A KASSPER Real-Time Embedded Signal Processor Testbed

Glenn Schrader

Massachusetts Institute of Technology
Lincoln Laboratory

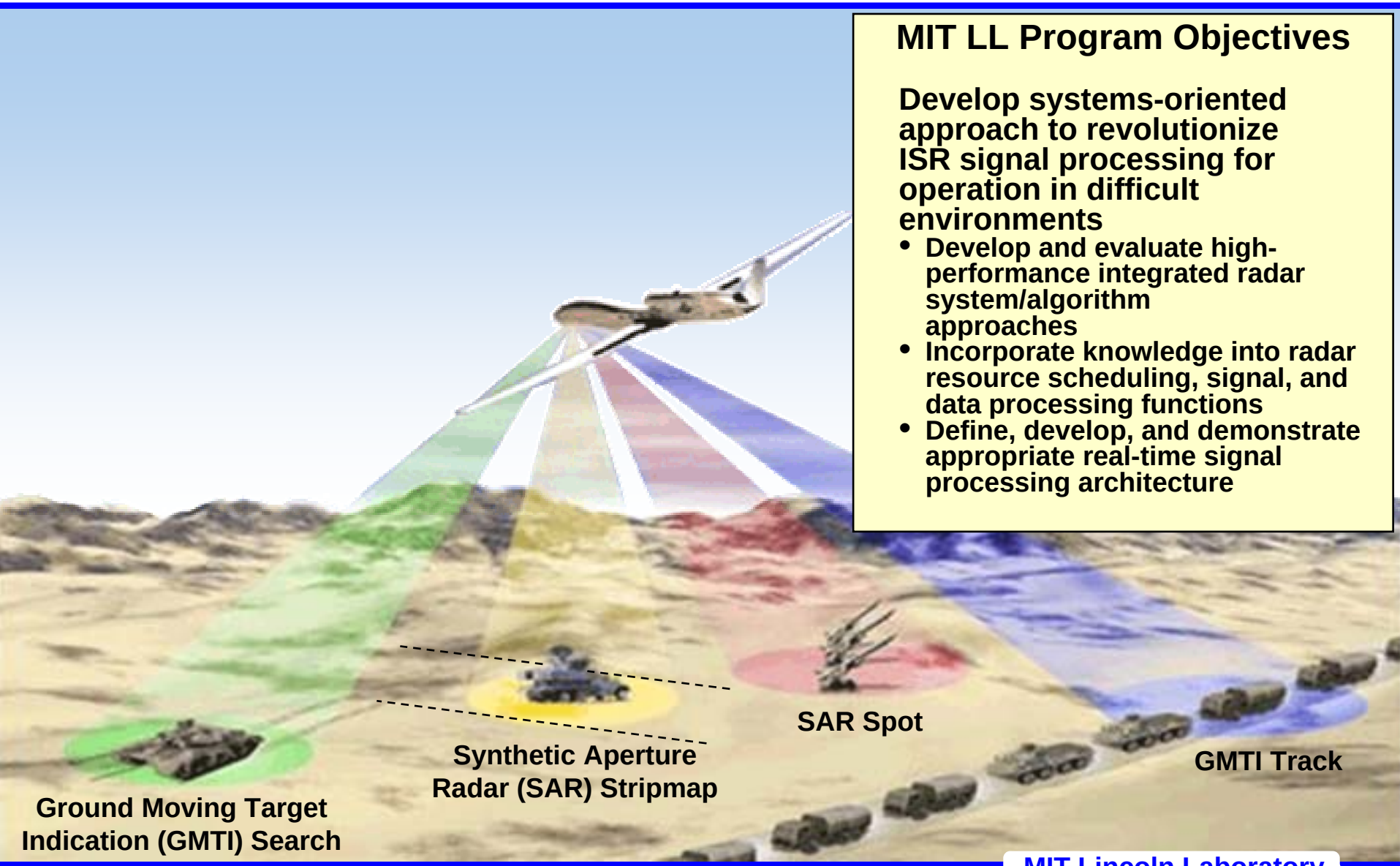
28 September 2004

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.

Outline

- **KASSPER Program Overview**
- **KASSPER Processor Testbed**
- **Computational Kernel Benchmarks**
  - **STAP Weight Application**
  - **Knowledge Database Pre-processing**
- **Summary**

Knowledge-Aided Sensor Signal Processing and Expert Reasoning (KASSPER)



MIT LL Program Objectives

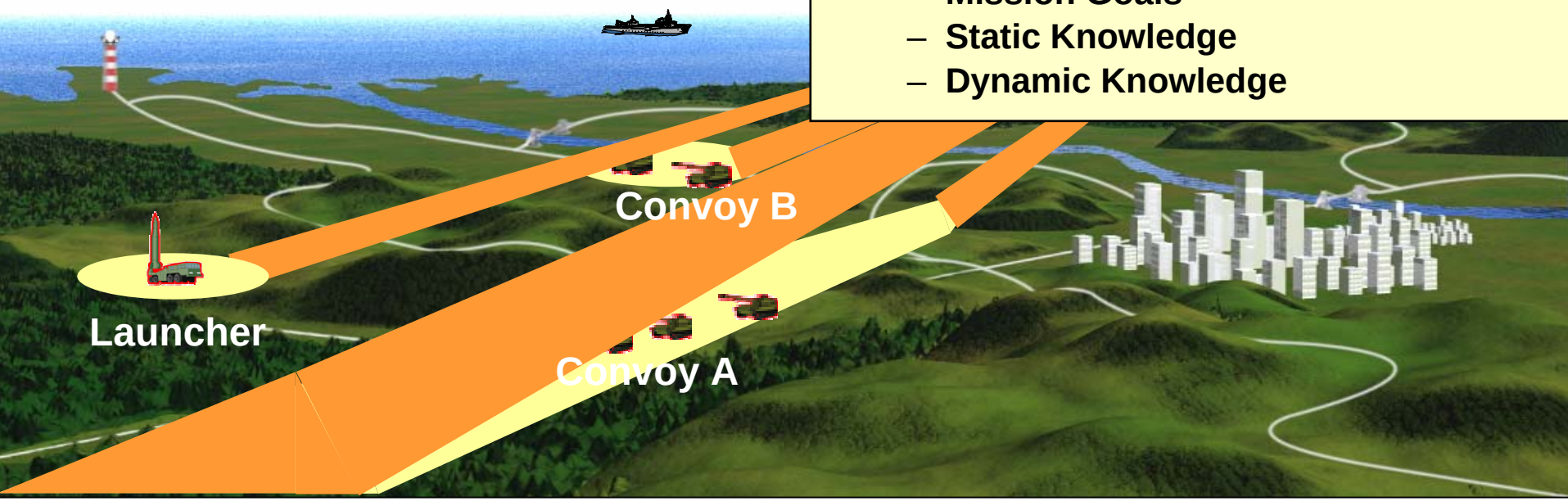
Develop systems-oriented approach to revolutionize ISR signal processing for operation in difficult environments

- **Develop and evaluate high-performance integrated radar system/algorithm approaches**
- **Incorporate knowledge into radar resource scheduling, signal, and data processing functions**
- **Define, develop, and demonstrate appropriate real-time signal processing architecture**

Representative KASSPER Problem

- **Radar System Challenges**
  - Heterogenous clutter environments
  - Clutter discretes
  - Dense target environments
- **Results in decreased system performance (e.g. higher Minimum Detectable Velocity, Probability of Detection vs. False Alarm Rate)**

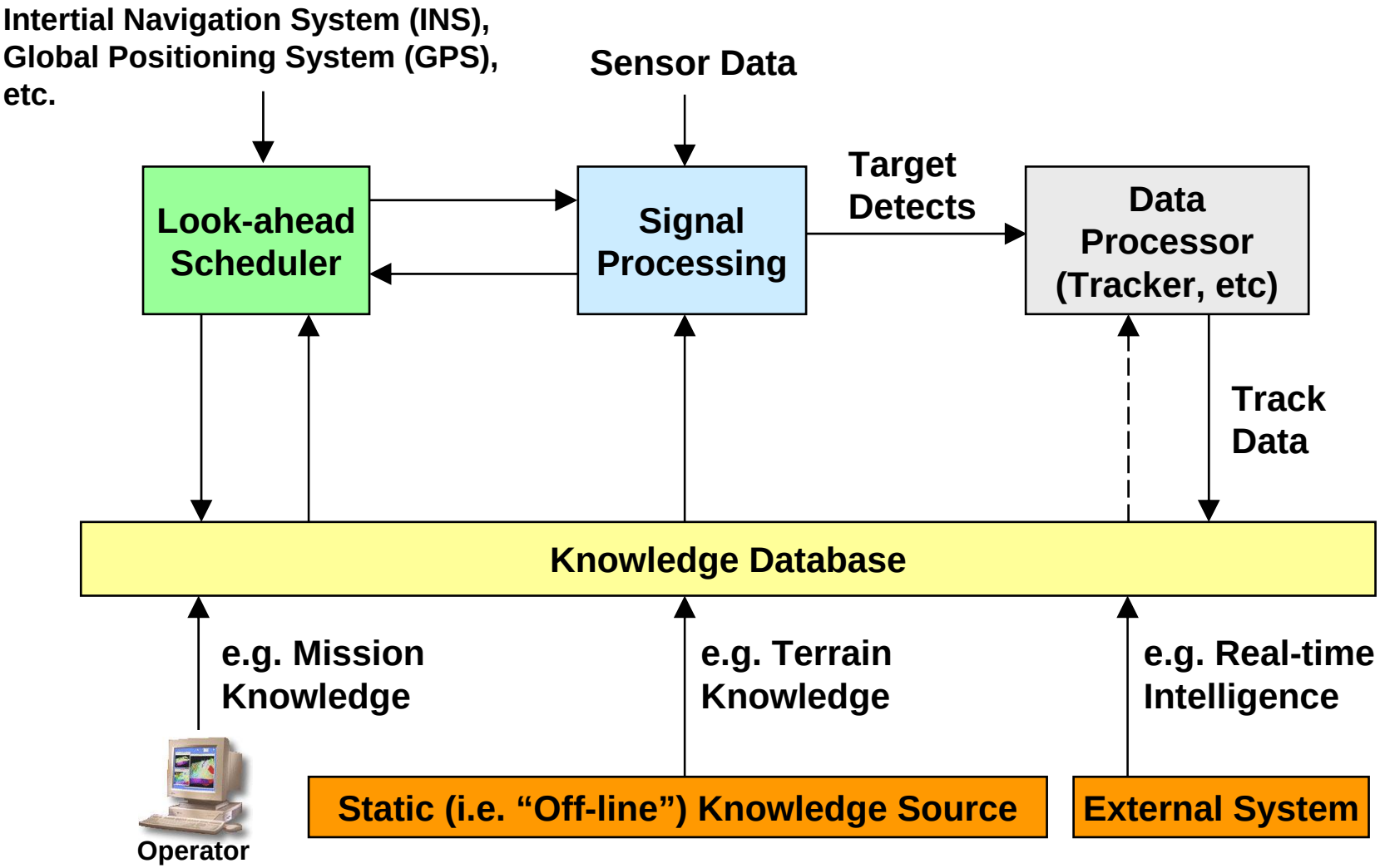
- **KASSPER System Characteristics**
  - Knowledge Processing
    - Knowledge-enabled Algorithms
    - Knowledge Repository
  - Multi-function Radar
    - Multiple Waveforms + Algorithms
    - Intelligent Scheduling
- **Knowledge Types**
  - Mission Goals
  - Static Knowledge
  - Dynamic Knowledge



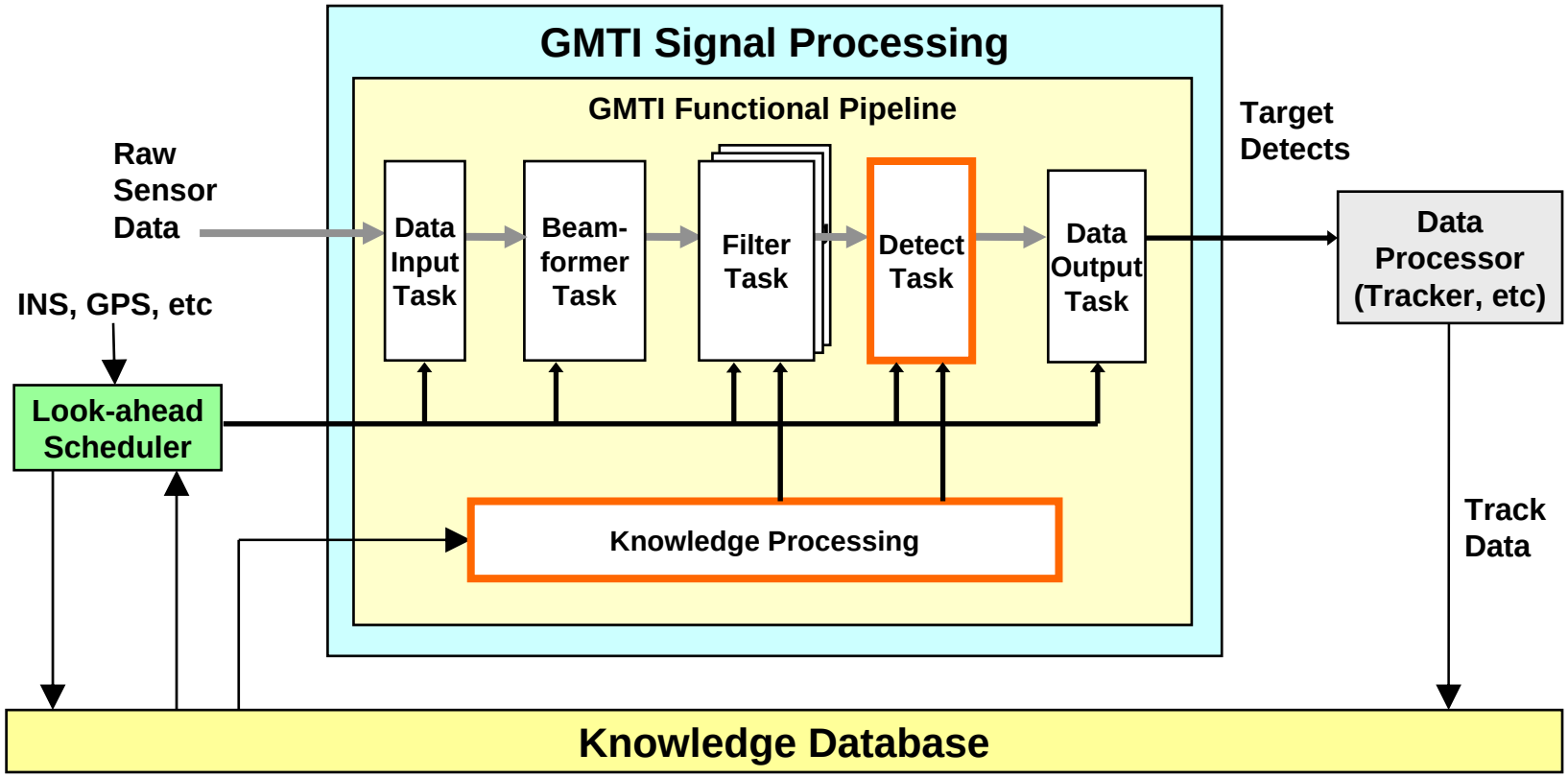
Outline

- KASSPER Program Overview
- KASSPER Processor Testbed
- Computational Kernel Benchmarks
  - STAP Weight Application
  - Knowledge Database Pre-processing
- Summary

High Level KASSPER Architecture



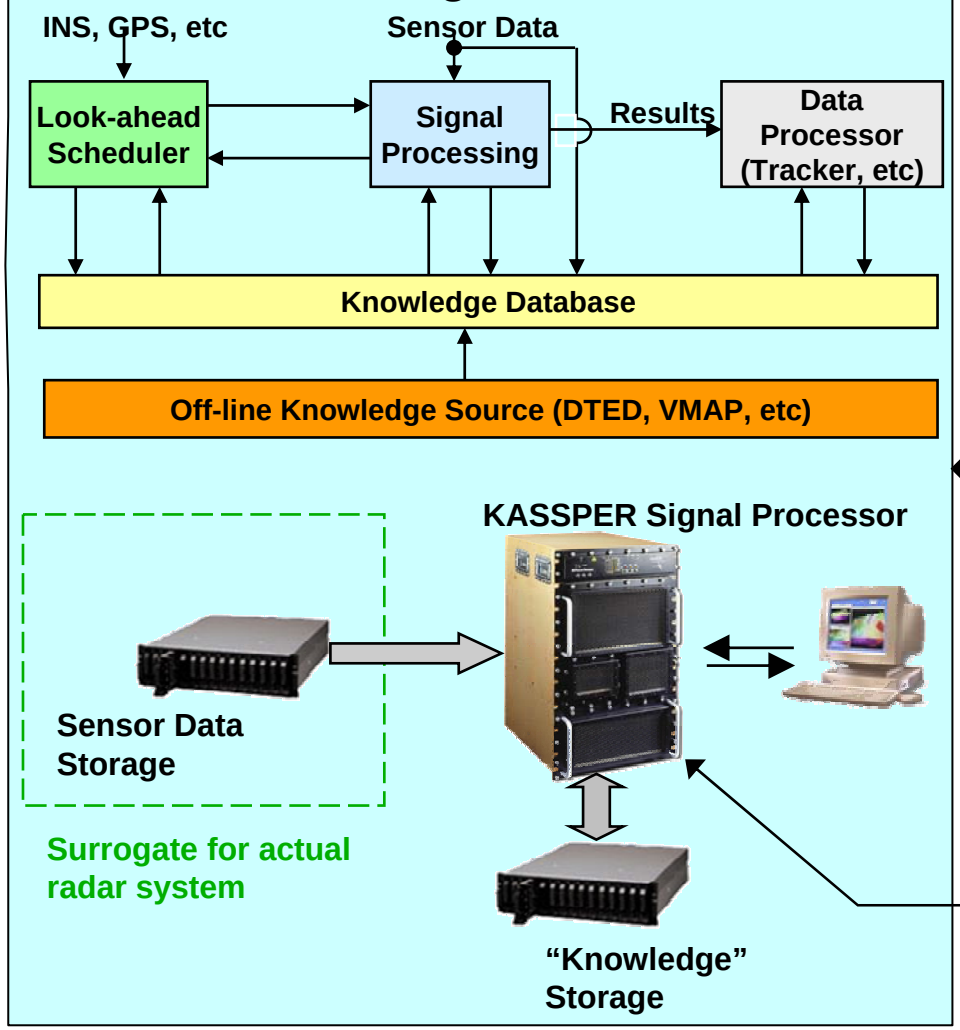
Baseline Signal Processing Chain



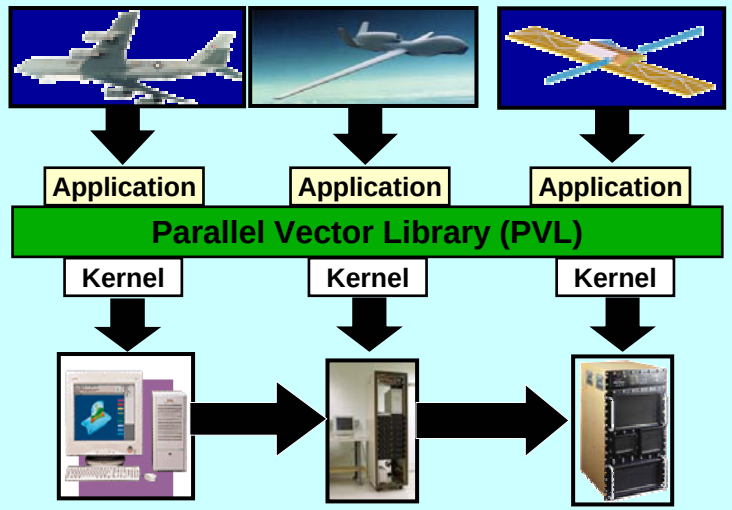
Baseline data cube sizes:
- 3 channels x 131 pulses x 1676 range cells
- 12 channels x 35 pulses x 1666 range cells

KASSPER Real-Time Processor Testbed

Processing Architecture



Software Architecture

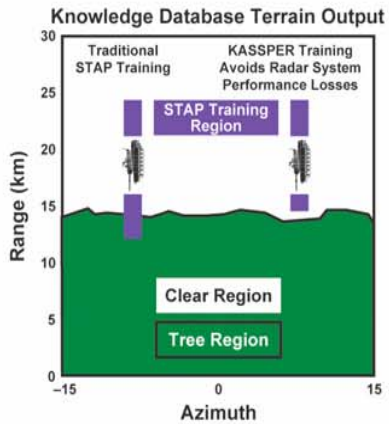


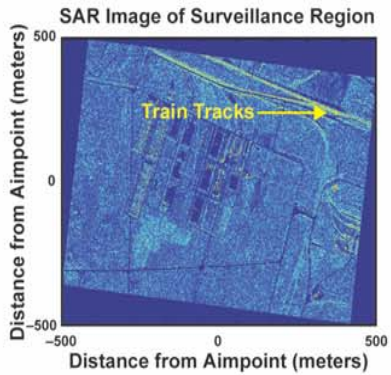
- Rapid Prototyping
- High Level Abstractions
- Scalability
- Productivity
- Rapid Algorithm Insertion

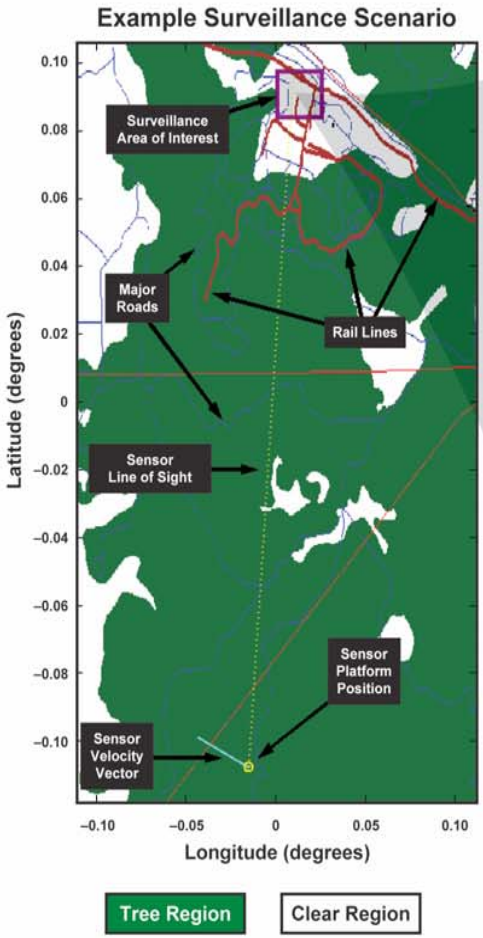
120 PPC G4 CPUs @ 500 MHZ
4 GFlop/Sec/CPU = 480 GFlop/Sec

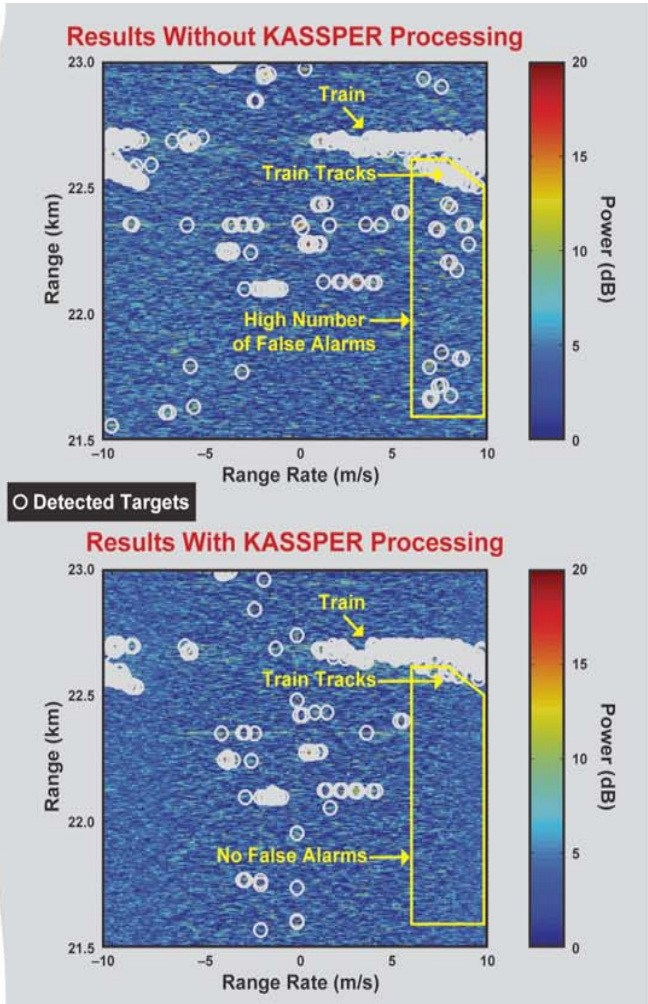
KASSPER Knowledge-Aided Processing Benefits

Additional Knowledge Sources for Use in KASSPER Processing





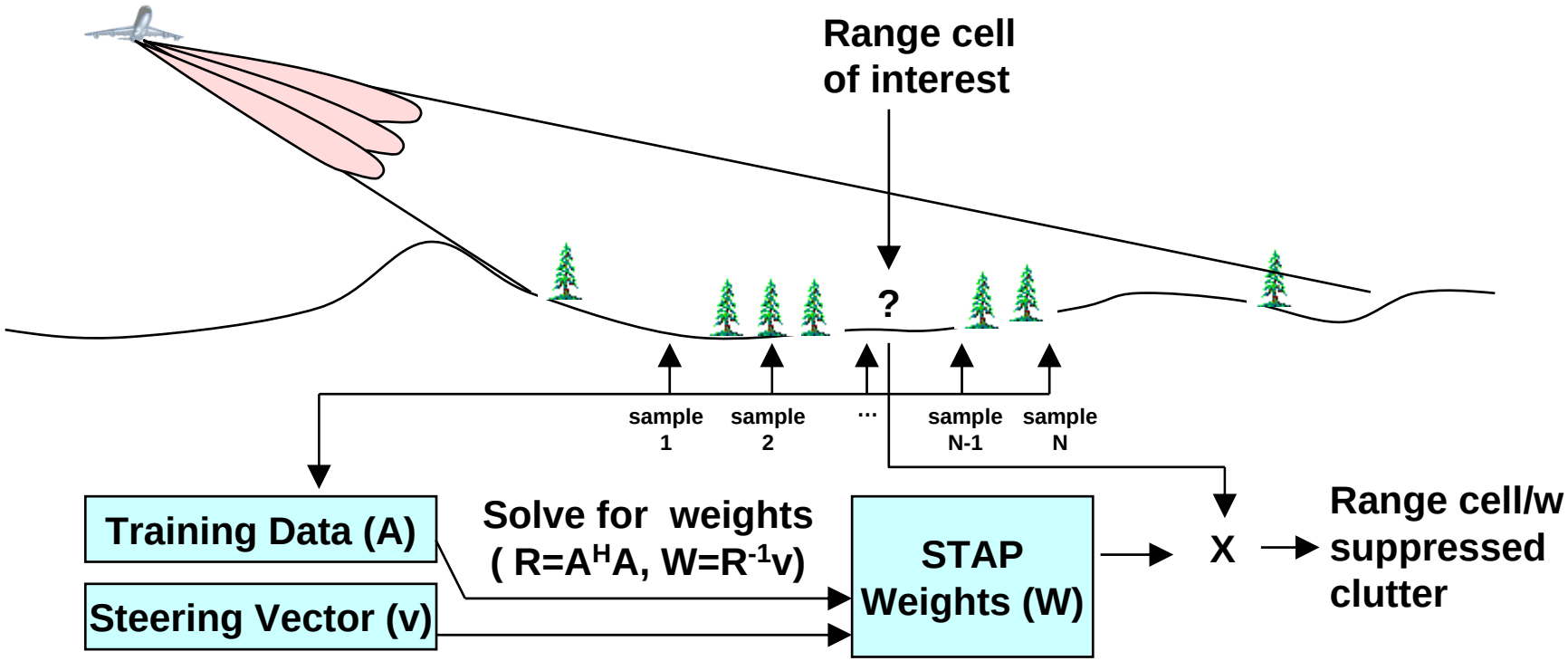




Outline

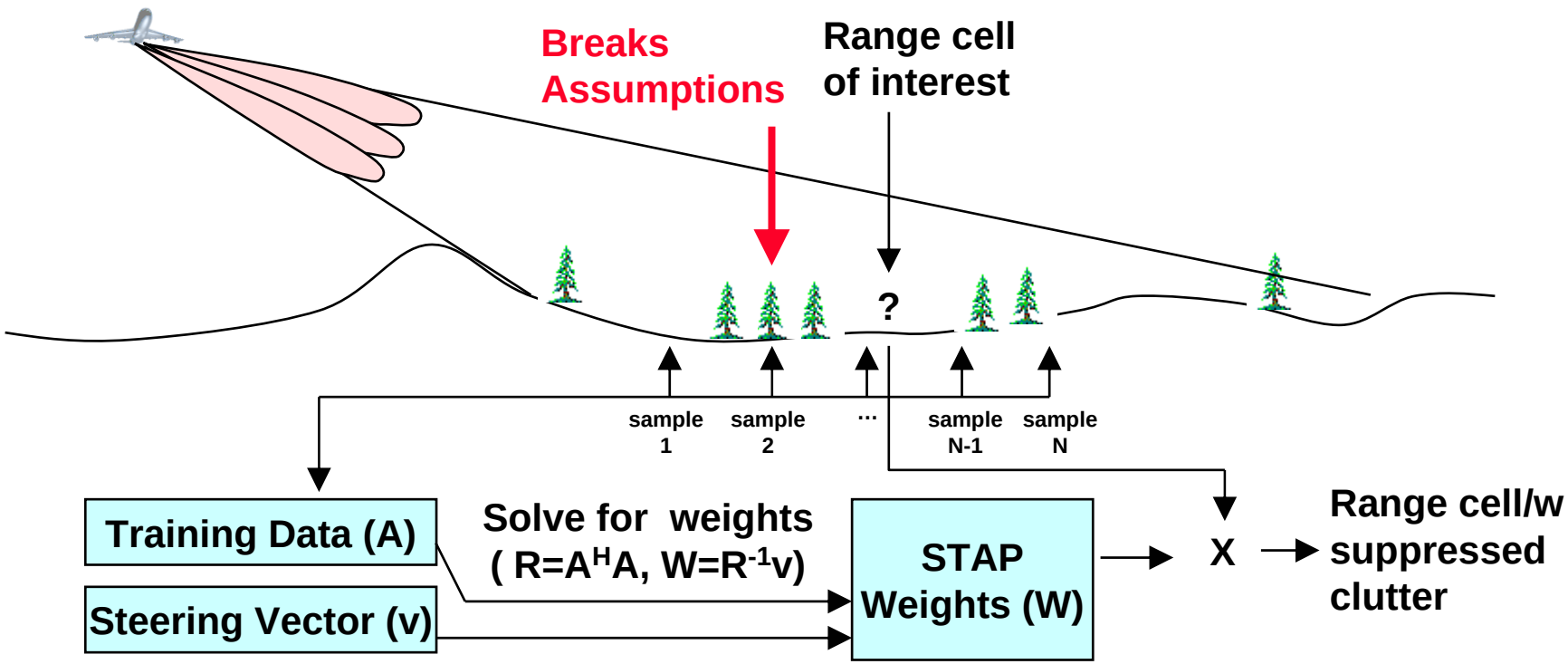
- KASSPER Program Overview
- KASSPER Processor Testbed
- Computational Kernel Benchmarks
  - STAP Weight Application
  - Knowledge Database Pre-processing
- Summary

Space Time Adaptive Processing (STAP)



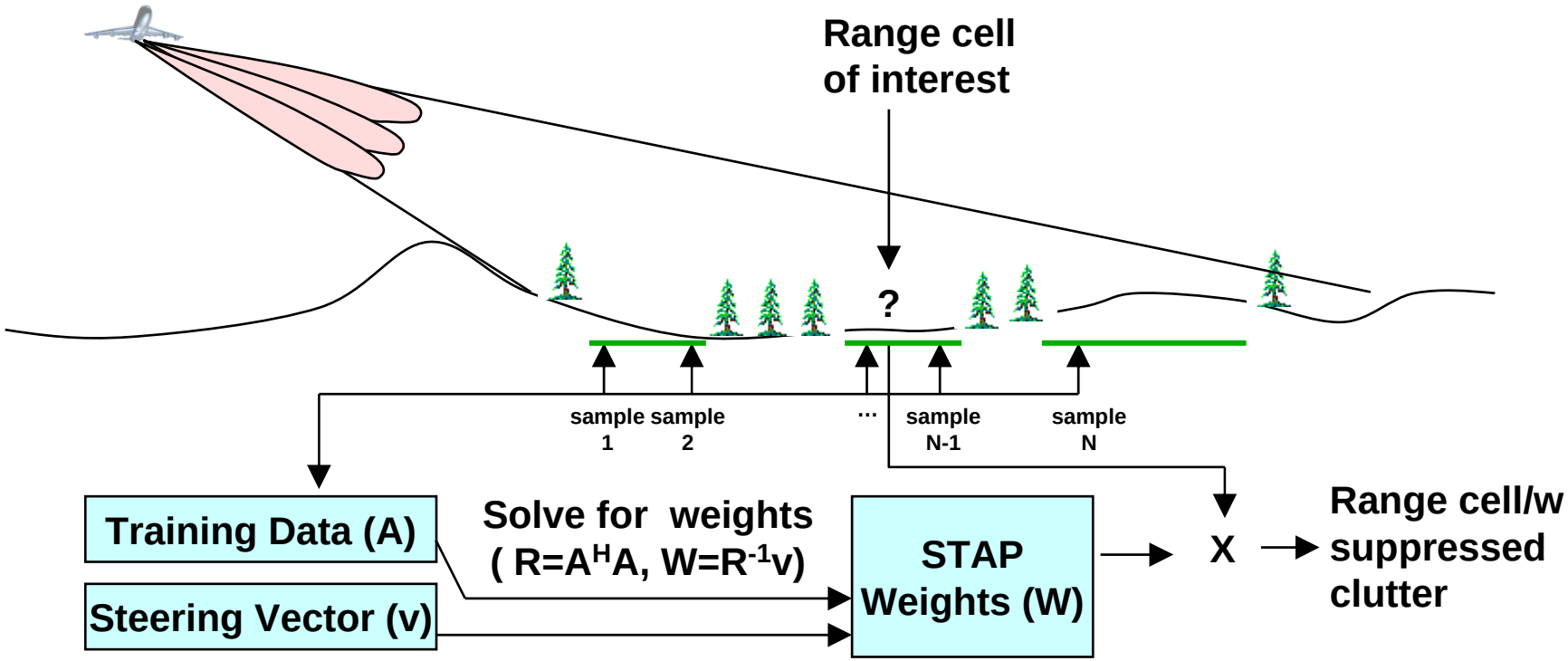
- Training samples are chosen to form an estimate of the background
- Multiplying the STAP Weights by the data at a range cell suppresses features that are similar to the features that were in the training data

Space Time Adaptive Processing (STAP)



- Simple training selection approaches may lead to degraded performance since the clutter in training set may not match the clutter in the range cell of interest

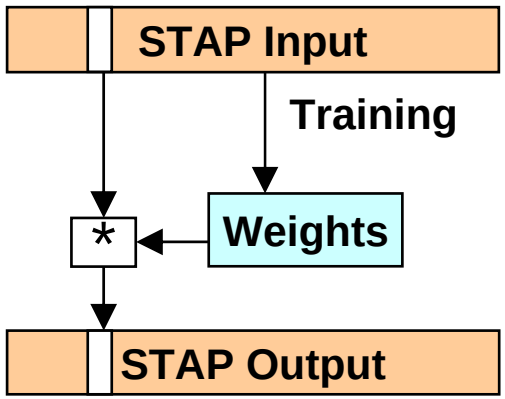
Space Time Adaptive Processing (STAP)

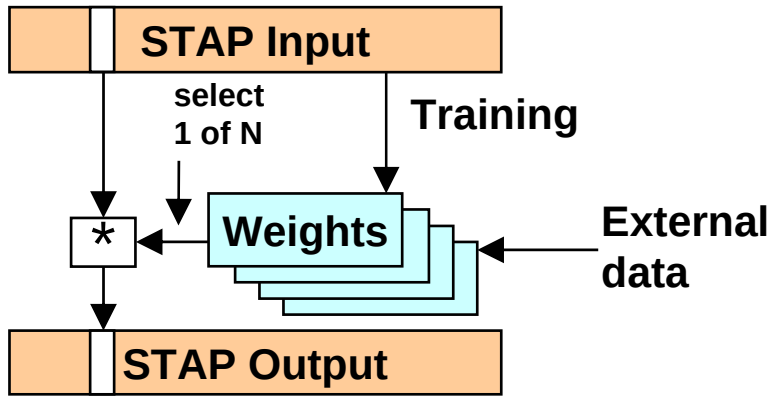


- Use of terrain knowledge to select training samples can mitigate the degradation

STAP Weights and Knowledge

- STAP weights computed from training samples within a training region
- To improve processor efficiency, many designs apply weights across a swath of range (i.e. matrix multiply)
- With knowledge enhanced STAP techniques, MUST assume that adjacent range cells will not use the same weights (i.e. weight selection/computation is data dependant)





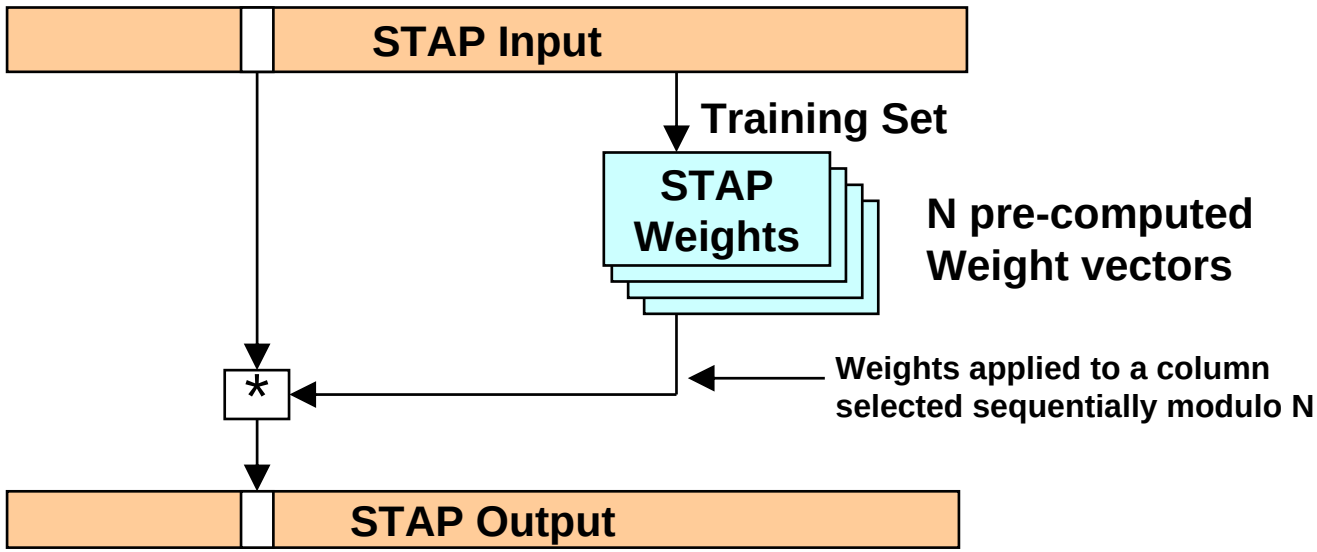
Equivalent to Matrix Multiply since same weights are used for all columns

Cannot use Matrix Multiply since the weights applied to each column are data dependant

Benchmark Algorithm Overview

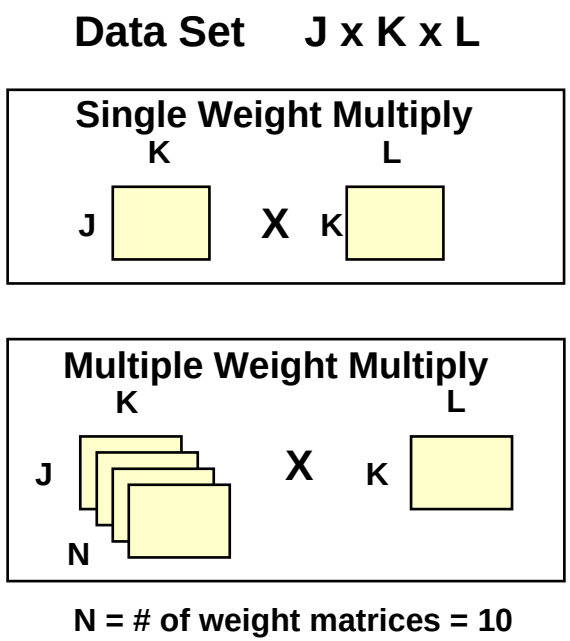
Benchmark Reference Algorithm (Single Weight Multiply)

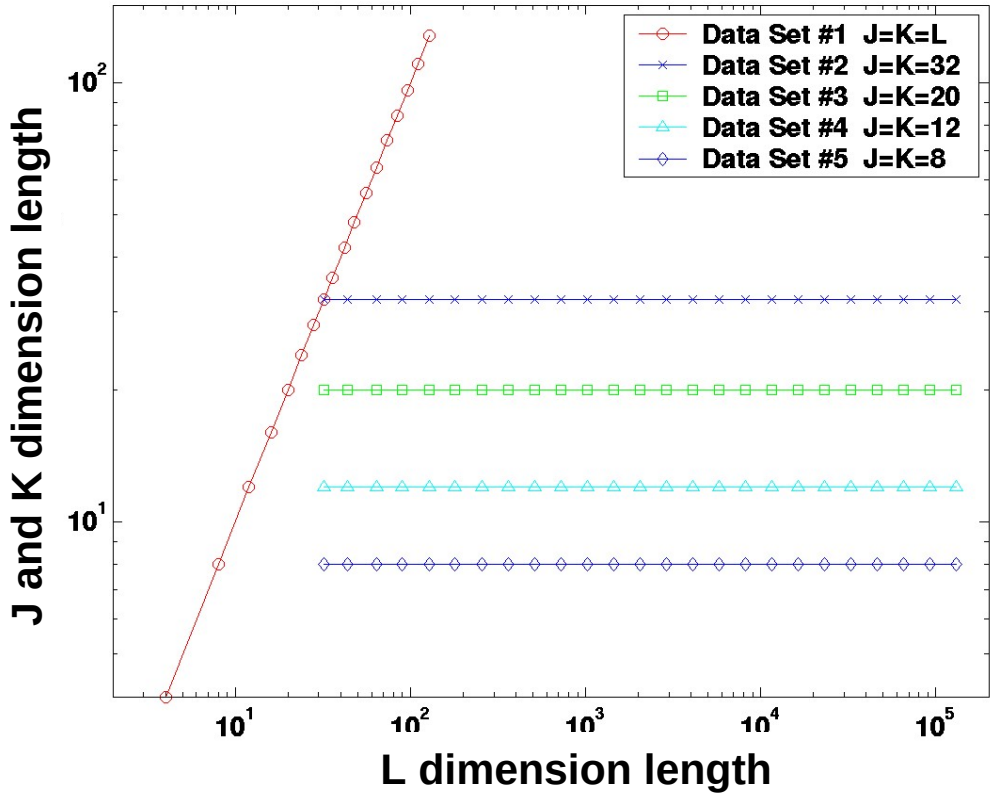
Benchmark Data Dependant Algorithm (Multiple Weight Multiply)



Benchmark's goal is to determine the achievable performance compared to the well-known matrix multiply algorithm.

Test Case Overview

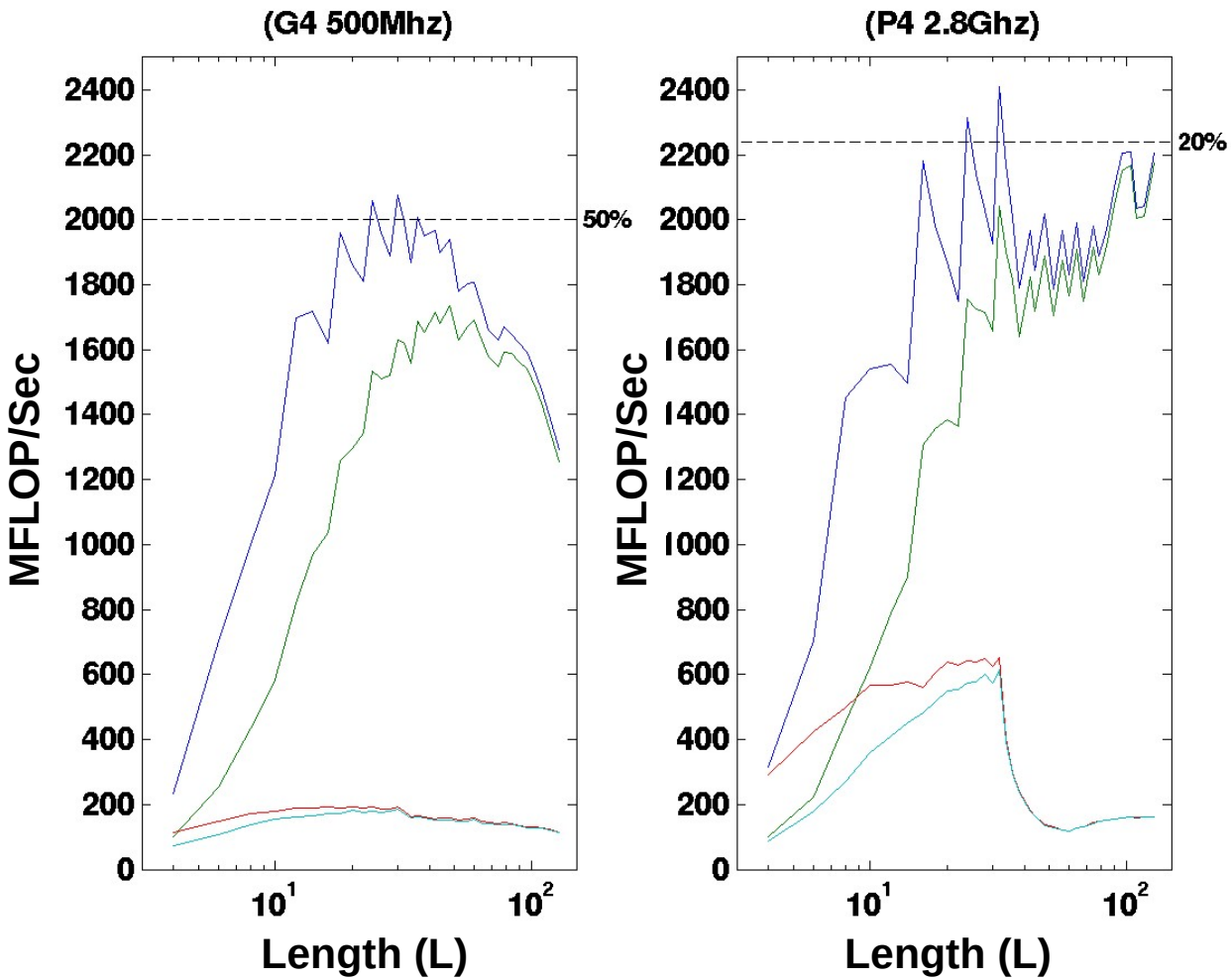




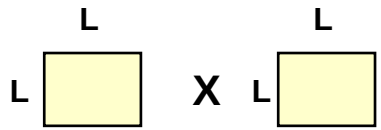
- **Complex Data**
  - Single precision interleaved
- **Two test architectures**
  - PowerPC G4 (500Mhz)
  - Pentium 4 (2.8Ghz)

- **Four test cases**
  - Single weight multiply/wo cache flush
  - Single weight multiply/w cache flush
  - Multiple weight multiply/wo cache flush
  - Multiple weight multiply/w cache flush

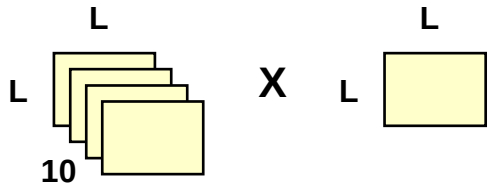
Data Set #1 - LxLxL



Single Weight Multiply

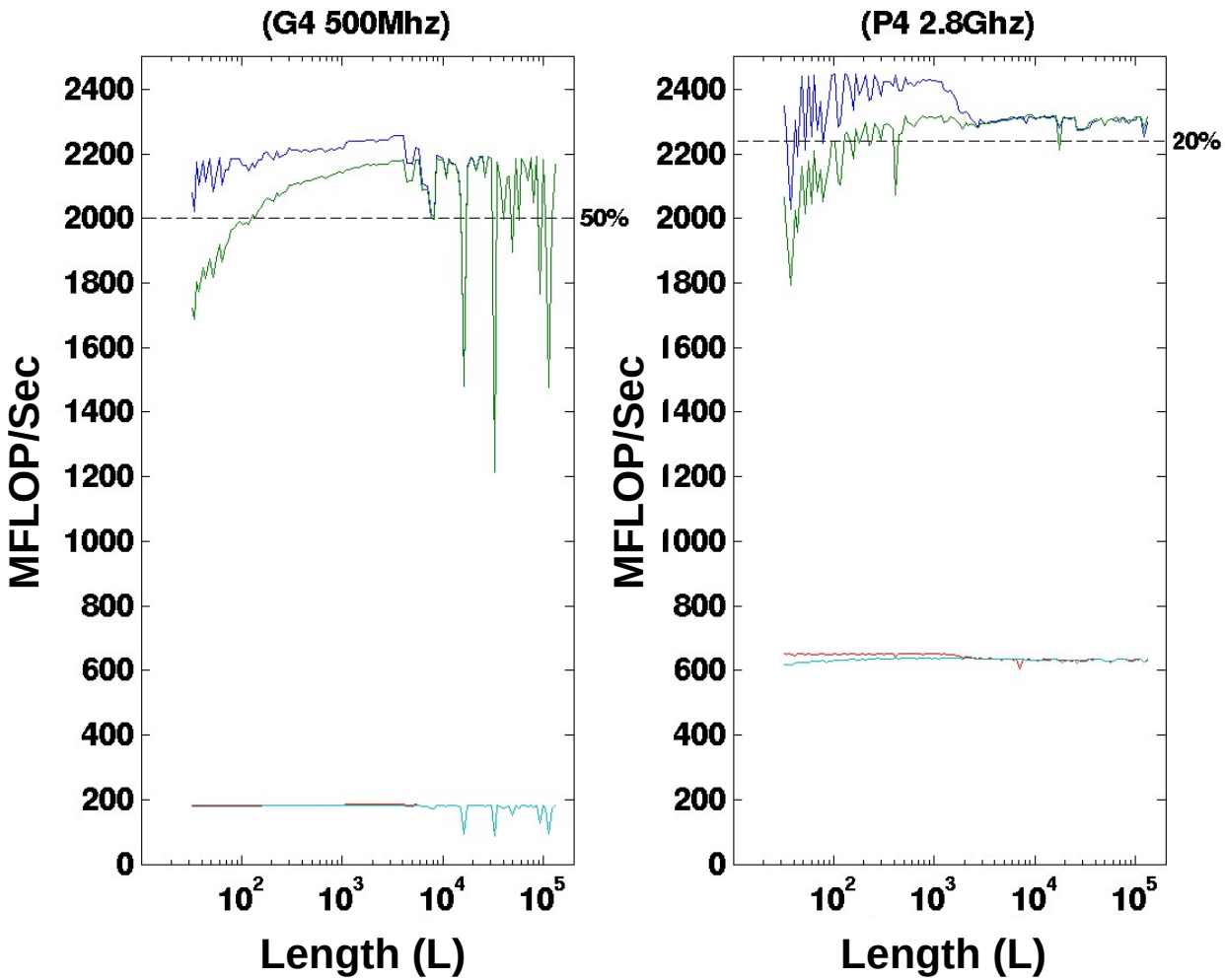


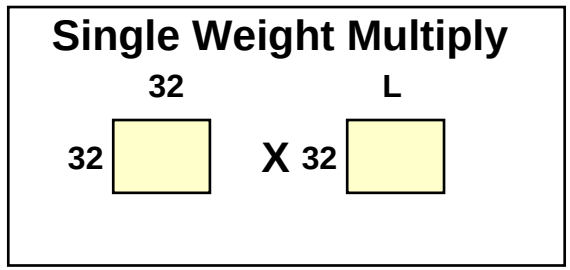
Multiple Weight Multiply

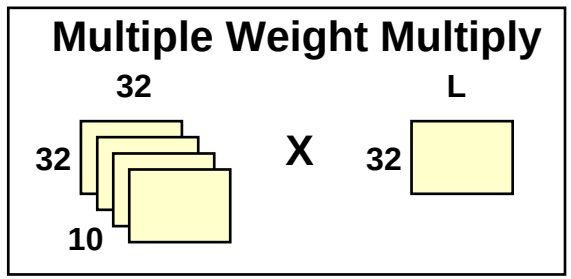


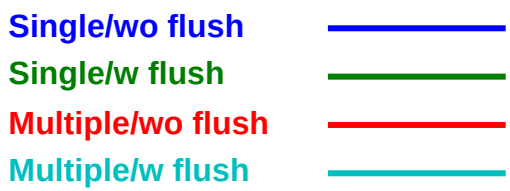
Single/wo flush
Single/w flush
Multiple/wo flush
Multiple/w flush

Data Set #2 – 32x32xL

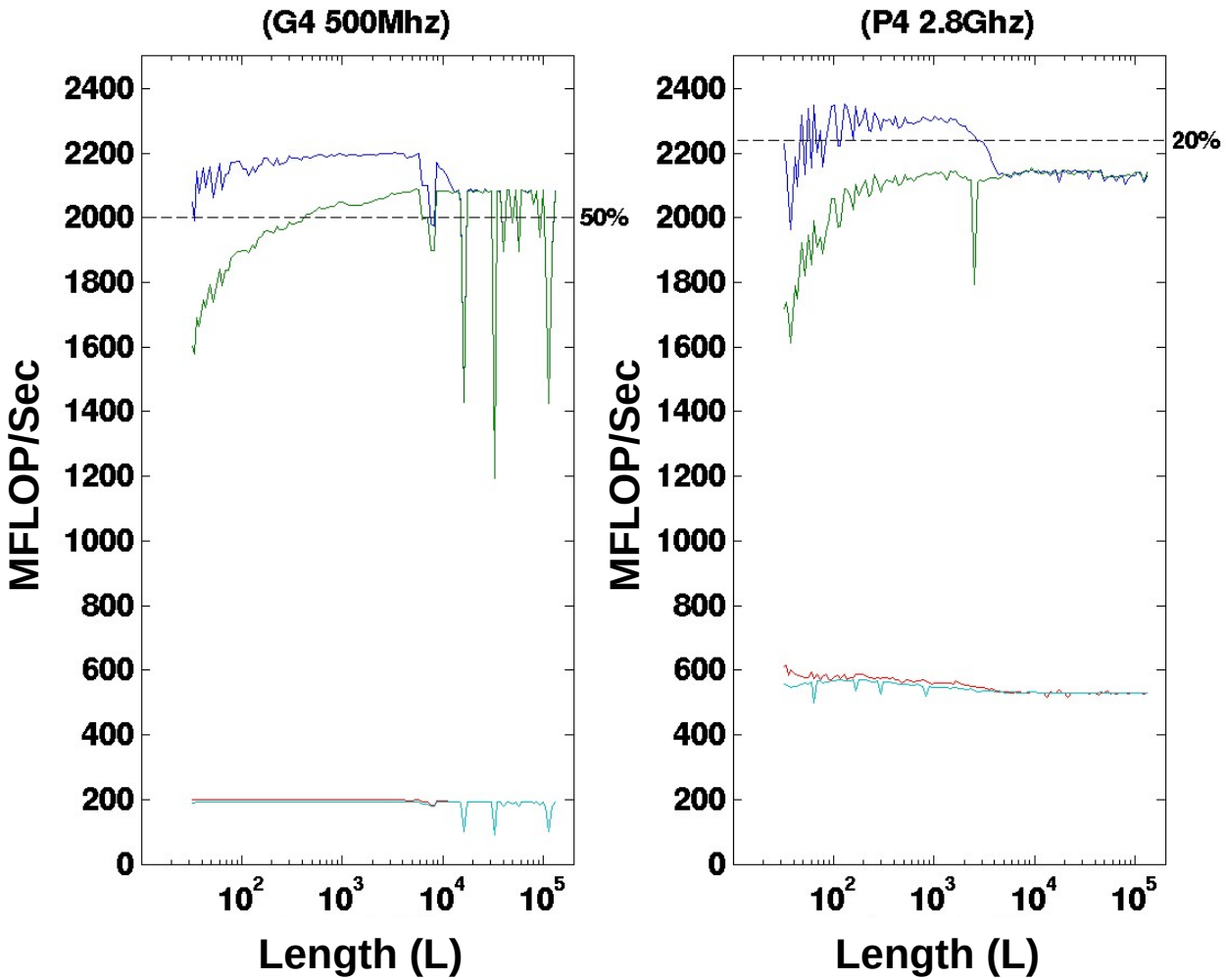


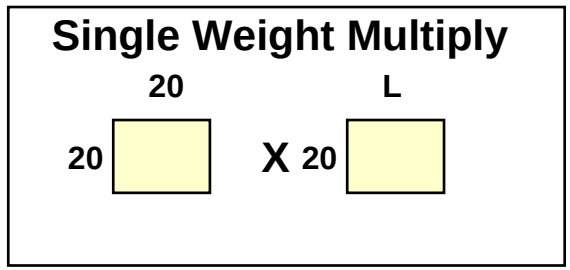


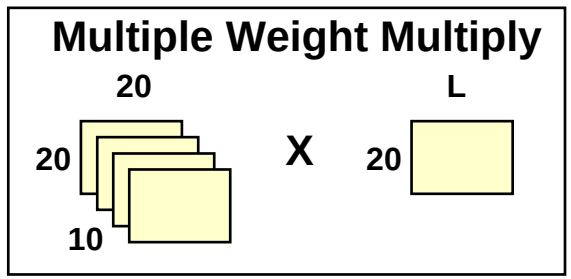




Data Set #3 – 20x20xL







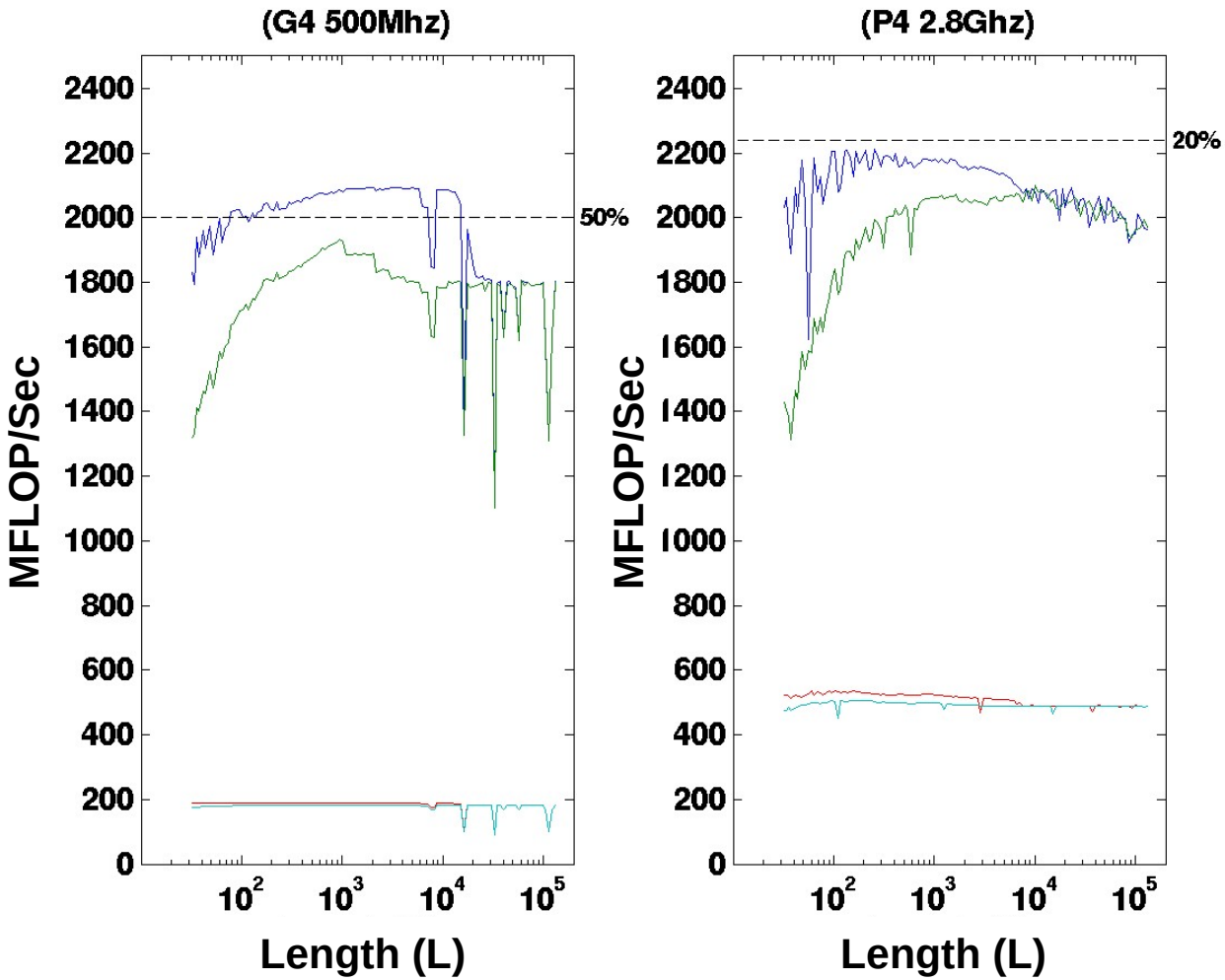
Single/wo flush

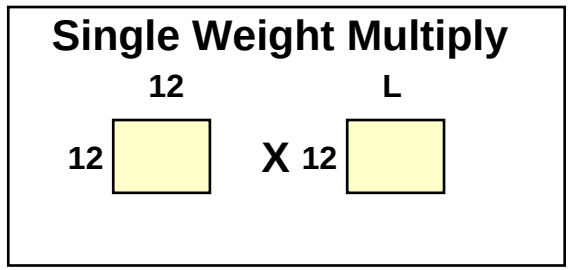
Single/w flush

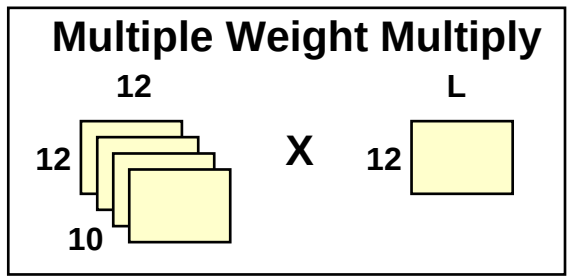
Multiple/wo flush

Multiple/w flush

Data Set #4 – 12x12xL







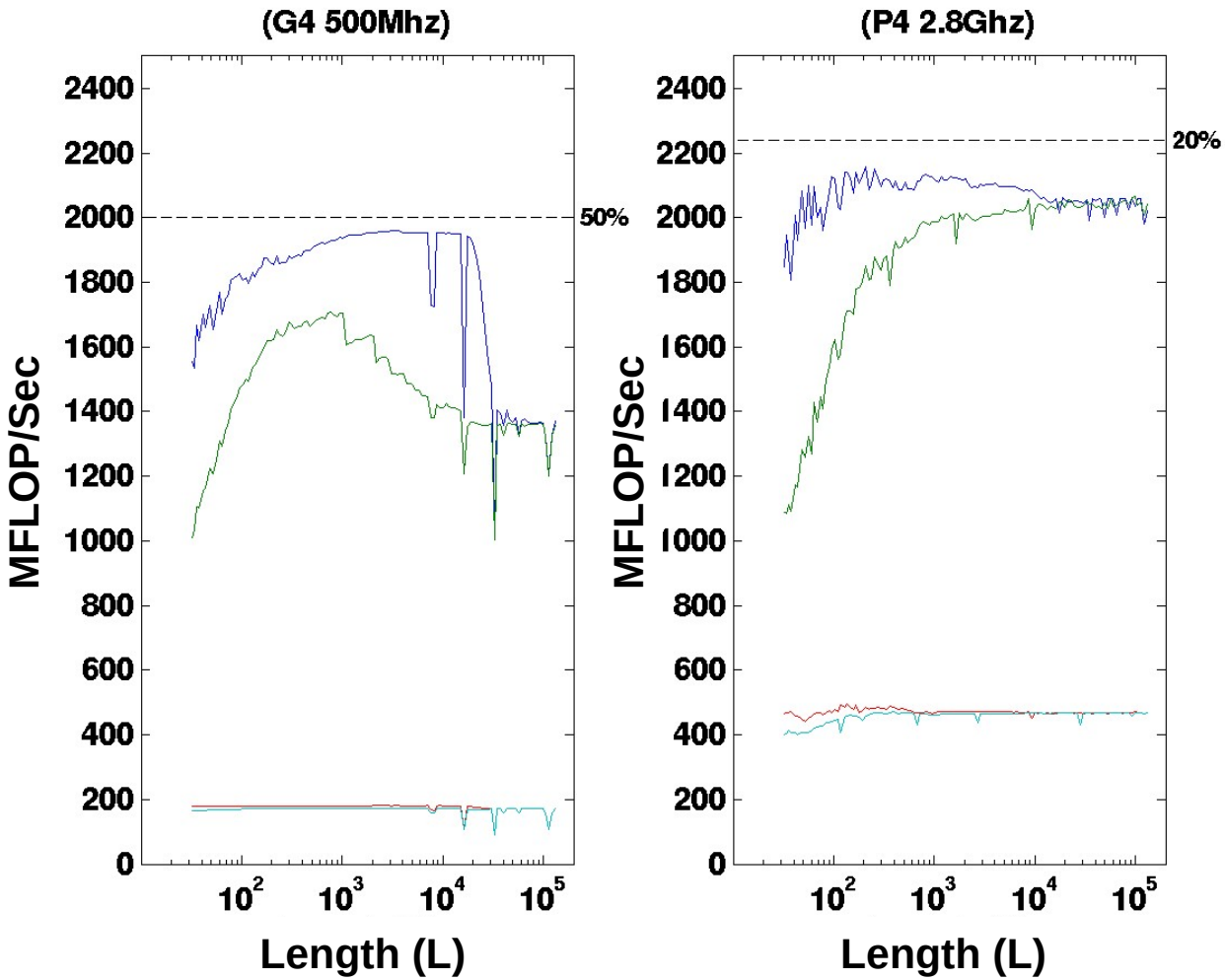
Single/wo flush

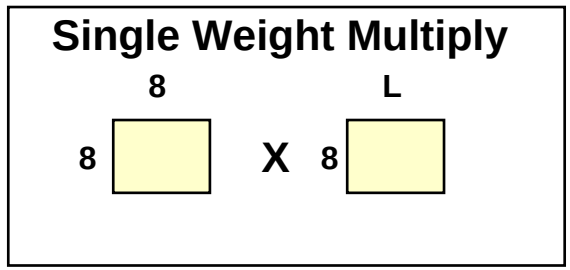
Single/w flush

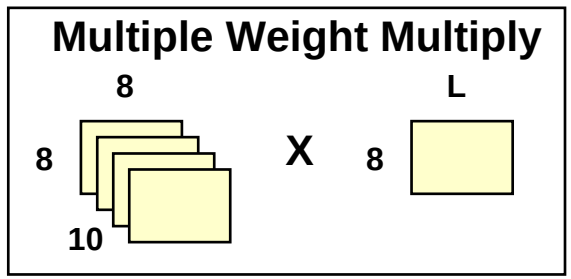
Multiple/wo flush

Multiple/w flush

Data Set #5 – 8x8xL







Single/wo flush

Single/w flush

Multiple/wo flush

Multiple/w flush

Performance Observations

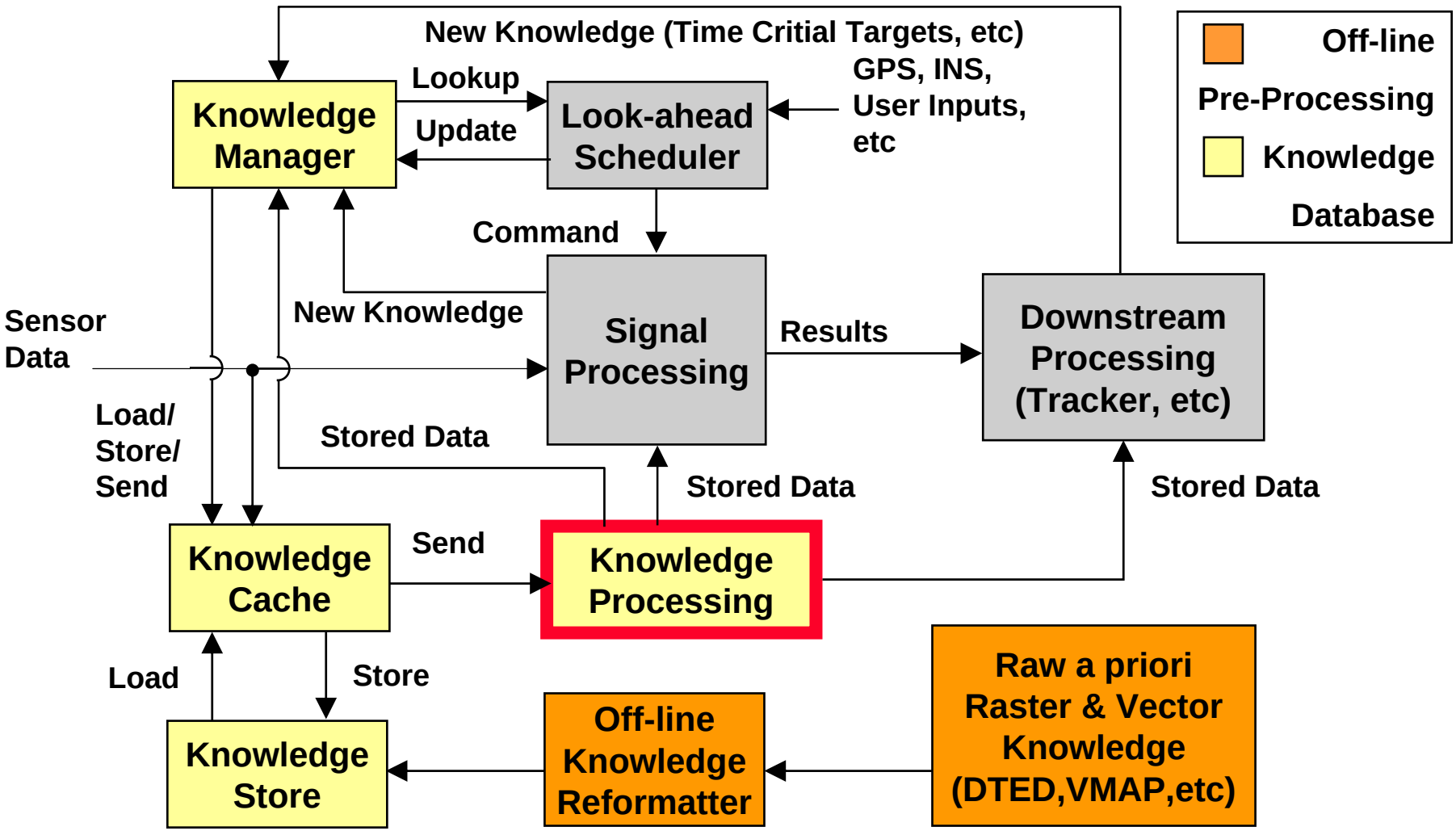
- Data dependent processing cannot be optimized as well
  - Branching prevents compilers from “unrolling” loops to improve pipelining
- Data dependent processing does not allow efficient “re-use” of already loaded data
  - Algorithms cannot make simplifying assumptions about already having the data that is needed.
- Data dependent processing does not vectorize
  - Using either Altivec or SSE is very difficult since data movement patterns become data dependant
- Higher memory bandwidth reduces cache impact
- Actual performance scales roughly with clock rate rather than theoretical peak performance

Approx 3x to 10x performance degradation,
Processing efficiency of 2-5% depending on CPU architecture

Outline

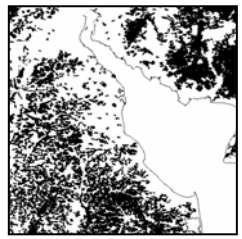
- KASSPER Program Overview
- KASSPER Processor Testbed
- Computational Kernel Benchmarks
  - STAP Weight Application
  - Knowledge Database Pre-processing
- Summary

Knowledge Database Architecture



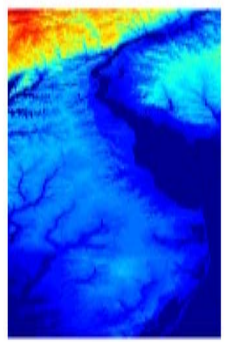
Static Knowledge

- **Vector Data**
  - Each feature represented by a set of point locations:
    - Points - longitude and latitude (i.e. towers, etc)
    - Lines - list of points, first and last points are not connected (i.e. roads, rail, etc)
    - Areas - list of points, first and last point are connected (i.e. forest, urban, etc)
  - Standard Vector Formats
    - Vector Smart Map (VMAP)
- **Matrix Data**
  - Rectangular arrays of evenly spaced data points
  - Standard Raster Formats
    - Digital Terrain Elevation Data (DTED)



Trees (area)

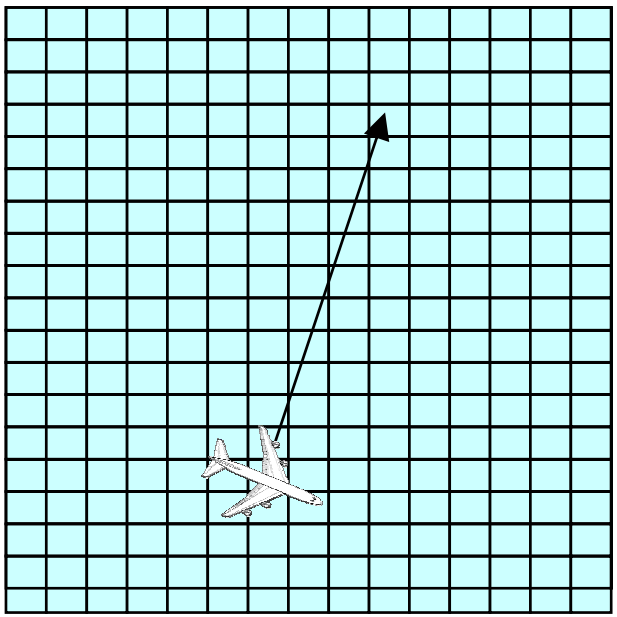
Roads (line)



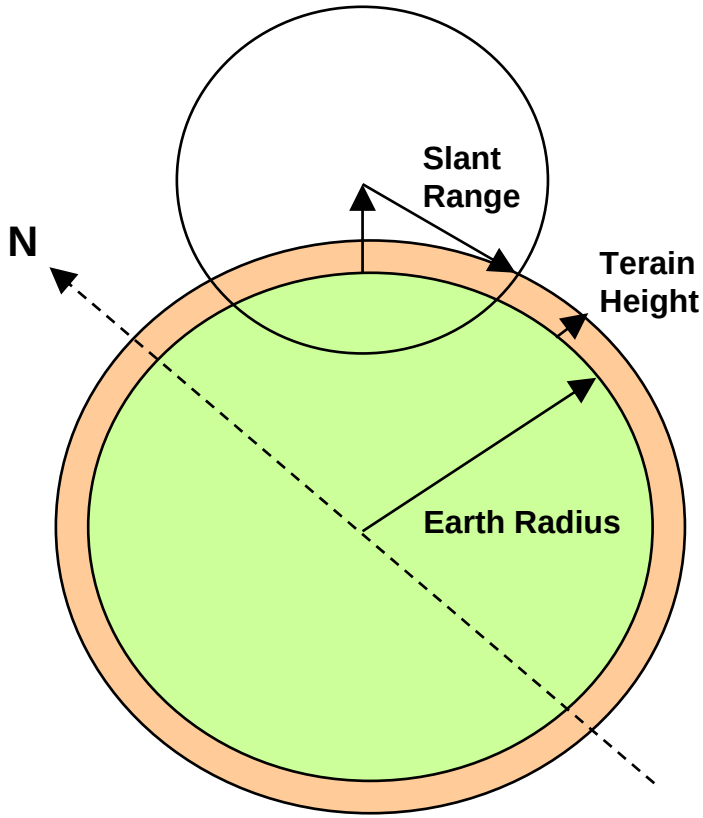
DTED

Terrain Interpolation

Sampled Terrain Height & Type data Data



Geographic Position



Converting from radar to geographic coordinates requires iterative refinement of estimate

Terrain Interpolation Performance Results

CPU Type	Time per interpolation	Processing Rate
P4 2.8	1.27uSec/interpolation	144MFlop/sec (1.2%)
G4 500	3.97uSec/interpolation	46MFlop/sec (2.3%)

- Data dependent processing does not vectorize
  - Using either Altivec or SSE is very difficult
- Data dependent processing cannot be optimized as well
  - Compilers cannot “unroll” loops to improve pipelining
- Data dependent processing does not allow efficient “re-use” of already loaded data
  - Algorithms cannot make simplifying assumptions about already having the data that is needed

Processing efficiency of 1-3% depending on CPU architecture

Outline

- KASSPER Program Overview
- KASSPER Processor Testbed
- Computational Kernel Benchmarks
  - STAP Weight Application
  - Knowledge Database Pre-processing
- Summary

Summary

- Observations
  - The use of knowledge processing provides a significant system performance improvement
  - Compared to traditional signal processing algorithms, the implementation of knowledge enabled algorithms can result in a factor of 4 or 5 lower CPU efficiency (as low as 1%-5%)
  - Next-generation systems that employ cognitive algorithms are likely to have similar performance issues
- Important Research Directions
  - Heterogeneous processors (i.e. a mix of COTS CPUs, FPGAs, GPUs, etc) can improve efficiency by better matching hardware to the individual problems being solved. For what class of systems is the current technology adequate?
  - Are emerging architectures for cognitive information processing needed (e.g. Polymorphous Computing Architecture – PCA, Application Specific Instruction set Processors - ASIPs)?