



KASSPER

Real-Time Embedded Signal Processor Testbed

**Glenn Schrader
Andrew Heckerling
Michael Harrison**

**Massachusetts Institute of Technology
Lincoln Laboratory**

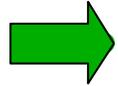
14-16 April 2003

This work is sponsored by the Defense Advanced Research Projects Agency, under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

MIT Lincoln Laboratory



Outline

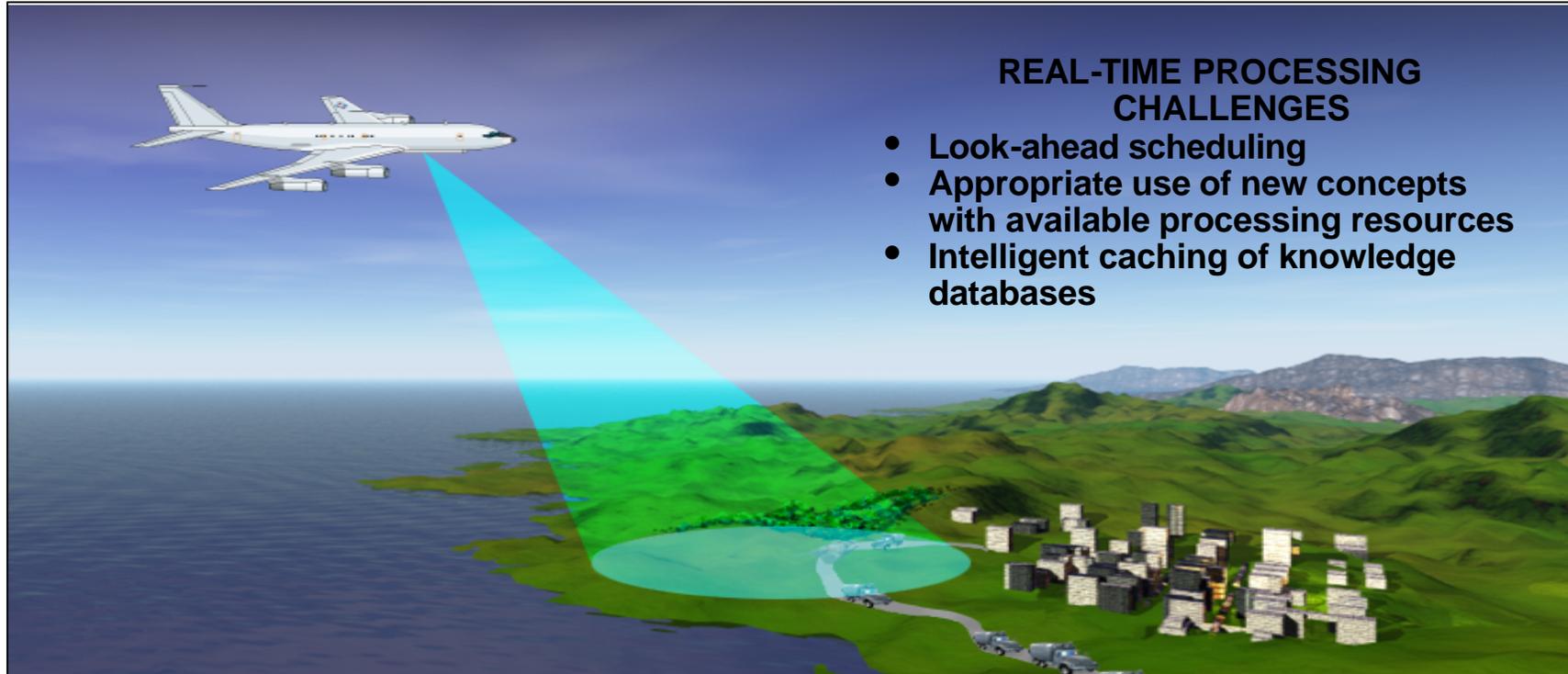


KASSPER Testbed

- **Overview**
- **Baseline Algorithms**
- **High Level Processor Scheduling**
- **Processor Requirements**
- **Application Architecture**
- **Processor Environment**
- **Parallel Vector Library**
- **Summary**



Lincoln Laboratory Real-time Testbed Development



REAL-TIME PROCESSING CHALLENGES

- Look-ahead scheduling
- Appropriate use of new concepts with available processing resources
- Intelligent caching of knowledge databases

FY '02:
Implemented
Baseline GMTI
Processing Chain,
Procured Processor,
Ported the Parallel
Vector Library (PVL)



**FY '03: Explore Architecture and
Algorithm Concepts**

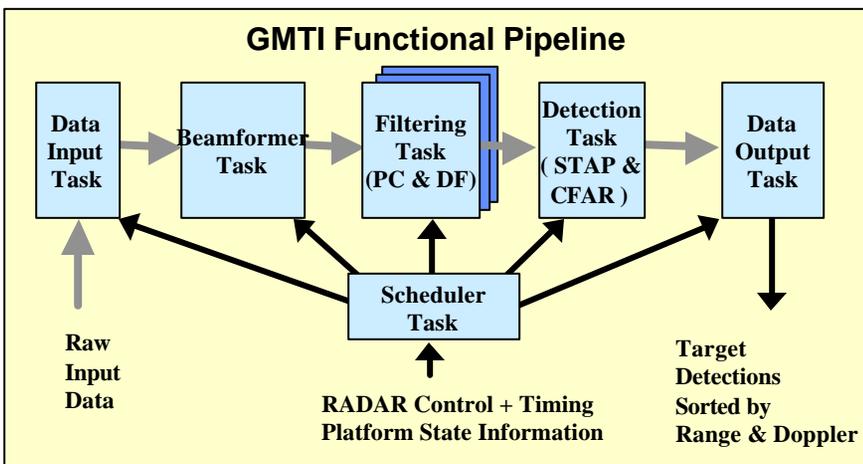
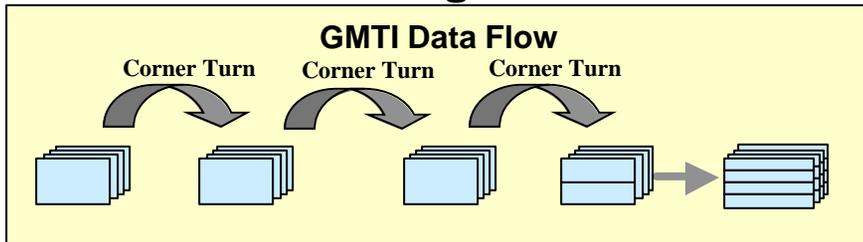
- Baseline SAR processing
- Multi-mode operation
- High speed input data



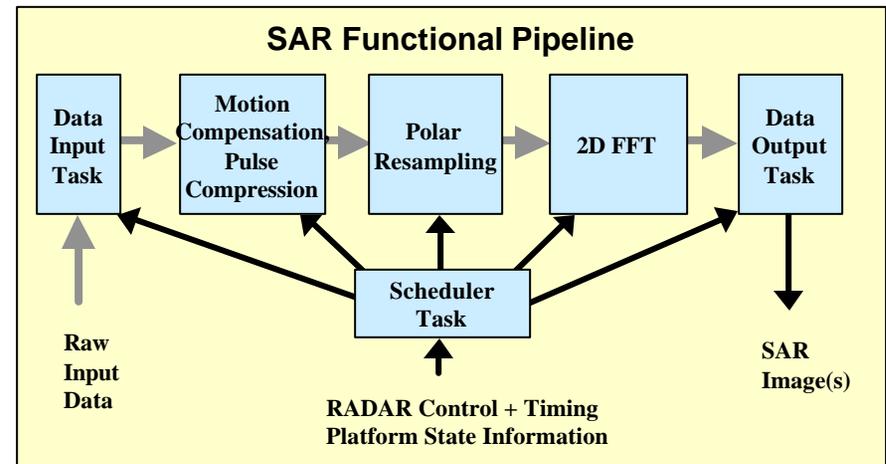
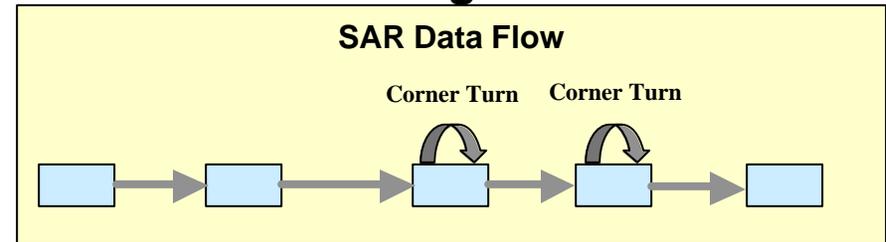
Baseline Processing Algorithms



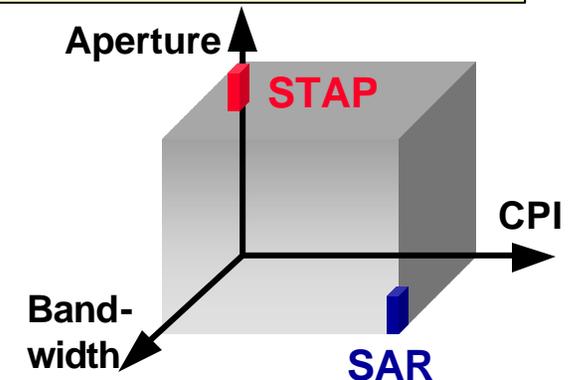
GMTI Algorithm



SAR Algorithm



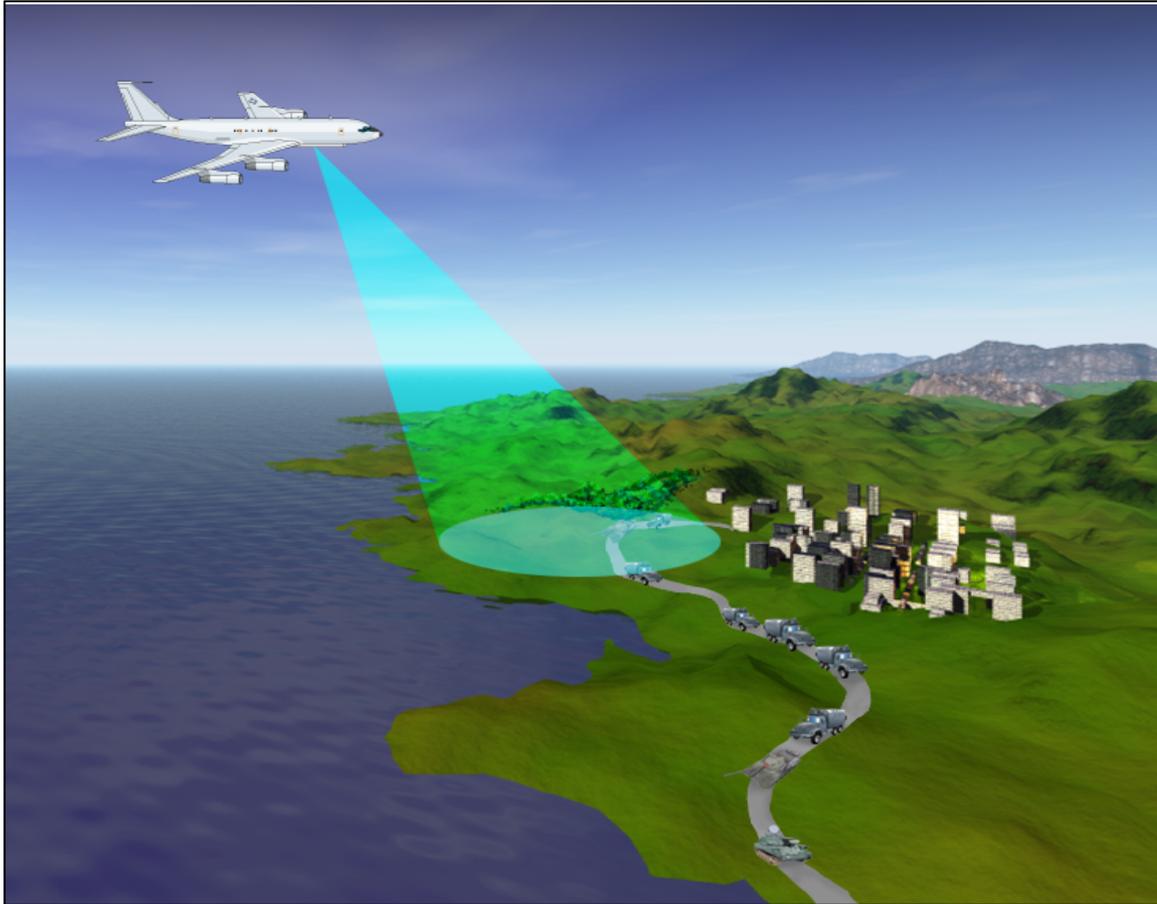
- Well understood algorithms
- Provides a starting point for exploring processing concepts
- Variants which include a-priori knowledge can be included as algorithms are defined



MIT Lincoln Laboratory



High Level Processor Scheduling



Schedule dwells to obtain the maximum information from the environment using the fewest system resources.

- **Based on the mission flight path and information gathering goals, predictively schedule the time, look angle and range extent to process for future dwells based on the predicted platform location and optimum viewing directions.**
- **If a dwell needs additional a-priori knowledge then schedule dwells to probe the environment (I.e. SAR, etc) to update the knowledge database.**
- **Schedule pre-loading for the a-priori data needed by each dwell prior the the dwell's start time.**



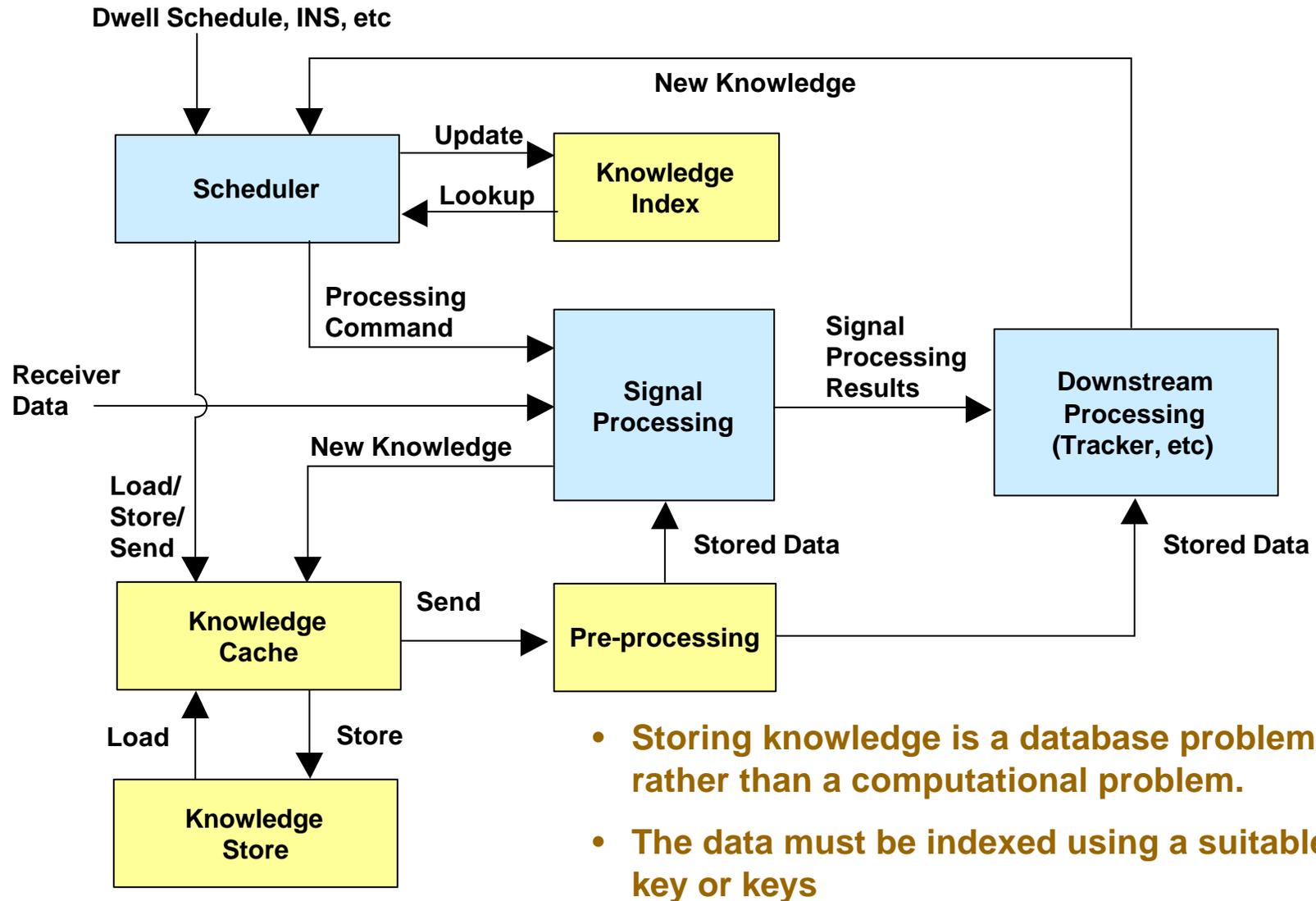
Processor Architecture Requirements



- **Allow incorporation of a-priori (i.e. stored) knowledge into the processing chain**
 - **Allow stored knowledge to be updated in real-time**
 - **Conceptually allow stored knowledge to be anything. I.e. raw receiver data (I/Q input data), intermediate processing results (covariance), complete processing results (detection lists, SAR images), downstream processing results (tracker output), knowledge collected offline (DTED, road location, terrain boundaries, etc).**
- **Support real-time switching between multiple radar modes/algorithms (i.e. CPI length, a-priori knowledge use, range extent to process, etc).**

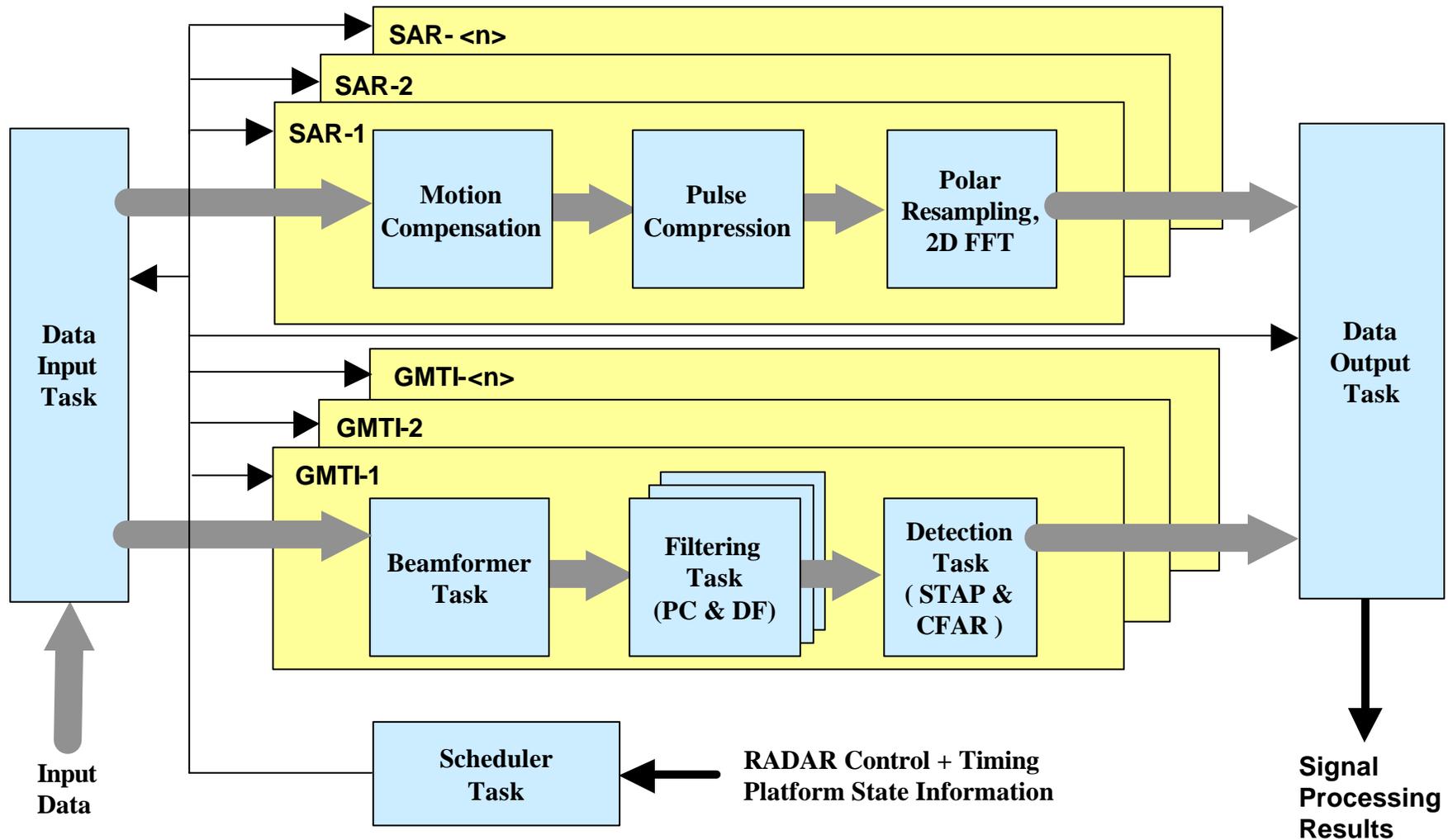


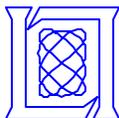
Preliminary Knowledge Database Architecture



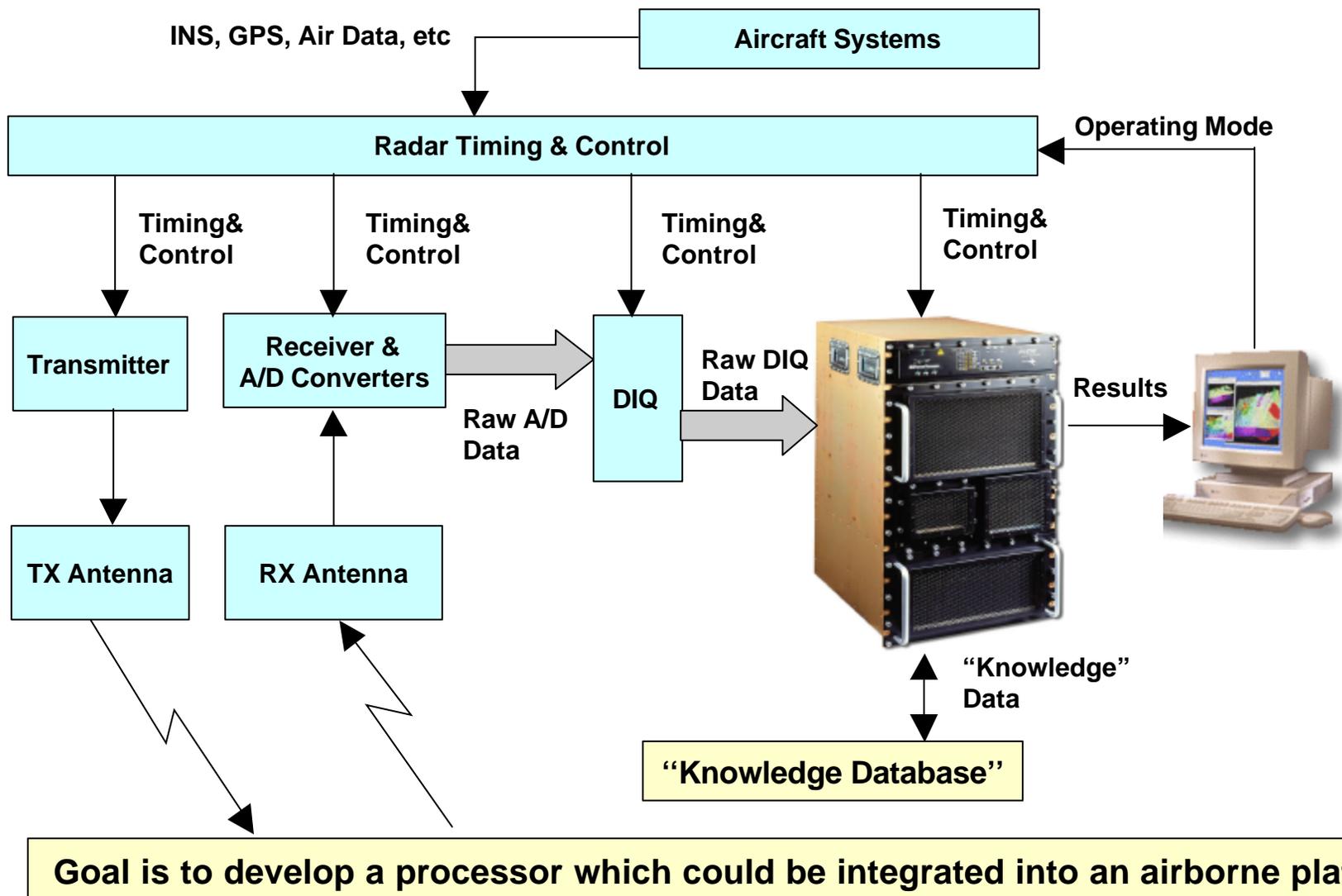


Multi-Mode Application Architecture



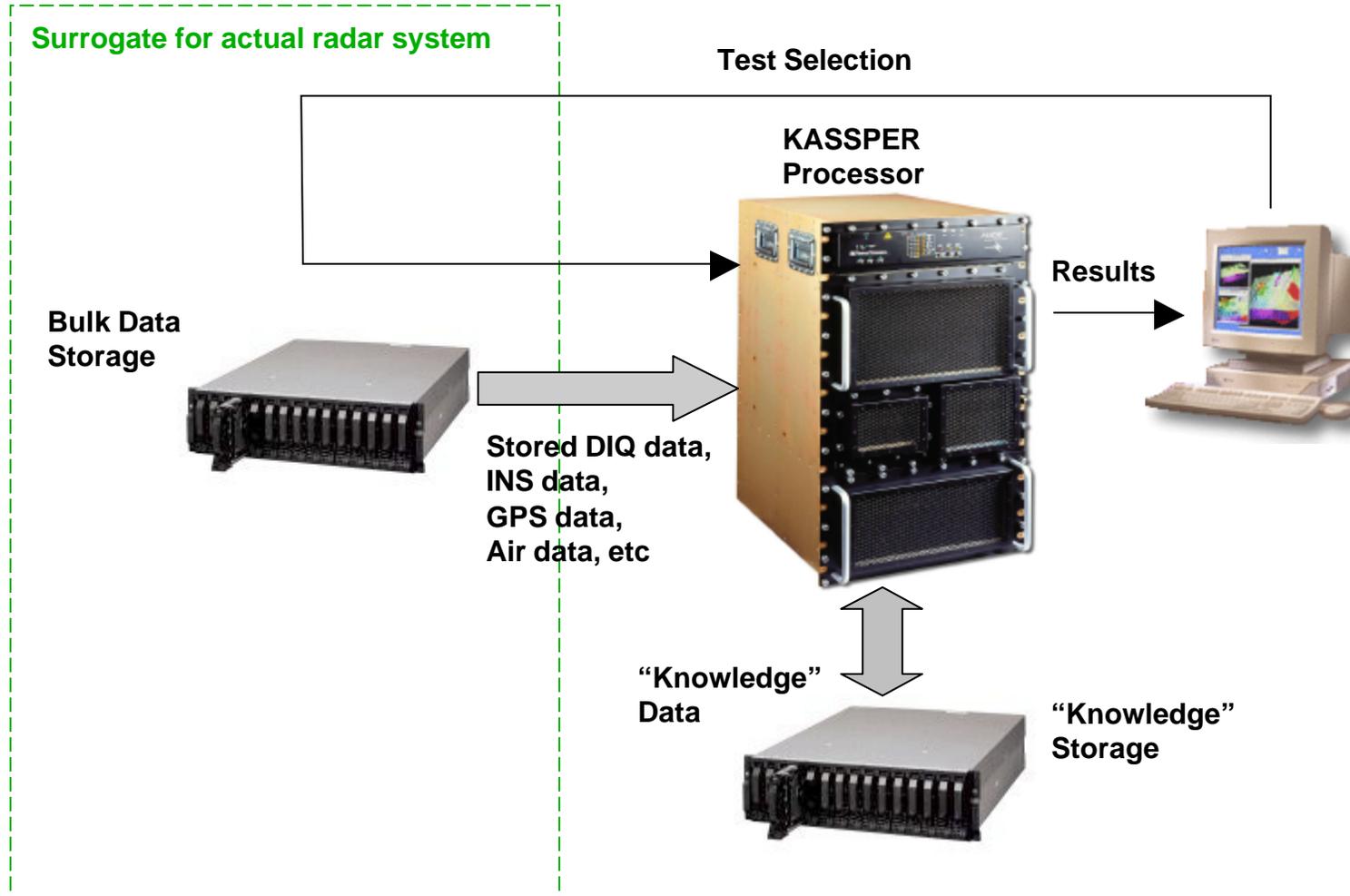


Integrated Airborne Processor Environment



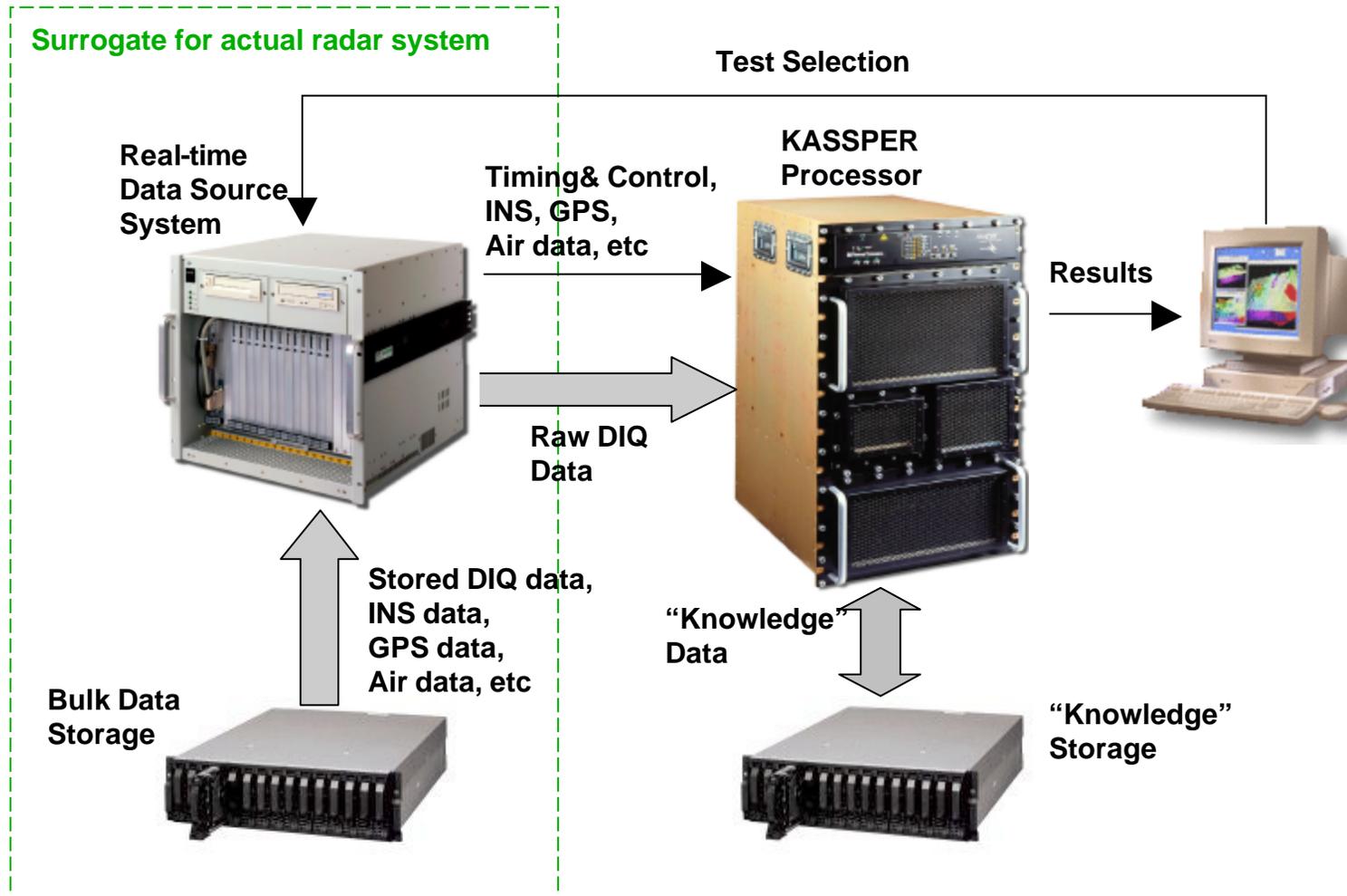


Laboratory Testbed Processor Environment



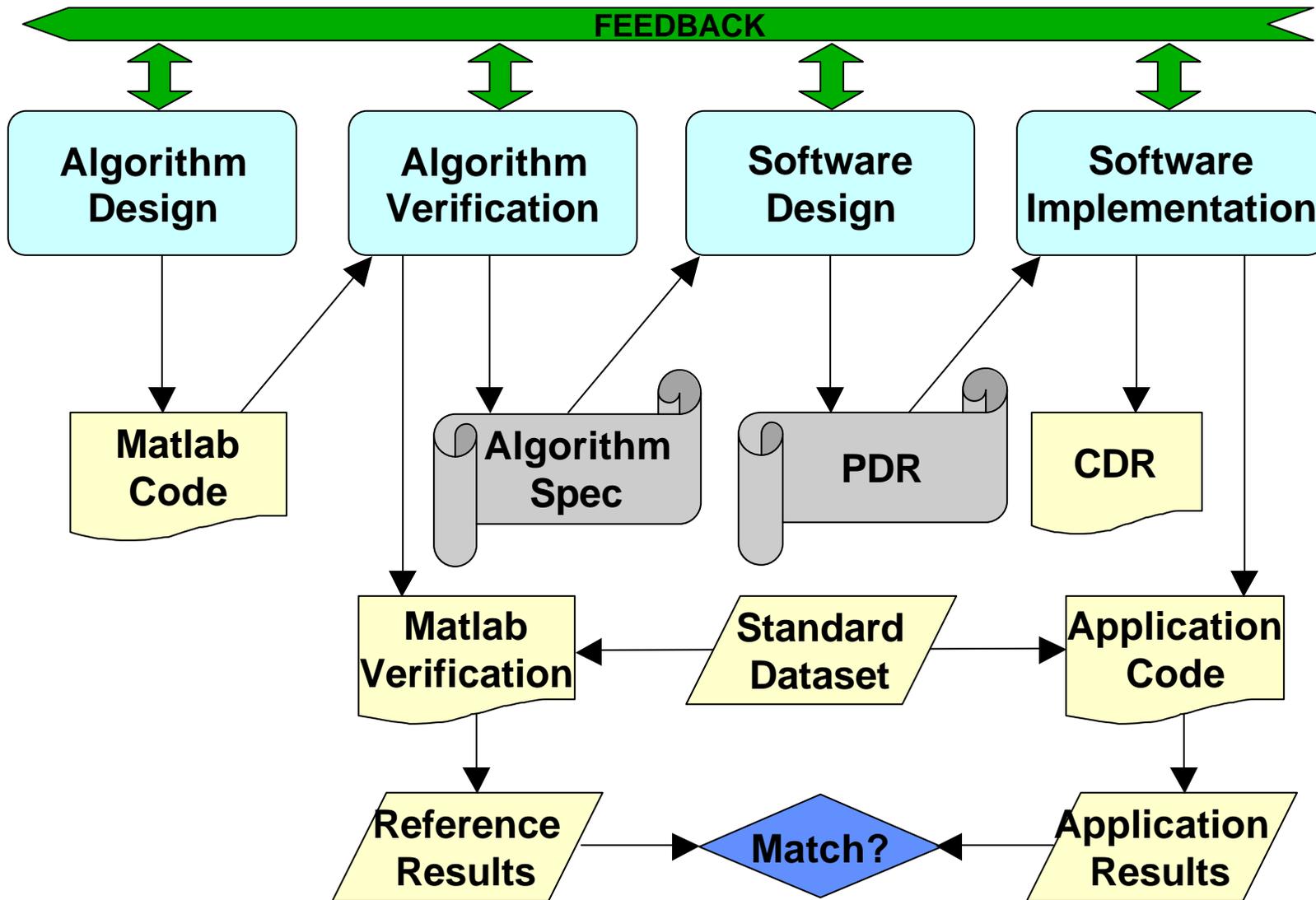


Laboratory Testbed Processor Environment





Algorithm Implementation Process

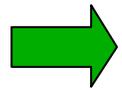




Outline



- **KASSPER Testbed**



Parallel Vector Library (PVL)

- **Introduction**
- **Basic PVL Concepts**
- **Application Structure**
- **Abstraction with Performance**
- **Future Directions**
- **Summary**



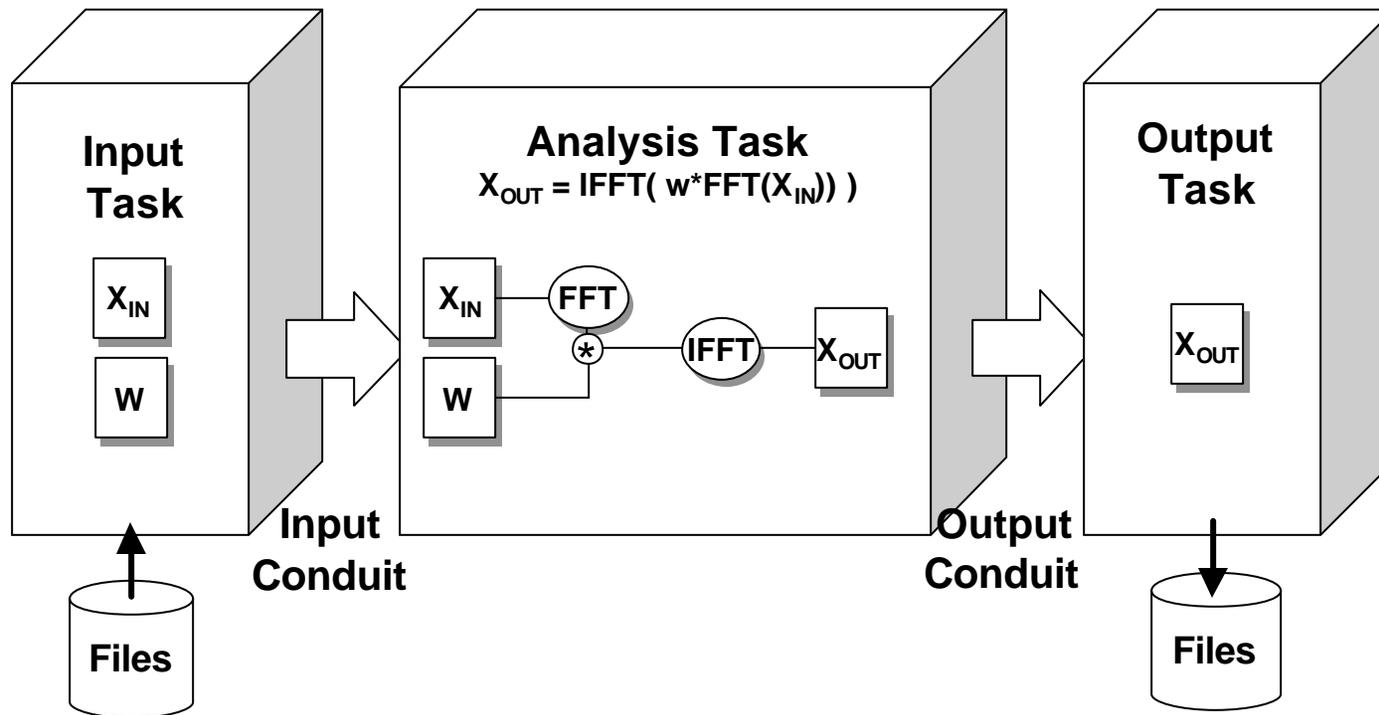
The Motivation For PVL



- **Developing high performance parallel signal processing software is difficult.**
- **Making parallel software portable and scalable adds additional complexity.**
- **Rapid Prototyping is more difficult when developing directly on a real-time processor.**



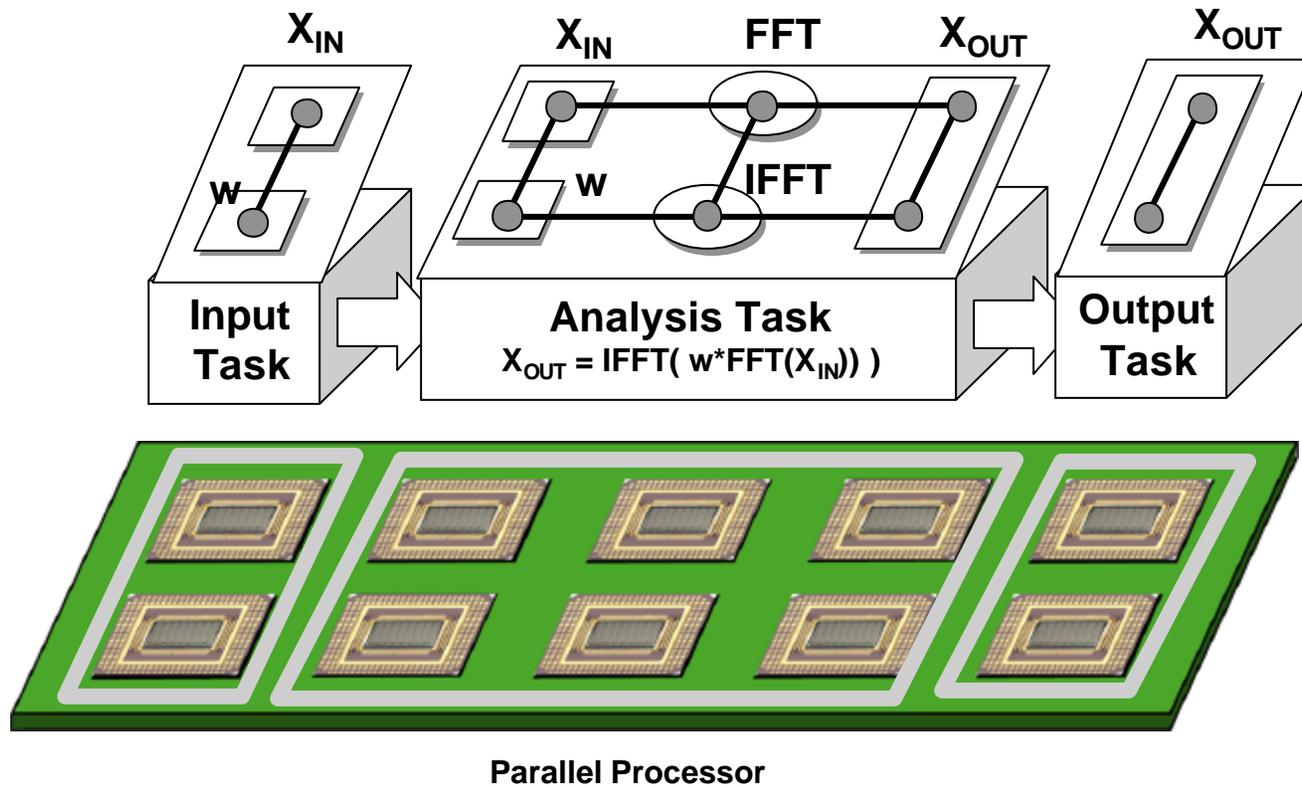
Algorithm Software Development: Basic System



- Can build real pipelined systems with PVL tasks and conduits (comm)



Algorithm Software Development: PVL Map/Grid Concept



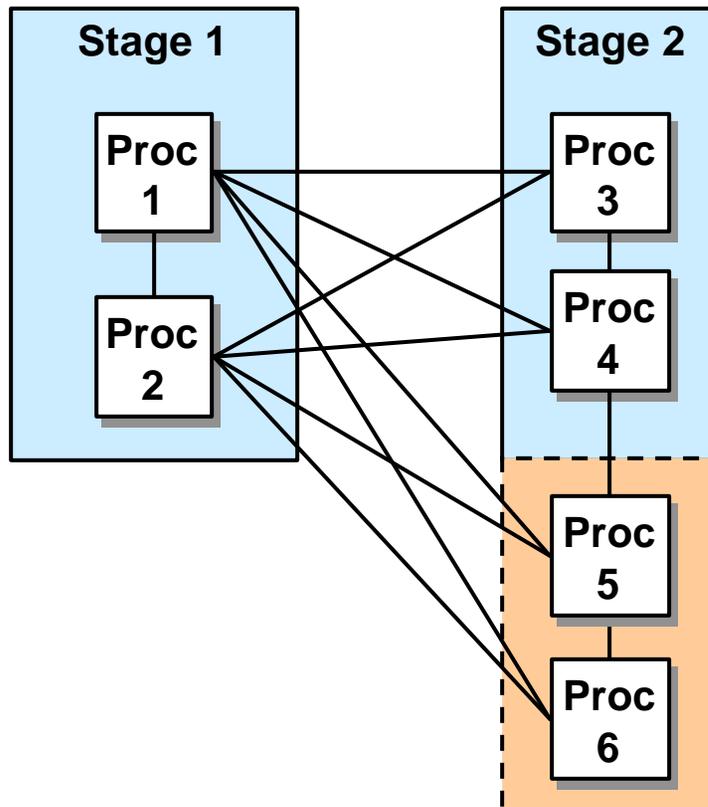
- Vectors, Matrices, Computations, and Tasks can be mapped to different grids of processors



Current Non-PVL Approach to Parallel Code Development and Mapping



Algorithm + Mapping



Code

```
while(!done)
{
    if ( procNum()==1 || procNum()==2 )
        stage1 ();
    else if ( procNum()==3 || procNum()==4 )
        stage2();
}
```

```
while(!done)
{
    if ( procNum()==1 || procNum()==2 )
        stage1();
    else if ( procNum()==3 || procNum()==4 ||
             procNum()==5 || procNum()==6 )
        stage2();
}
```

- Algorithm and hardware mapping are linked
- Resulting code is non-scalable and non-portable



Key Insights Into Parallel Software Development



- **Algorithm Implementation and Mapping to the Processor should be as orthogonal as possible.**
- **Computation and communication should be seamless from the application's point of view.**
- **~90% of the work comes from ~10% of the code so make the 90% that isn't performance critical as easy to develop as possible.**
- **Allow access to the underlying data so that application specific processing functions can still be implemented.**
- **Rapid prototyping is easier if application code is portable between the workstation and real-time environments.**



PVL Application Code Development



Separate the job of writing a parallel application from the job of assigning hardware to that application

“Application Developer”

- Converts algorithm into code

```
while( !done )  
{  
    task1();  
    task2();  
}
```

- Writes code once
- Easier to code, because only concerned with mathematics, not distribution

“Mapper”

- Maps code to hardware
- Creates new mappings when code is scaled or ported

Task 1

Proc 0

Proc 1

Task 2

Proc 2

Proc 3

Creating ‘good’ mappings requires expert developers and is not automated.



Application code example



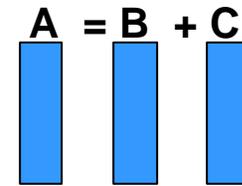
```
#include <Vector.h>
#include <AddPvl.h>

void addVectors(aMap, bMap, cMap) {
  Vector< Complex<Float> > a('a', aMap, LENGTH);
  Vector< Complex<Float> > b('b', bMap, LENGTH);
  Vector< Complex<Float> > c('c', cMap, LENGTH);

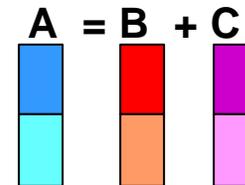
  a = 0; b = 1; c = 2;

  a=b+c;
}
```

Single Processor Mapping



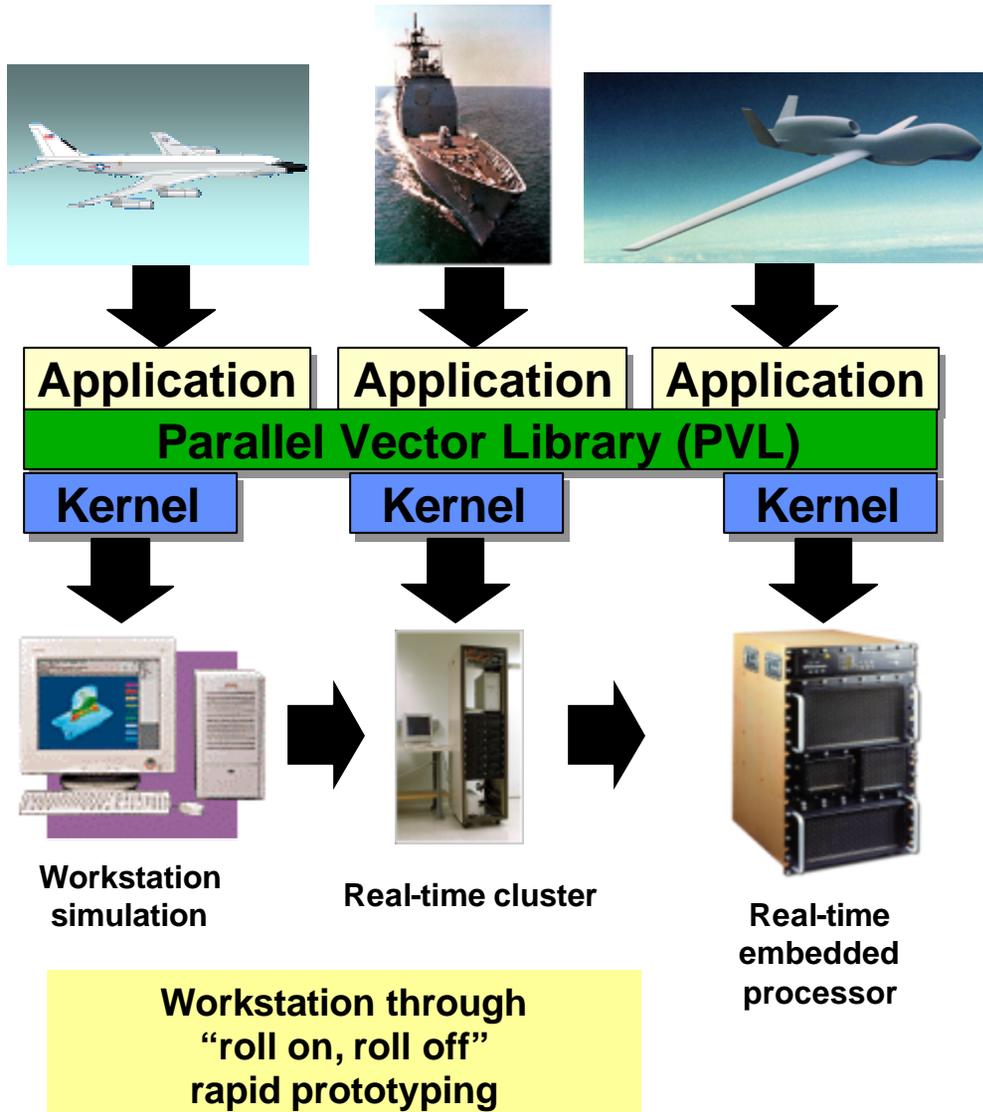
Multi Processor Mapping



- Single processor and multi-processor code are the *same*
- *Maps* can be changed without changing software
- High level code is compact



Parallel Vector Library (PVL)



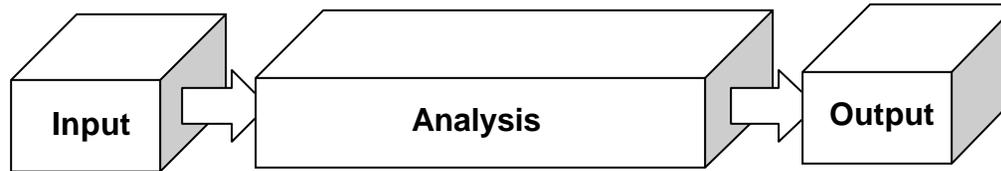
Measured PVL Portability for STANDARD Missile Application		
		Portability
Application	Machine independence	99%
Library	Optimized parallel library	97%
Kernel	Machine dependence	74%



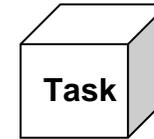
PVL Layered Architecture



Application



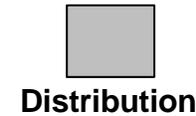
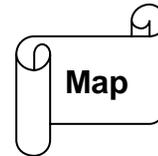
Productivity



User Interface

Parallel Vector Library

Performance



Portability



Hardware Interface

Hardware



Workstation



Intel Cluster

PowerPC Cluster



Embedded Board



Embedded Multi-computer

- Layers enable simple interfaces between the application, the library, and the hardware

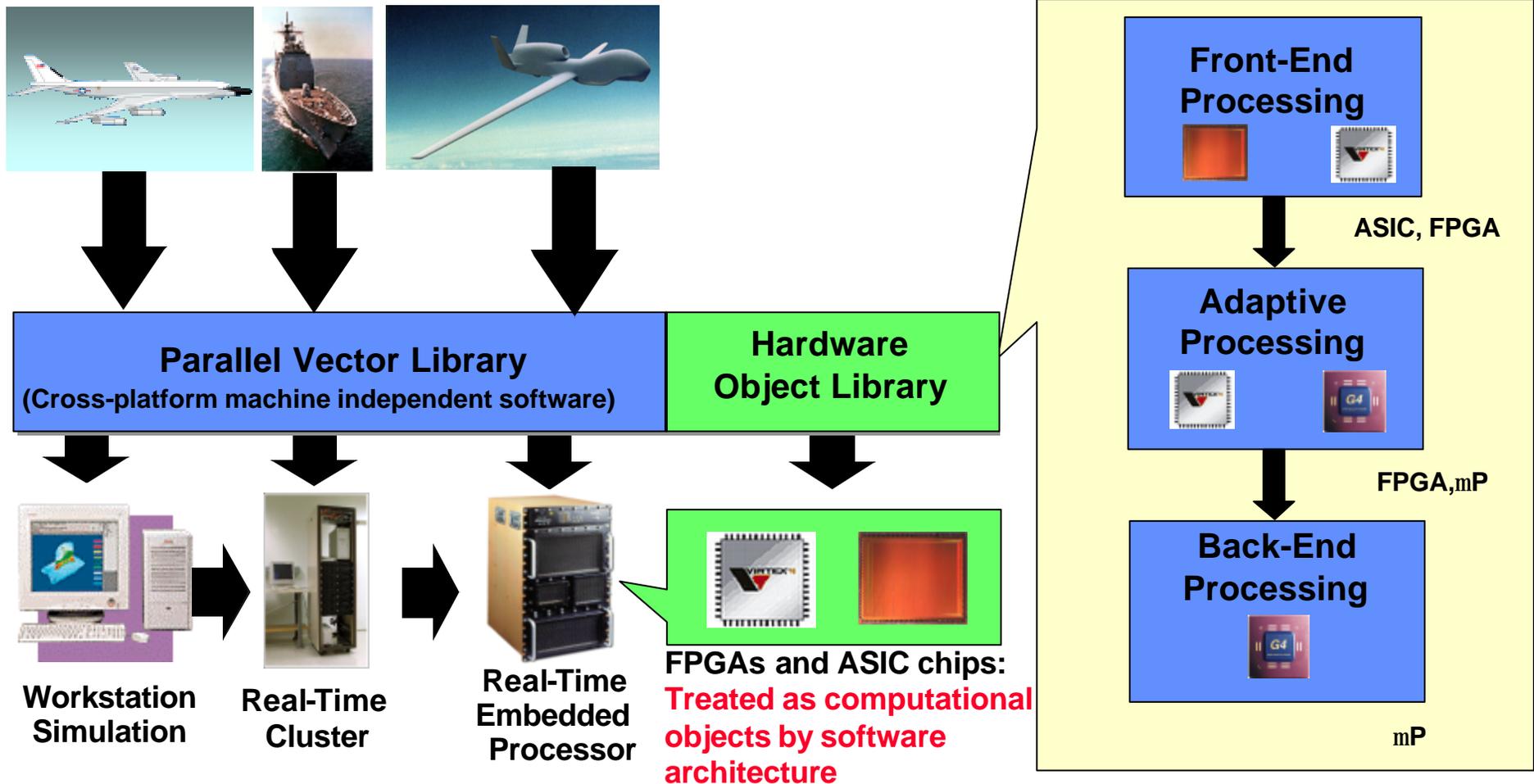
Lincoln Laboratory



Extension of PVL into Heterogeneous Processor Hardware Architecture



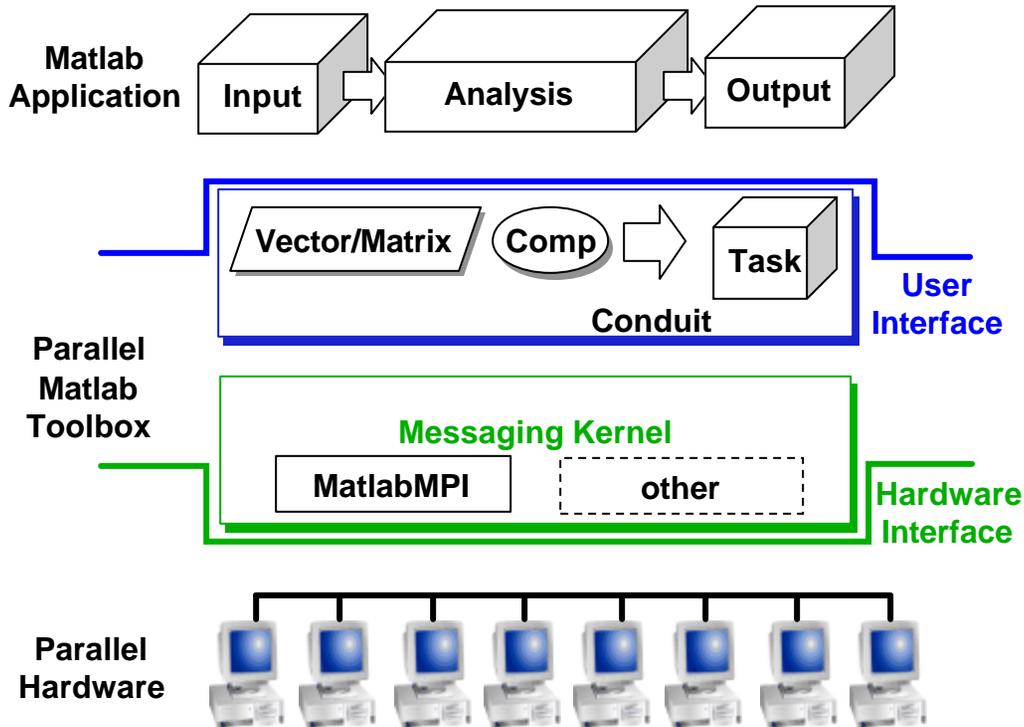
Future Radar Signal Processing Architecture



Enables use of the appropriate hardware for each algorithm with simplified programming



Parallel Matlab



- **Preserve existing Matlab interface**
- **Hide all parallel communication**
- **Allow transparent switching between single and multi processor cases**

<http://www.ll.mit.edu/MatlabMPI>

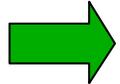
- **DoD has a clear need to rapidly develop, test and deploy new techniques for analyzing sensor data**
 - Most DoD algorithm development and simulations are done in Matlab
 - High performance has traditionally been available only in other languages
 - Transformation involves extensive software development and testing
- **Parallel Matlab eliminates transformation by providing**
 - High performance (parallel processing)
 - High productivity (10x fewer lines of code than C)
 - High portability (Matlab available on many platforms)



Outline



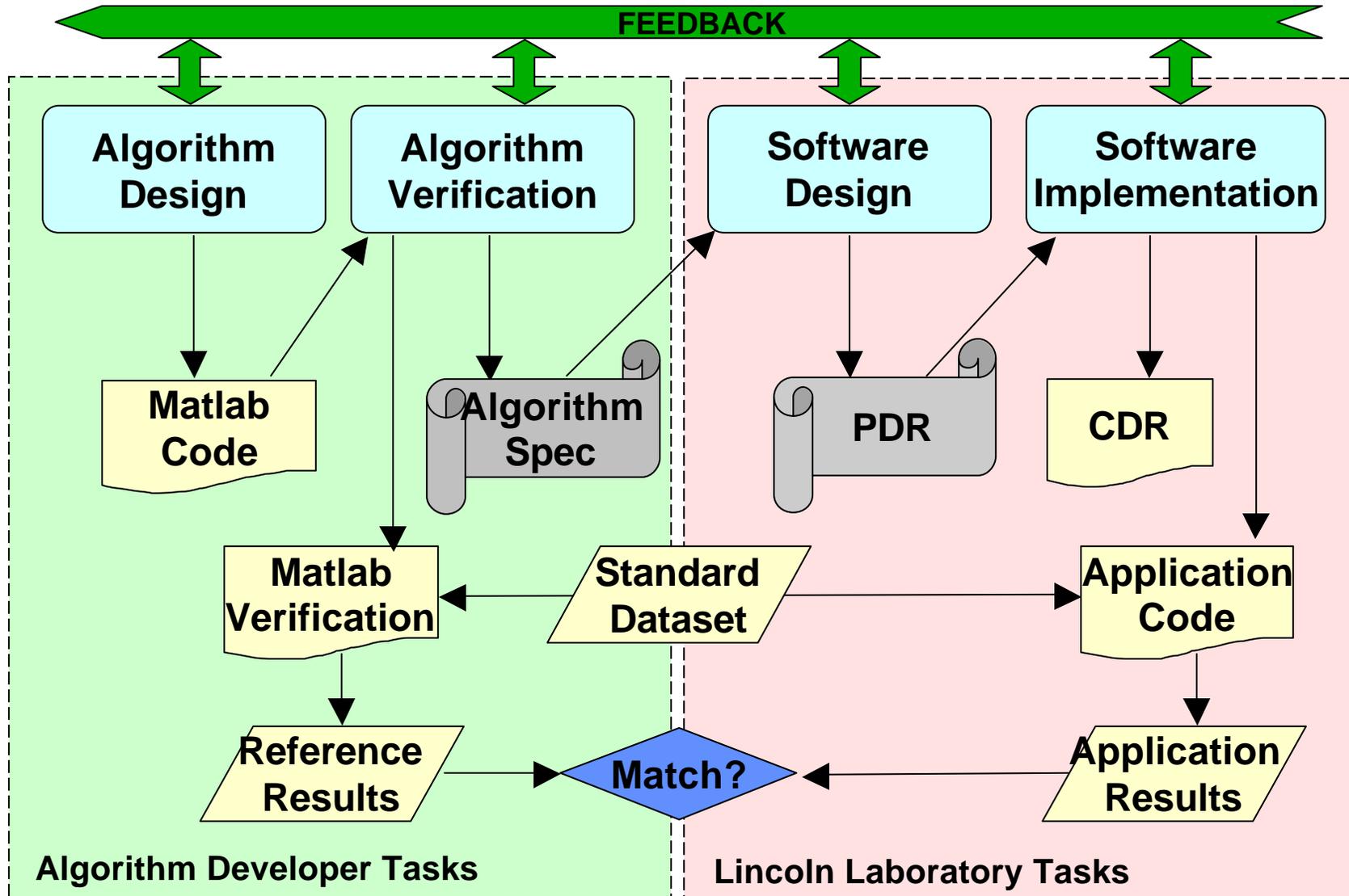
- KASSPER Testbed
- Parallel Vector Library



Summary



KASSPER Team Processor Testbed Algorithm Insertion Process





Summary



- **Developing a processor architecture to support KASSPER**
 - Predictive Scheduling
 - Knowledge Database
 - Intelligent Caching
- **Preliminary hardware and software architectures have been defined to develop and test processor concepts**
- **GMTI baseline application running on the parallel processor**
- **PVL Port to the Mercury processor is complete**
- **Near-term activities**
 - Simplified test cases for investigating architecture concepts
 - GMTI Performance optimization
 - Baseline SAR implementation
 - Multi-mode operation
 - High speed input data storage
 - Processor expansion