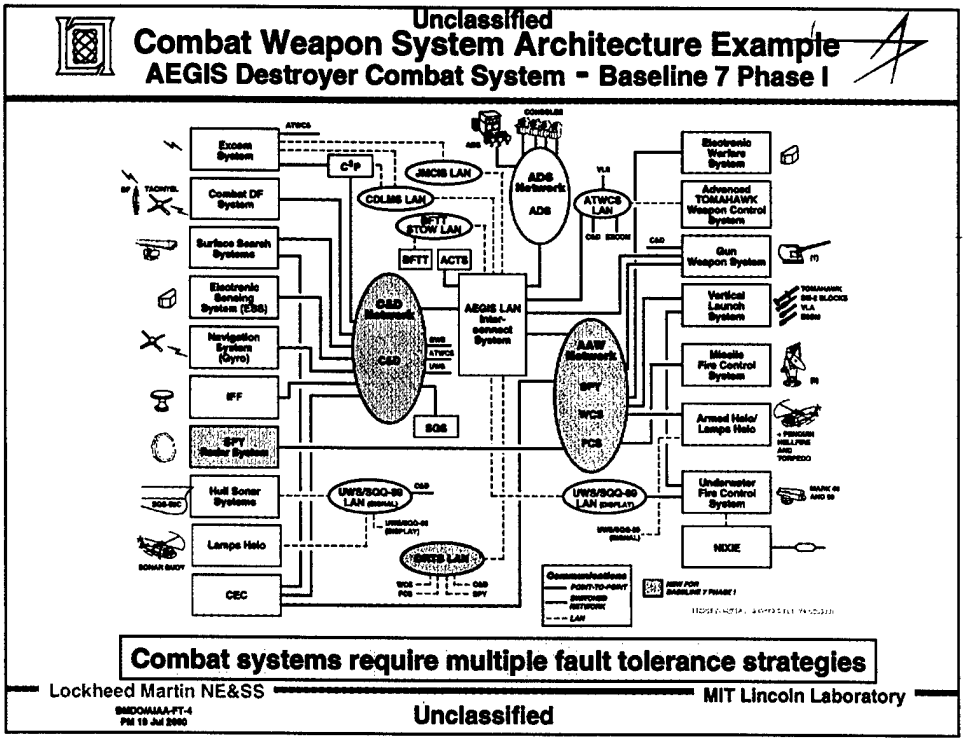
- Introduction
- Fault tolerance concepts and examples
- Common Signal Processor Application Programmer Interface (CSP API)
- Summary



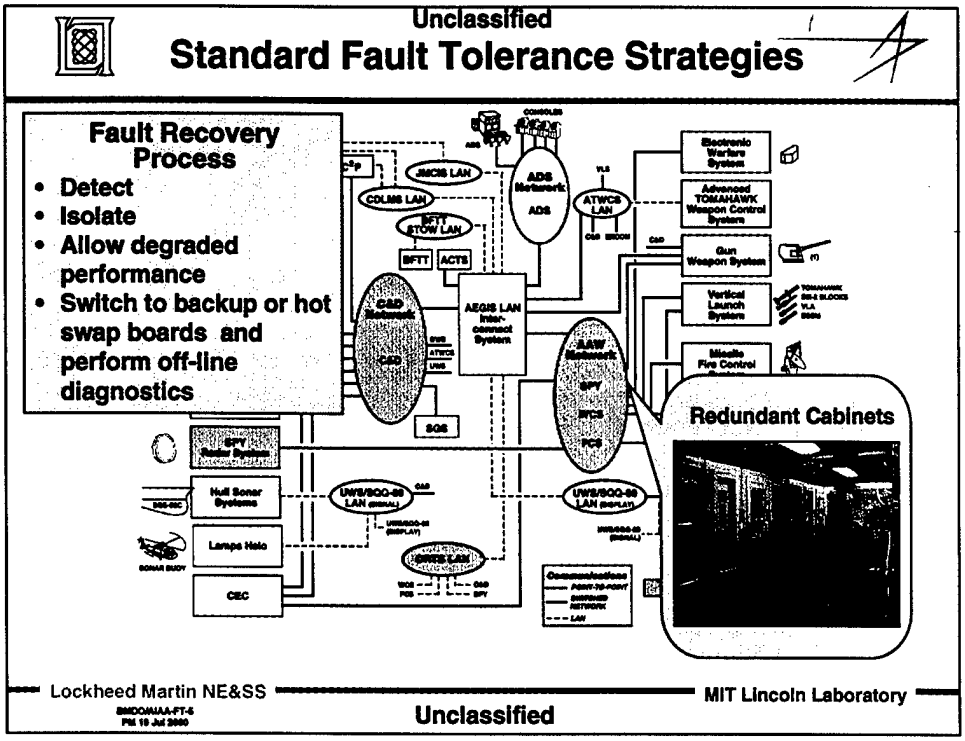
The above picture illustrates an example TBMD littoral environment scenario with AEGIS systems attempting to perform exoatmospheric and/or endoatmospheric reentry vehicle intercept. The AEGIS system must operate at very high availability rates, i.e., low failure rates combined with quick recovery or equipment replacement times, while trying detect, track, and intercept ballistic missiles despite enemy countermeasures such as ground-based and airborne jammers. Real-time execution of the necessary signal processing algorithms is expected to require total processor throughput exceeding 1 Gflops/s (one-billion floating point operations per second).

Commercial off-the-shelf (COTS) microprocessor-based Massively Parallel Processing (MPP) systems have been shown to be capable of meeting TBMD mission-related throughput and latency requirements. In addition to exploiting the high performance of COTS systems, the government is not required to fund chip design, foundry construction, and production which can exceed one-billion dollars. MPP systems present the opportunity to implement new software-based approaches to processor fault tolerance by using spare processor elements or multiprocessor boards to replace failed processor(s) via mechanisms embedded in high-level software constructs.

This talk will focus on software techniques that will achieve high level of reliability and availability.



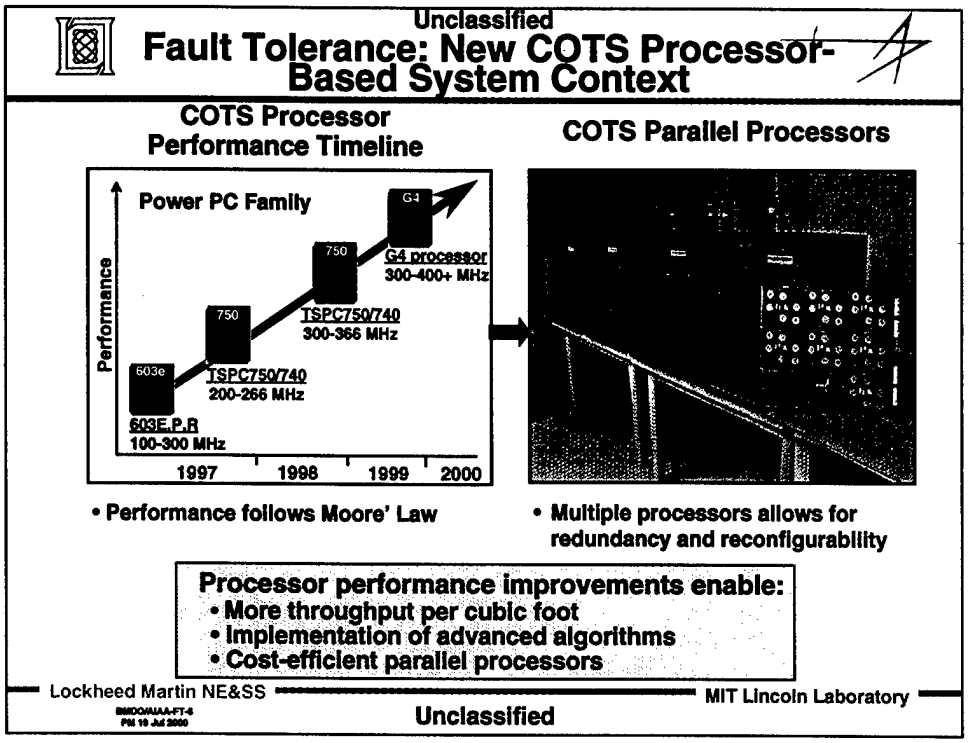
Military combat systems can be extremely complex as depicted by a recent AEGIS Destroyer combat system architecture in the above figure. Potential failures can occur in weapons subsystems, control computers, special-purpose signal processors, and communications systems and networks. Fault tolerance strategies that can meet system reliability and availability requirements typically consist of the combination of high levels of redundancy with the quick isolation and replacement of faulty components.



The fault tolerance process begins with the detection of a system or processor fault using monitoring and built-in test programs designed to check system performance. If faults such as failed data channels, failed processors/boards/cabinets, erratic communications, or computational errors are detected, programs and indicators are used to isolate and locate the fault. The fault could yield degraded system performance (e.g., require operation on fewer radar pulses) or result in complete system failure.

To recover from the fault, a board could be hot swapped in or a redundant system can be switched in. Fully redundant processor systems have several disadvantages: larger footprints, extra weight, increased power requirements, and the complete loss of one system during offline fault recovery/maintenance.

To complete the process, off-line maintenance would be performed to determine the cause of the fault and perform the necessary repair.

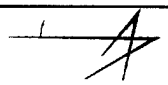


The upper left-hand figure depicts what is commonly referred to as Moore's Law, i.e., computational power doubles approximately every 18 months. This particular example is for the Motorola Power PC family, a set of processors currently popular in Massively Parallel Processing systems. Improved processor performance enables more computational power per cubic foot (particularly important for size-constraint applications such as missile seeker processors), the implementation of advanced radar and discrimination algorithms (e.g., wideband high-range radar resolution necessary for exoatmospheric target discrimination and superresolution range-doppler image processing), and the implementation of ultra-high-performance parallel processors as depicted in the upper right-hand figure.

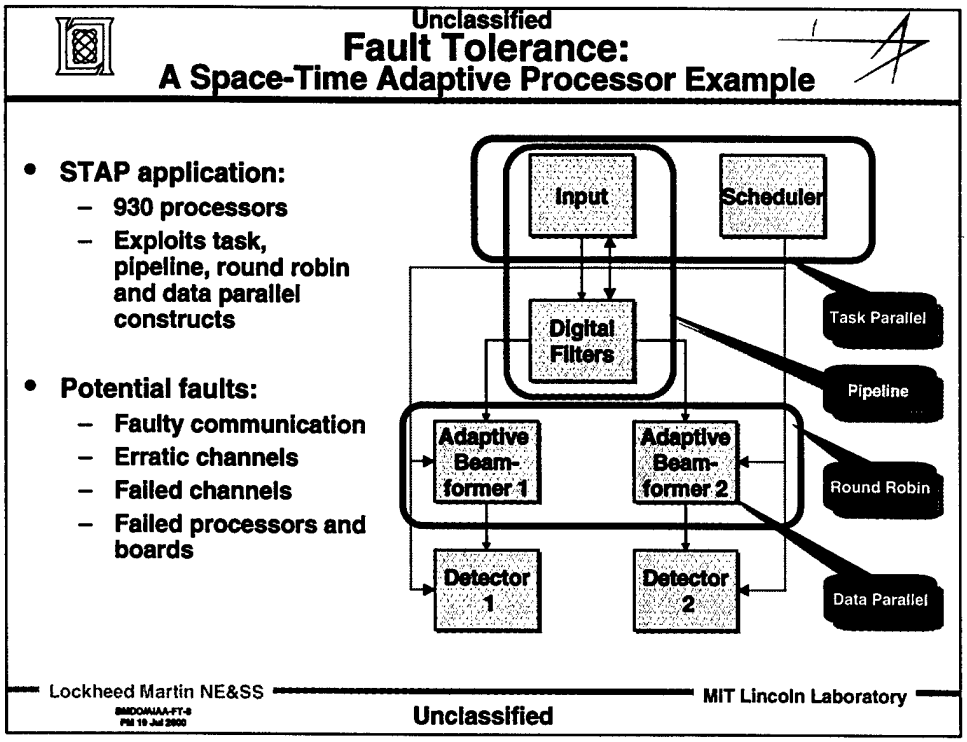
These parallel processors systems will require the development of new fault tolerance strategies to exploit inherent processor redundancy and system reconfiguration capabilities to replace current hot swap/redundancy practices.

Unclassified

Outline



- Introduction
- Fault tolerance concepts and examples
- Common Signal Processor Application Programmer Interface (CSP API)
- Summary

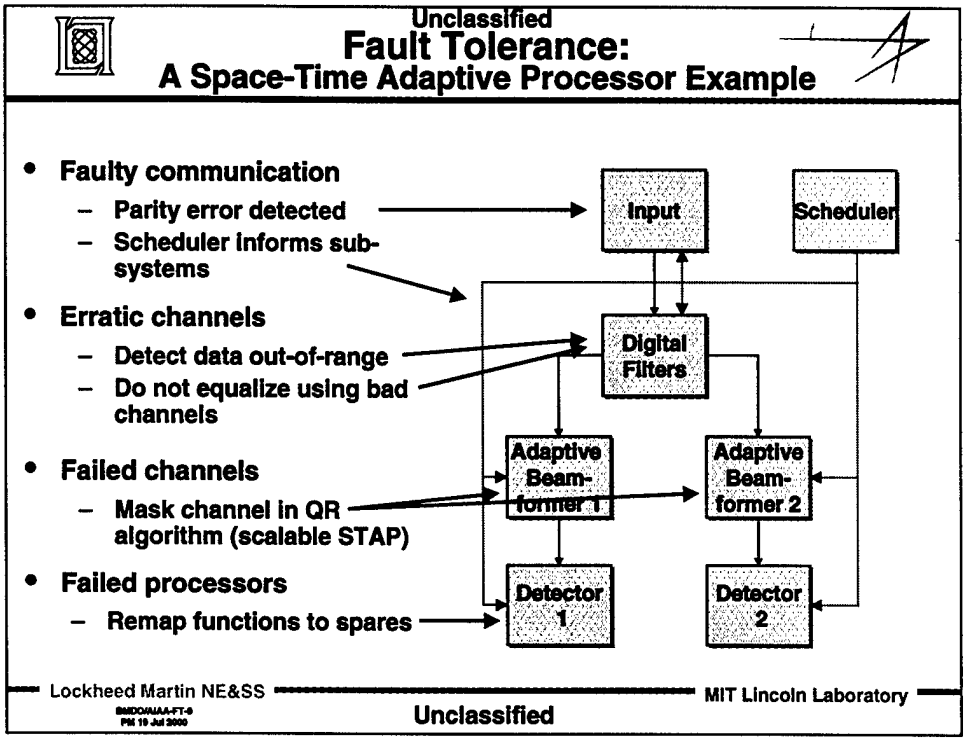


To further illustrate fault tolerance concepts, we present two examples in this section. The first example shows the potential for faults in MPP systems. The second example demonstrates the implementation of fault tolerance strategies .

Space-time adaptive processing (STAP) is a two-dimensional adaptive filtering technique that mitigates jammer and clutter interference in order to maintain required target detection performance. Analog antenna signals are digitized, downconverted to complex baseband, and passed through pulse compression filters. Adaptive spatial and doppler filter is performed and the output data is passed to detection algorithms.

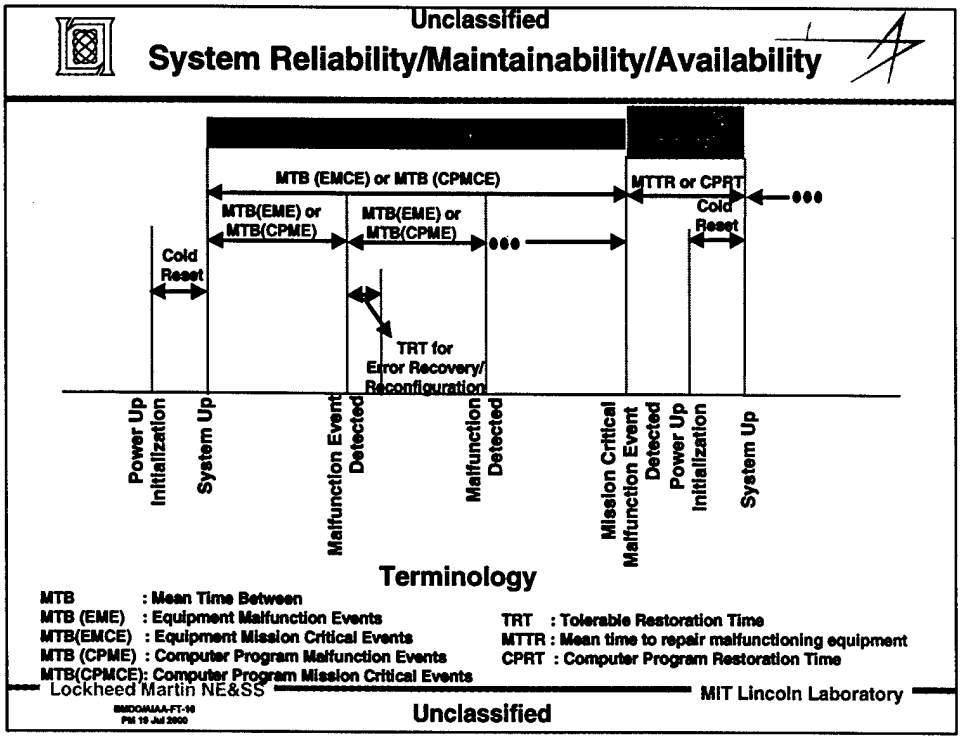
The above figure shows how an MPP system can be configured to operate on two data sets simultaneously (beamformer/detector paths 1 and 2). The scheduler assigns different tasks to the different processors and determines when the task can be executed. In addition to obvious use of task parallelism, MPP systems can be configured to operate on data sets distributed over several processors. For example, matrix addition operations lend themselves to data parallelism.

However, this architecture is succeptable to several faults including communications dropouts, failed or erratic data channels, and failed hardware.



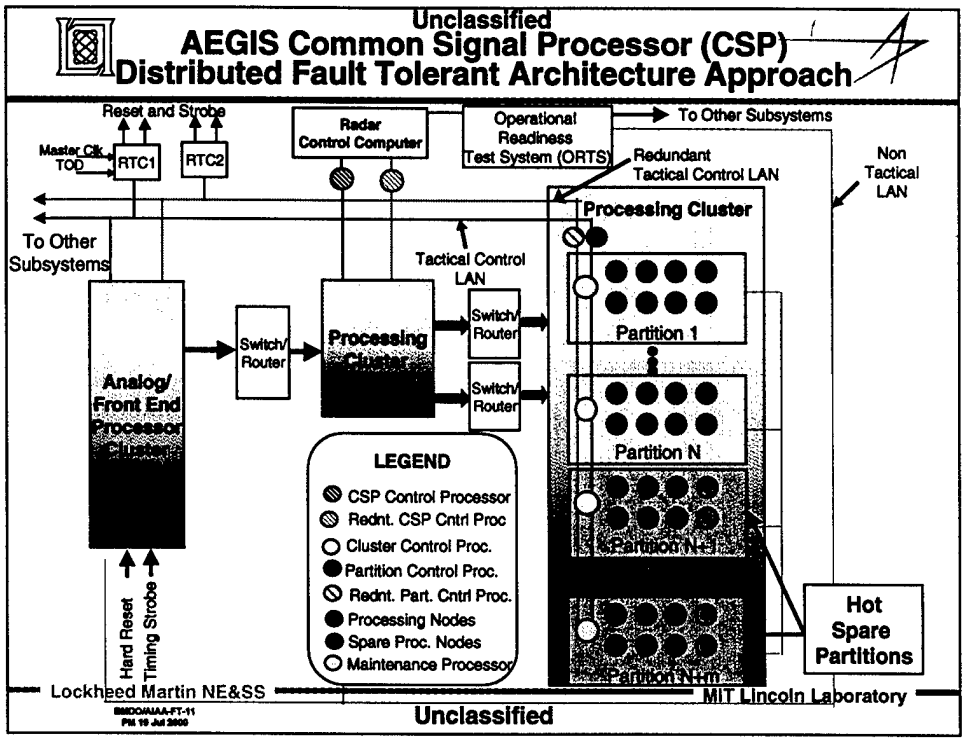
When such faults occur, there are different approaches that can be utilized to maintain system availability. First, if a communications error occurs, the scheduler will notify all subsystems of a potential problem with the data. Second, if monitoring indicates that an input data channel contains errors,

the channel can either be dropped which might result in degraded performance. Third, if an error occurs in the adaptive beamformer and insufficient computational resources are available, the processor might be reconfigured to operate on only one data set at a time. Finally, if a processor associated with detection algorithms fails, the system can remap this function to a spare processor.



This chart addresses some basic terminology associated with system reliability, maintainability, and availability concepts. The left-hand side of the chart shows a portion of the timeline where the system is available to perform some or all of its functions, i.e., it is available. Since failures are nondeterministic, statistical average measures (mean time between events) must be used to assess performance. For a combat weapons system, the average availability time is expected to be on the order of several months. The goal in any fault tolerance strategy is to maximize the average time when the system is available and minimize the average time when the system is not available. The techniques illustrated in the next several charts, when used in conjunction with MPP systems, can achieve this goal.

The example timeline itself follows the system's availability from power up through an initial failure detection and recovery. When a mission-critical unrecoverable failure occurs, the system will not be available and off-line maintenance must be performed.

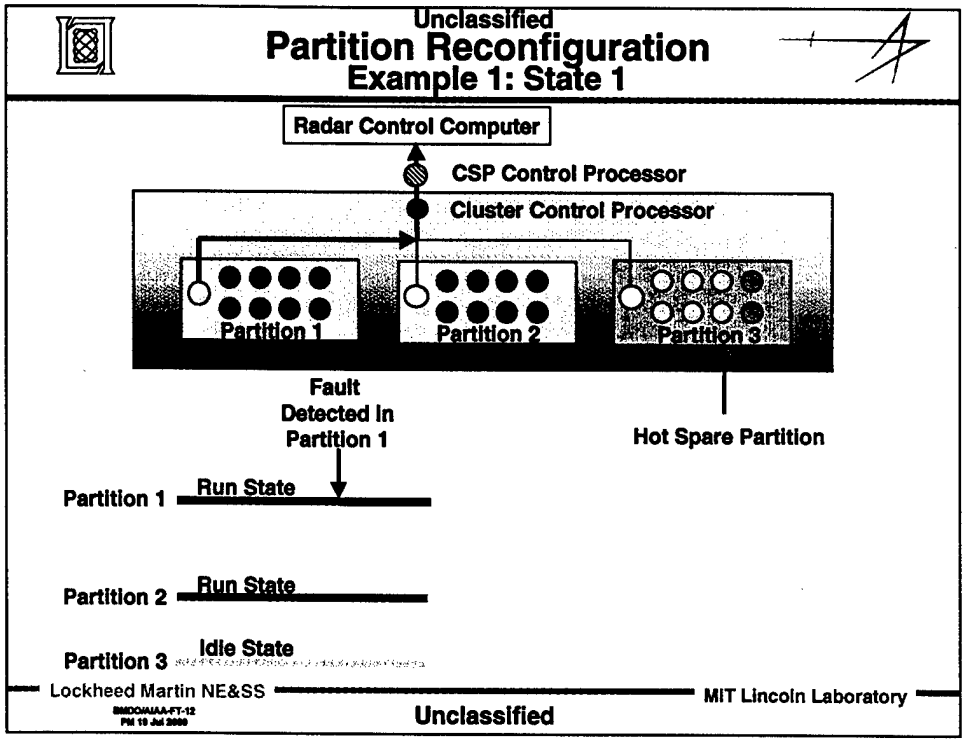


The above figure depicts the AEGIS Common Signal Processor architecture. The radar control computer oversees system, task, and data flow control processes. Analog radar data is operated on by the front-end processor and is then passed on to various processor clusters.

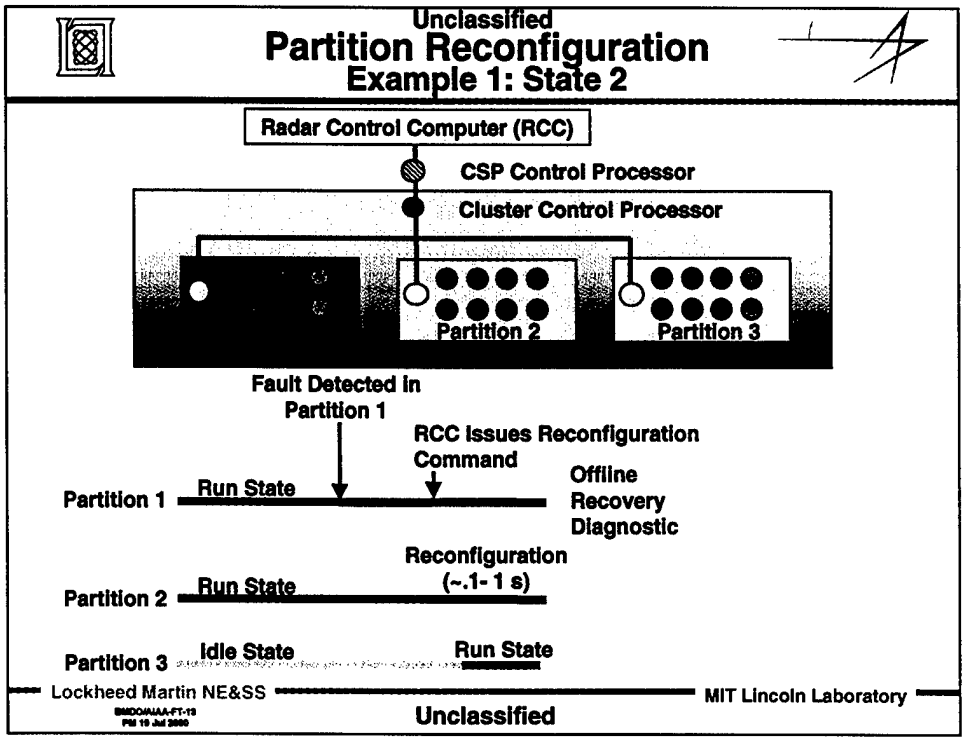
For illustrative purposes, an exploded view of one processor cluster is presented. Each processing cluster contains several partitions where some are allocated to real-time algorithm execution and others are kept in reserve in case of partition failure. Each cluster contains a control processor, processing nodes, and spare processing nodes. The control processor receives instructions

from the radar control computer.

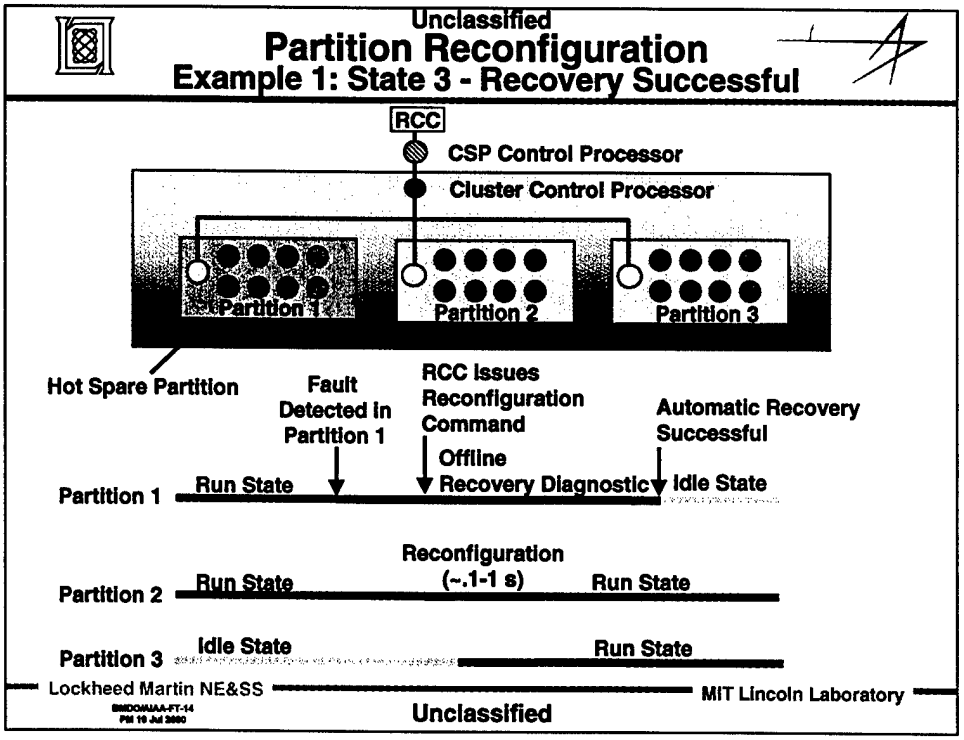
We will now proceed with several examples that demonstrate how this fault tolerance architecture can deal with certain types of faults.



We'll use a three-partition cluster example to illustrate how a cluster can have its partitions dynamically reconfigured to maintain availability despite multiple processor faults in one partition. At the beginning of run-time execution, the system has mapped computations to Partitions 1 and 2 and Partition 3 has been kept in an idle state as a hot spare. After a certain time, faults in two Partition 1 processor elements are detected. The system allocates some time to determine if Partition 1 can recover from the fault. This corresponds to the red section associated with Partition 1timeline.

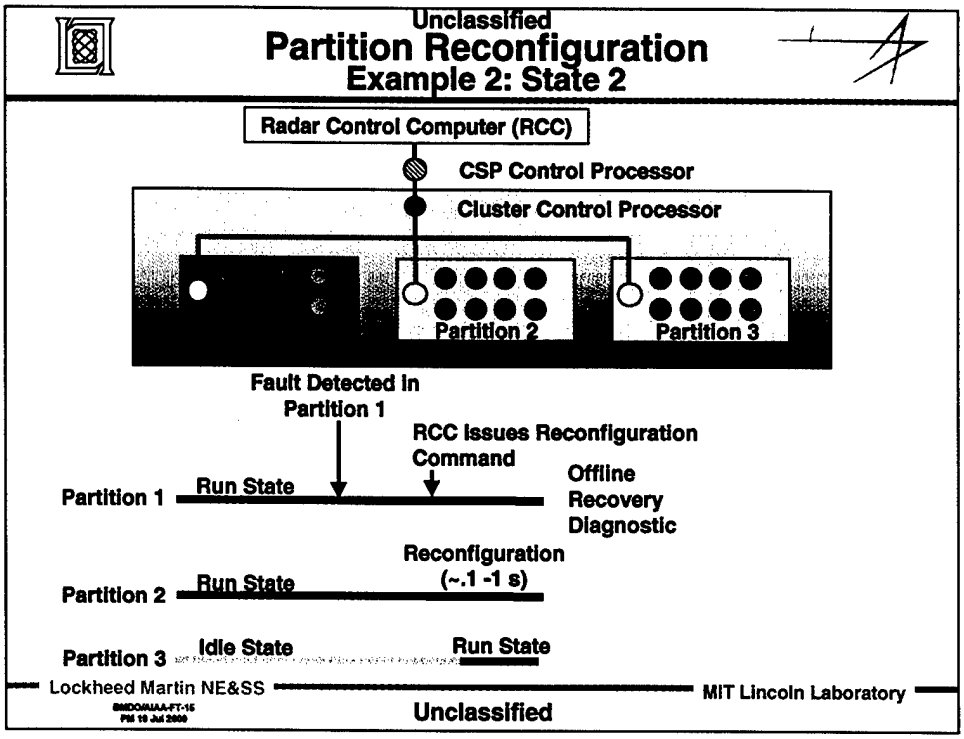


In the next part of this example, we'll assume that Partition 1 will go into a fault recovery mode. The radar control computer must now act to reconfigure the cluster system to maintain availability. The appropriate commands are sent to and through the Common Signal and Cluster control processor. Within a time frame ranging from 0.1 to 1.0 seconds, these commands will be executed so that the cluster will be reconfigured to perform the computations on Partitions 2 and 3. Reconfiguration commands include awakening Partition 3 from its idle state, reinitializing Partitions 2and 3 to operate on the appropriate data, and execute offline diagnostic tests on Partition 1 to ascertain if the fault is or is not recoverable.

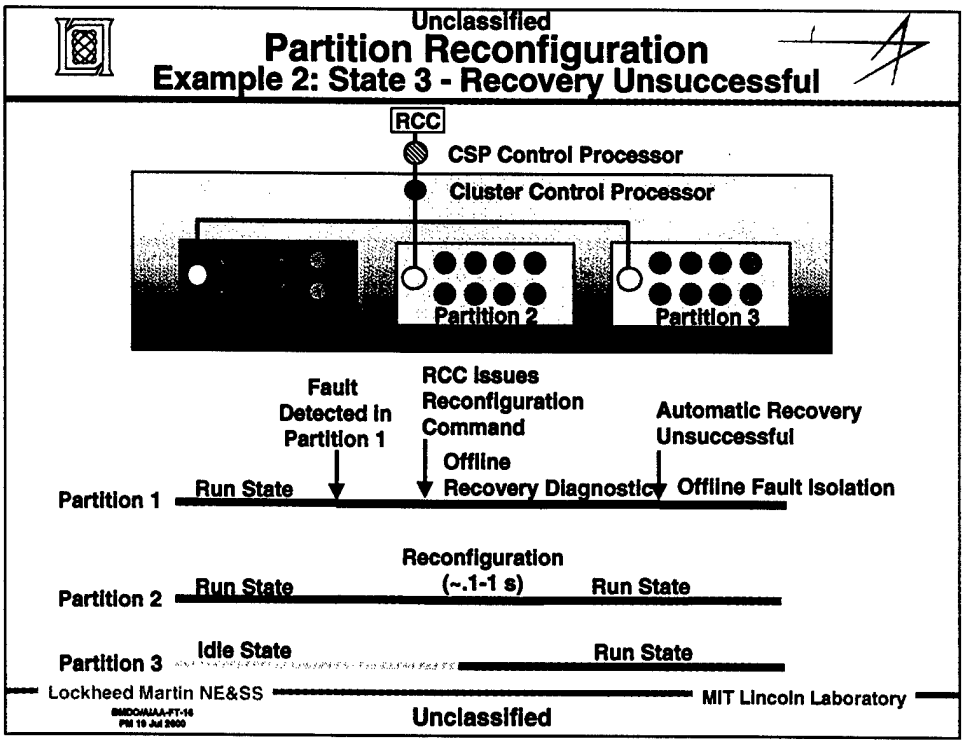


We continue with this example by assuming that the offline diagnostics indicate that the fault is recoverable, commands are given to correct and test this partition, and the control processors are notified that the partition is available again, this time as a hot spare. As indicated by the upper time line, Partition 1 is now in the idle. The system has now recovered from the fault with minimal down time and resumes its fault monitoring process. Thus, an overall high availability rate can be maintained.

As will be discussed in the next section, high-level software commands in the Application Programmer Interface (API) will contain the necessary functionality to automatically deal with fault conditions such as one described in this example. Embedding fault detection/isolation/recovery functionality in high-level software will simply the programming task by hiding some degree of complexity and also result in a reduction in the number of lines of code (hence, cost).

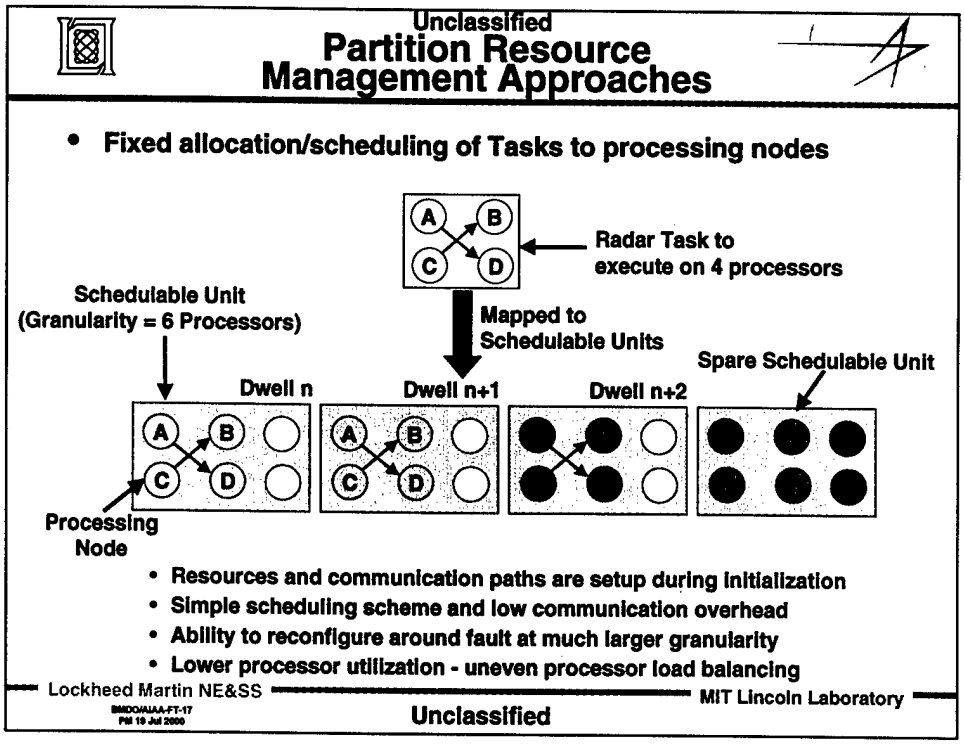


Let's return to the state where the partition reconfiguration commands have been given by the control processors and Partition 2 has been placed in the run state.



Now let's assume that an unrecoverable fault has occurred. The failed partition would then be subjected to offline testing in order to isolate the fault. One possible action would be to hot swap in another board which would then be result in a new cluster configuration with it as the hot spare. If another unrecoverable fault occurs before a partition replacement or repair, the system would have to accept either degraded performance or shut down the entire cluster.

Through these examples, one can clearly see the benefits of software control of fault tolerance for multiprocessor systems. The amount of processor redundancy would be principally determined by availability requirements, processor system size, weight, and power constraints, and system cost.

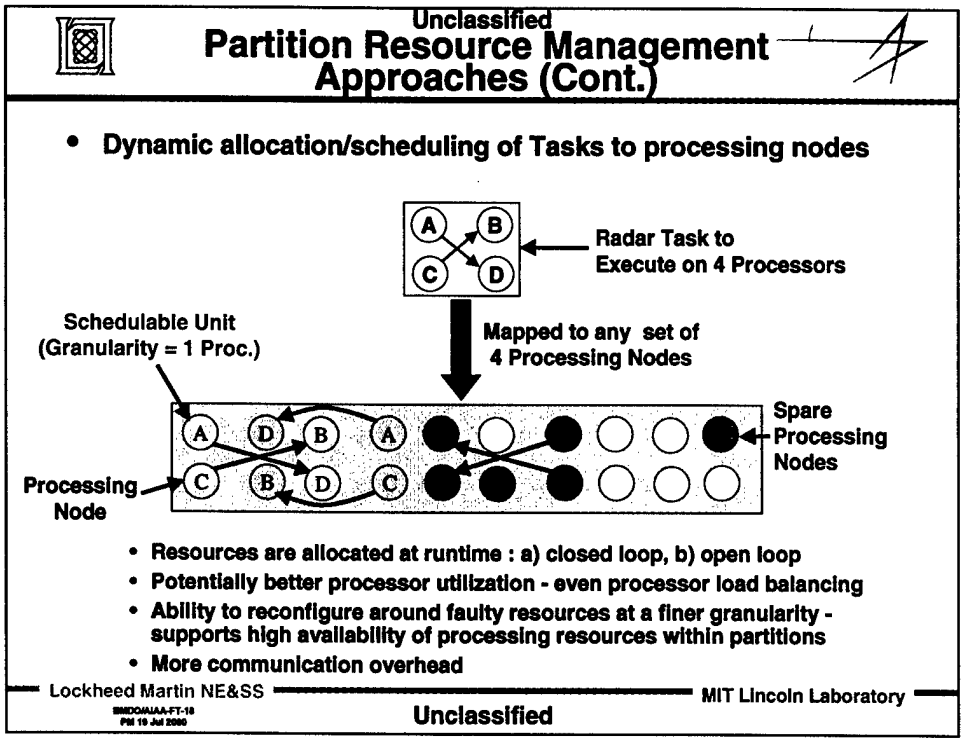


In the previous examples, we looked at a higher level of fault tolerance processing which operated at the partition level, i.e., partitions were monitored and recovery procedures were confined to partition functionality. However, our multiprocessor partition architecture can provide a finer granularity of fault detection, isolation, and recovery.

Let's first assume that we have a radar processing task that requires a minimum of four processor elements (denoted A, B, C, and D) and well-defined data flows flow as indicated by the arrows. The data and required processing flow for each radar dwell is mapped to four of six available processors in each partition as illustrated in the above figure. A spare 6-processor partition is kept available for reconfiguration.

This fixed resource allocation and task scheduling process enables processing resources and communications paths to be setup during initialization, simplifies scheduling and minimizes overhead, and allows fault reconfiguration at high granularity (which requires simpler software control).

However, we are not maximizing the efficiency of this processor architecture by keeping a rigid task schedule and processor resource allocation (many processors can be idle.

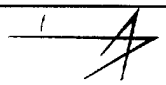


If we reduce the granularity to single processor node, we can achieve the ultimate in resource allocation and reconfiguration. In the above figure, we allocate the resources at task run time. The data flow is potentially quite different from a fixed initial resource allocation case. Note that mapping and data flow for the three radar dwells. Much better processor utilization is expected with this very low level of granularity along with greater flexibility to reconfigure around a fault. Spare processing nodes as opposed to partitions can be brought in when a fault occurs.

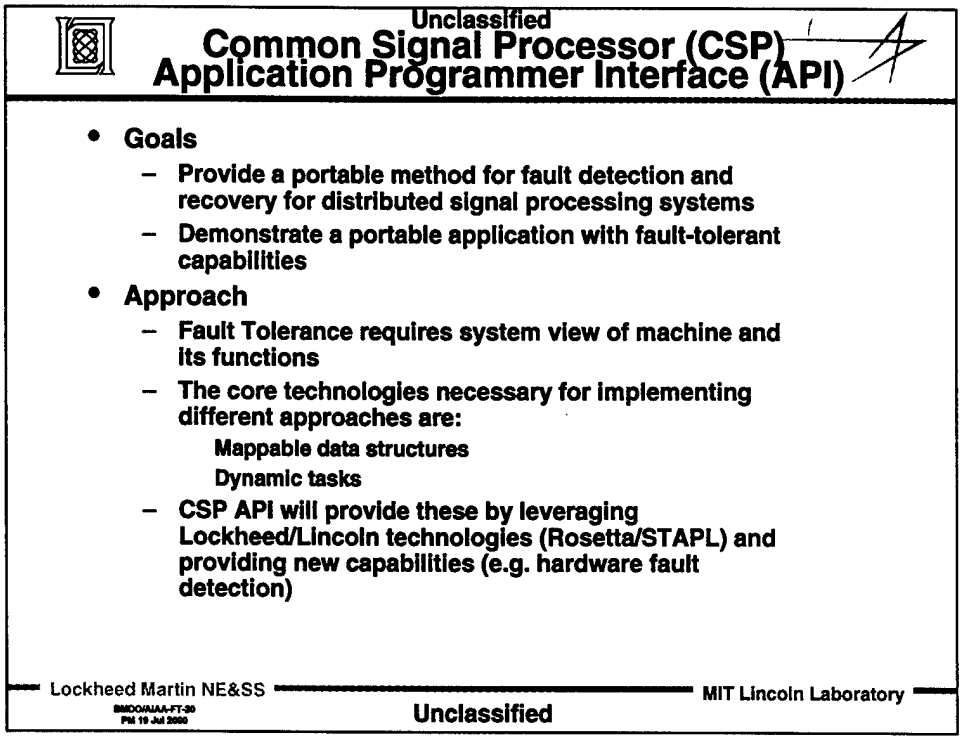
The increased fault recovery and resource allocation capabilities do require additional communications overhead and increased software control complexity. Both partition management schemes addressed in this paper will be evaluated as part of the joint Lockheed Martin/Lincoln Laboratory development program.

Unclassified

Outline

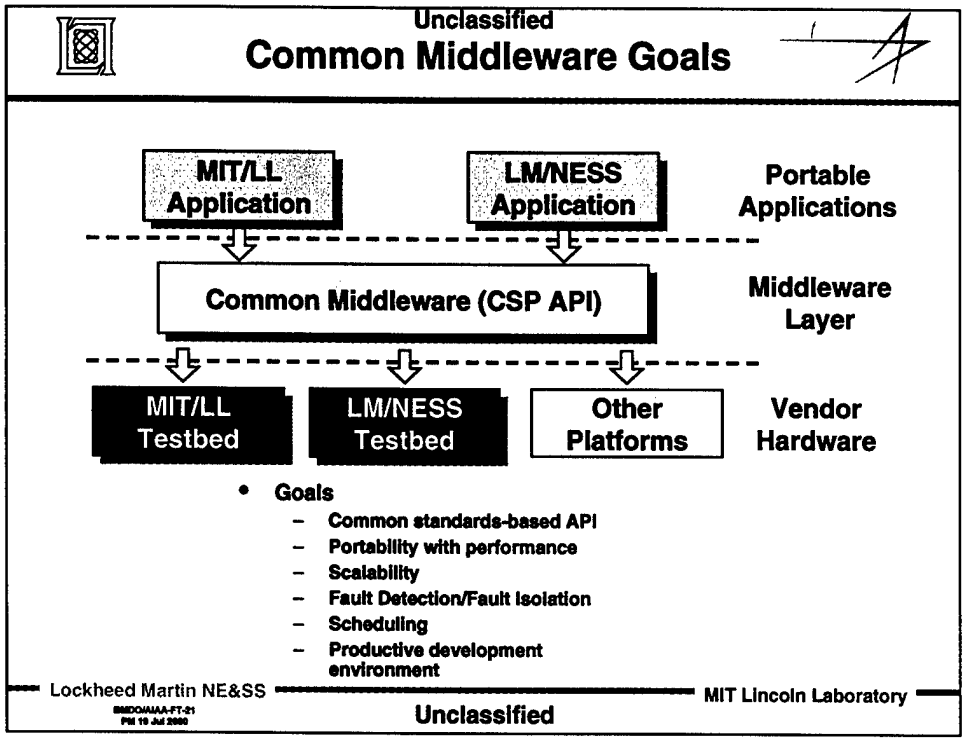


- Introduction
- Fault tolerance concepts and examples
- Common Signal Processor Application Programmer Interface (CSP API)
- Summary



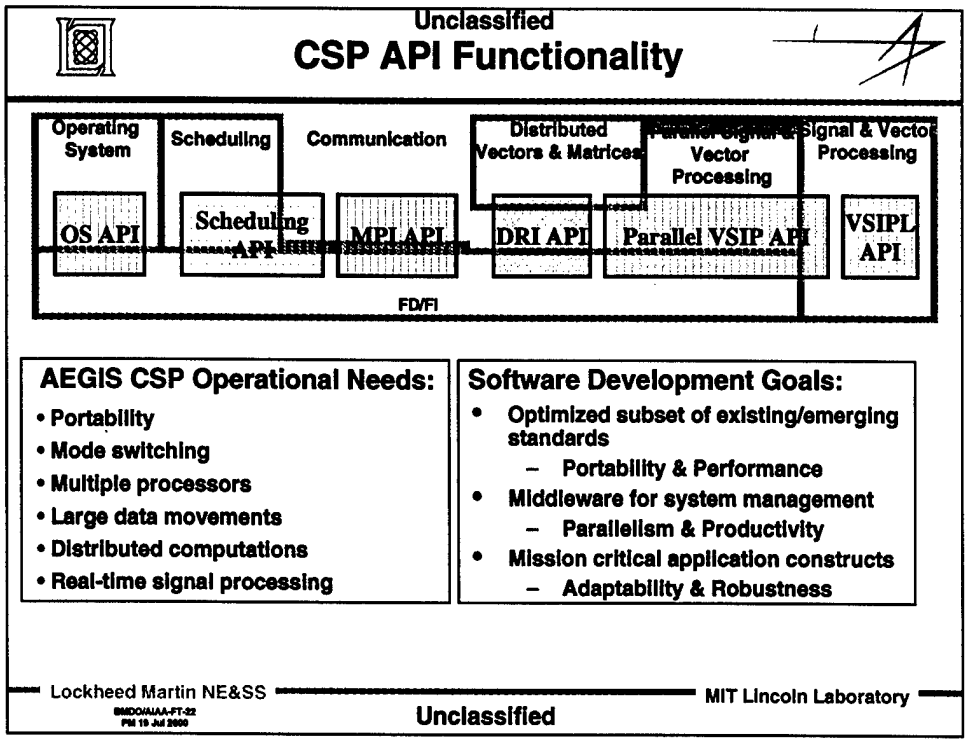
The Application Programmer Interface (API) provides a portable, high-level software means to address task execution, resource allocation, and fault detection. Portability is important because due to the rapid pace of computational processing performance improvements along with the long life cycles of military systems, software and hardware architectures can no longer afford to be point solutions. In addition, portability implies that the API can be moved from platform to platform with minimal changes thus serving a wide range of applications (e.g., the same API can be used for a ground-based radar as well as SPY).

We approaching API development by leveraging previous successful efforts in this area. New technology will be developed to deal with dynamic tasks and mappable data structures. Dynamic tasking was addressed in the previous section. Proper mapping of data structures such as vectors and matrices (common in multichannel radar signal processing systems) will need to be explored in order to determine how to best achieve maximum efficiency and reconfigurability.



To provide a portable, fault tolerant capable API, we plan to develop common middleware. Let's first describe the applications which sit at the highest level. The applications call functions found in the middleware just as one would call BLAS or LAPACK routines if writing Cor Fortran programs. The middleware then executes machine-specific code that has been optimized for a particular hardware architecture. Thus, at the highest level, the programmer is isolated from the machine-specific code which results in a much higher degree of code portability.

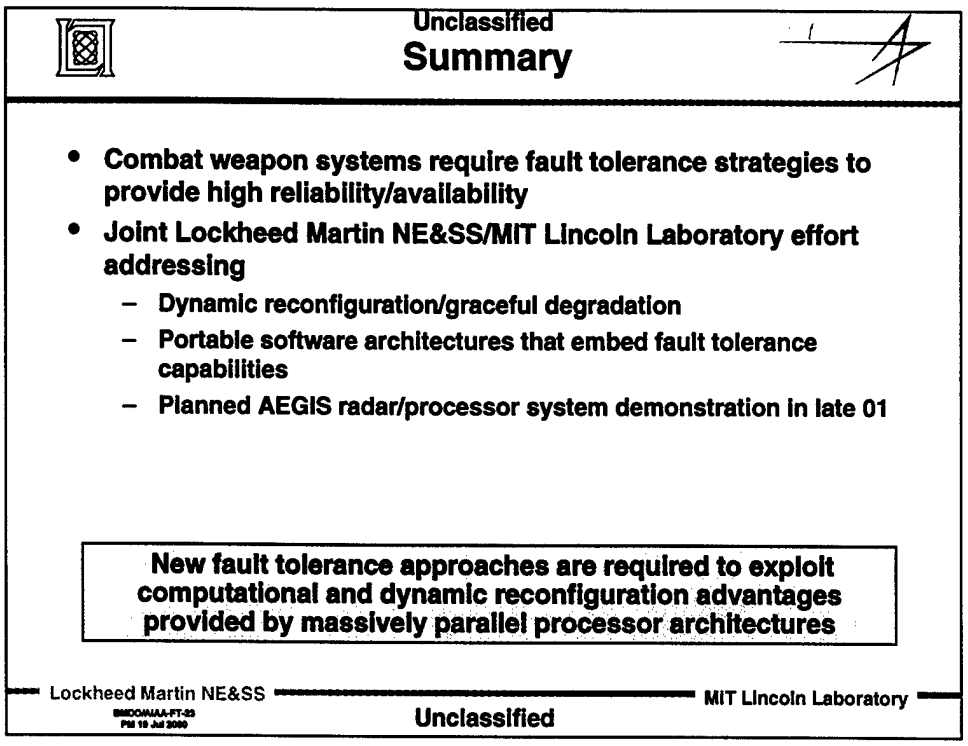
The principal common middleware development goals are: 1) to use common standards to promote portability, 2) ensure that performance is maintained, 3) incorporate fault detection, isolation, and recovery capabilities, 4) incorporate fixed and dynamic scheduling, and 5) create a highly productive code development environment.



The above chart depicts the functionality that will be included in the API. These functions range from operating system to signal processing. Both the Message Passing Interface (MPI) and Vector, Signal,and Image Processing Libraries (VSIPL) are vendor-supported standards. Parallel VSIPL is currently under development and will most likely become a standard in the near future.

In particular, the AEGIS CSP API requires portability to support life-cycle maintenance, multiple processors for high throughput, low latency operation, the ability to move large data blocks to perform range and doppler processing, and distributed computation capability in order to execute complex real-time signal processing algorithms.

This API development effort will aim to provide portability, performance, improved productivity, and robust operation. The resulting software architecture will enable highly reliable and available multiprocessor systems capable of meeting military signal processing challenges .



The examples in this paper have shown that new fault tolerance approaches are necessary to support the advanced capabilities provided by Massively Parallel Processors. These approaches can leverage the inherent processor redundancy and reconfiguration flexibility provided by MPPs to achieve very high system availability levels which are necessary for combat weapons systems. The API will be designed to provide performance, portability, and productivity. Lincoln Laboratory and Lockheed Martin will be jointly developing these approaches over the next year as part of an AEGIS radar processor risk-mitigation program.