For:mmwildin
Printed on:Wed, Mar 17, 1999 12:52:01

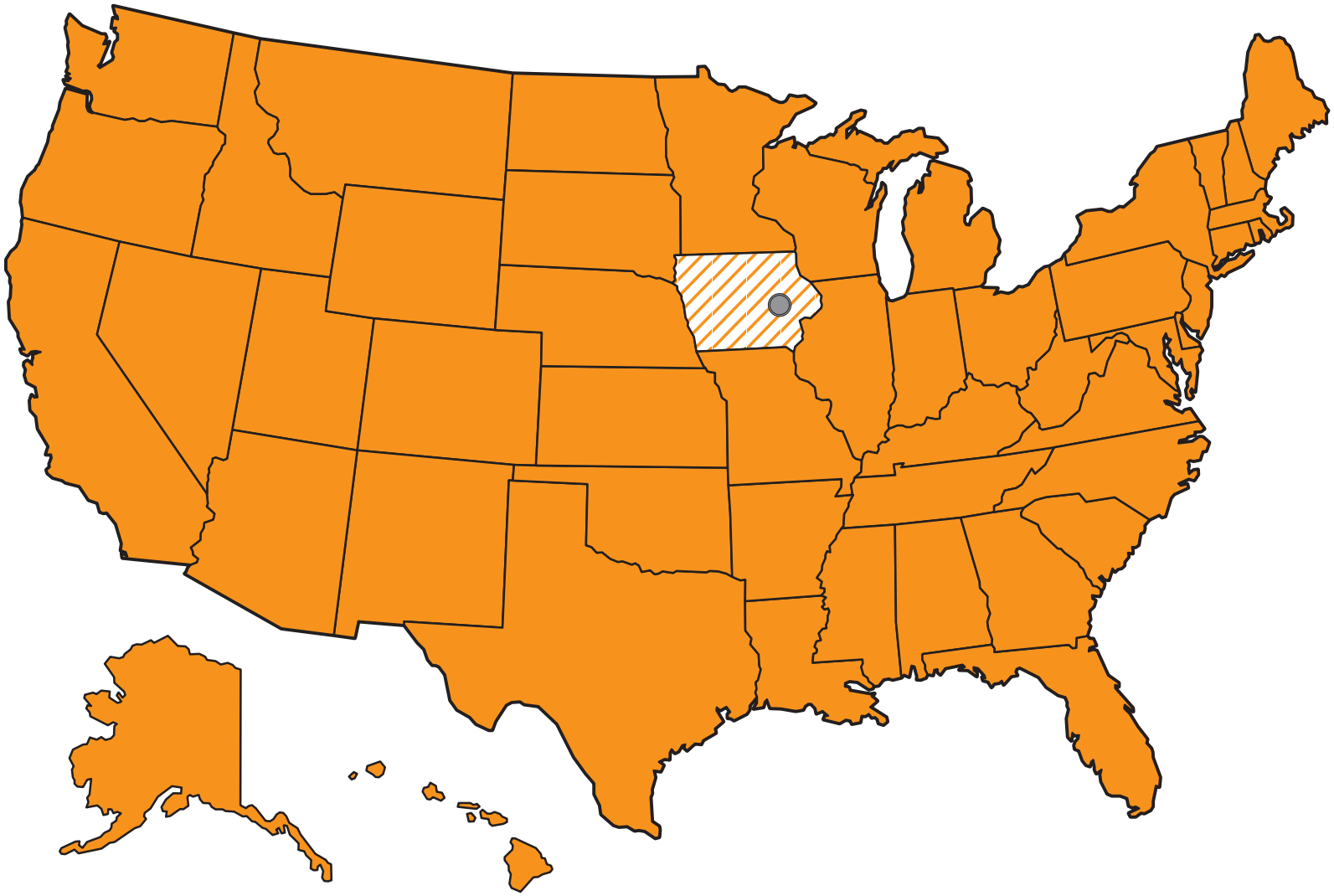
Document:Texas Jan 98
Last saved on:Tue, Jan 20, 1998 13:17:05

Formal Verification at Rockwell Collins

David Greve
Dr. David Hardin
Dr. Matthew Wilding

Rockwell Collins Advanced Technology Center
Cedar Rapids, IA

Rockwell Avionics
Collins



Rockwell Avionics
Collins

Two Active Formal Verification Research Projects

Partition Management Unit ("PMU") Verification

- DARPA–sponsored effort with SRI CSL
- verify that a PMU–equipped system supports task isolation

Executable formal models to support processor development

- internally–supported effort
- construct a processor model that supports simulation and analysis

Both these projects involve computer systems modeling using the logics provided by automated theorem proving tools.

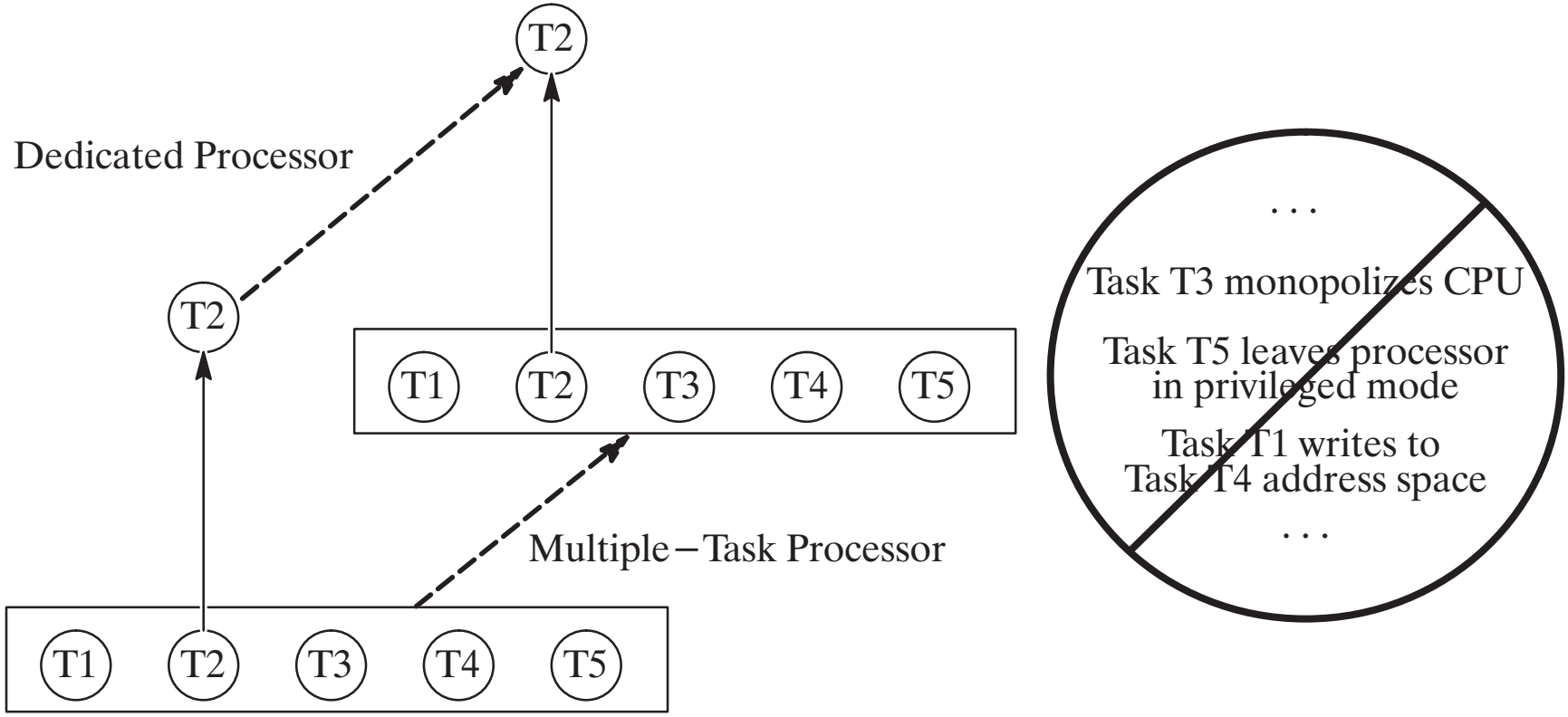
PMU Verification Motivation

Applications are partitioned in part to lessen high verification costs of avionics.

Increased computer system performance encourages function integration.

How can we build a system supporting multiple applications that allows for independent verification?

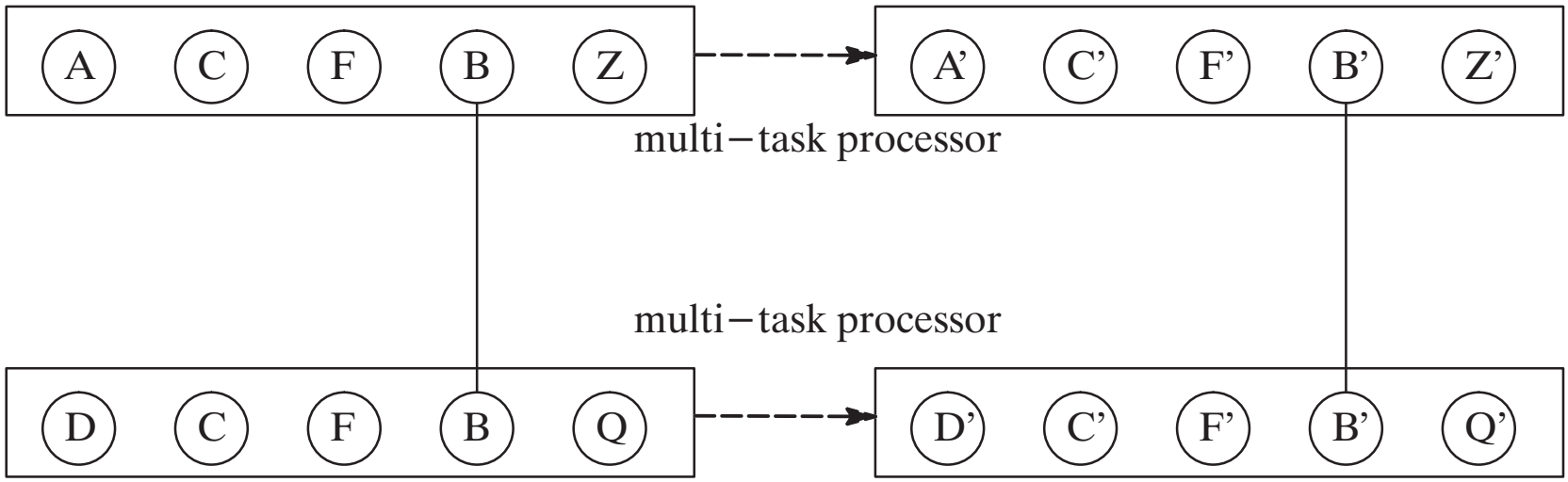
Virtual Machines



Approach used by Rushby and Bevier to describe isolation.

Doesn't provide the "contract" our developers need to develop their applications separately.

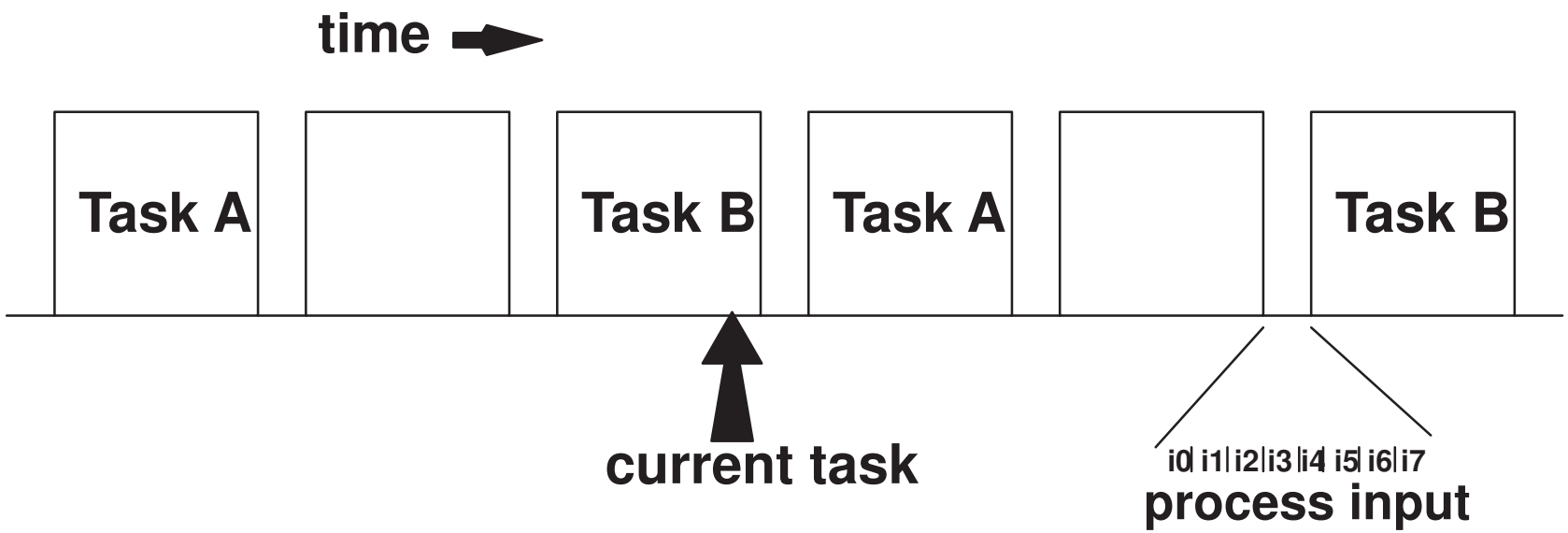
Invariant Performance



A partition has ***invariant performance*** if its behavior is a function of its initial state and its inputs. We will provide as a partitioning contract that each of the partitions supported by our system has invariant performance, a theorem we call the ***invariant performance theorem***.

This theorem allows separate verification of applications.

Boxcar Architecture



We have designed an architecture that is a cross between event−driven and time−triggered systems. This approach appears to be acceptable for avionics applications and (if implemented carefully) has invariant performance.

PVS

"PVS is a verification system: an interactive environment for writing formal specifications and checking formal proofs."

– Tutorial Introduction to PVS

PVS was built by and is supported by the SRI Computer Science Lab.

We have used PVS experimentally to model processors and derive symbolic results from microcode execution.

- AAMP5, AAMP–FV
- JEM1 symbolic simulation
- robust proofs about prototype models

Where We Are

Invariant performance identified as essential property

Boxcar architecture designed

PVS computer system reasoning developed

- PVS improvements
- toy processor model code analysis automated a la Nqthm code proofs
- simple AAMP–FV assembly programs automated
- currently building support for initial PMU–equipped system analysis

Executable Formal Models for Processor Development

Our (small) group has built processors for 25 years, including the JEM1, the world's first Java direct execution machine. Two JEM1 models are cycle−based simulators.

C model

- interface to simulation environment tool
- very fast

PVS logic model

- supports symbolic simulation
- 3 early JEM1 walkthrus used results
- couldn't keep up with development process
- model validation issues

A single model for simulation and analysis would have several advantages

- simulation and analysis can leverage each other
- potential simulation and symbolic simulation integration
- It *should* support fast symbolic simulation and formal verification.
- It *must* have extremely good performance as a simulator.

ACL2

"ACL2 is both a programming language in which you can model computer systems and a tool to help you prove properties of those models."

– ACL2 home page

"The ACL2 model runs several times faster than the compiled SPW model and hence is a potentially valuable debugging tool in its own right."

– Brock, Kaufmann, Moore (FMCAD '96)

We hope that ACL2 can support models of our processors that can be used for both simulation and analysis.

A Small Processor Prototype

We constructed C and ACL2 versions of a model for "tiny", a simple processor.

We developed a simple C driver to run these models. (We run ACL2 using GCL, and used GCL to compile the ACL2 model.)

The ACL2 model runs 132x slower on a simple program.

We optimized ACL2 execution by eliminating all garbage collection in the execution of the model. Specifically this meant

- **optimizing arithmetic operations**
- **using arrays instead of lists**
- **speeding up the array operations**

Optimizing Arithmetic Operations

ACL2 provides a facility for declaring and verifying that expressions return bounded integer values that can be represented efficiently.

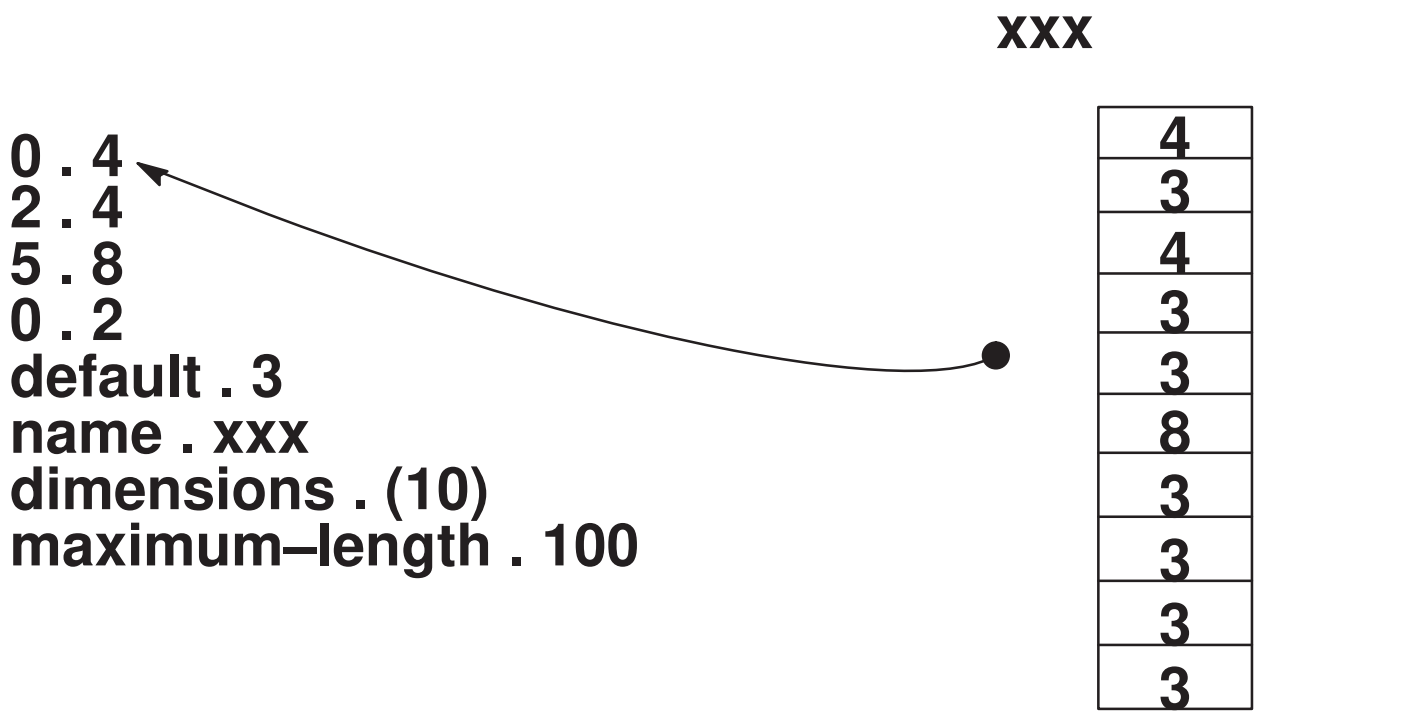
```
(defmacro bv32 (x) '(the (signed–byte 32) ,x))

... (bv32 (1– (bv32 n))) ...
```

This leads to extra proof obligations and efficient compiled code.

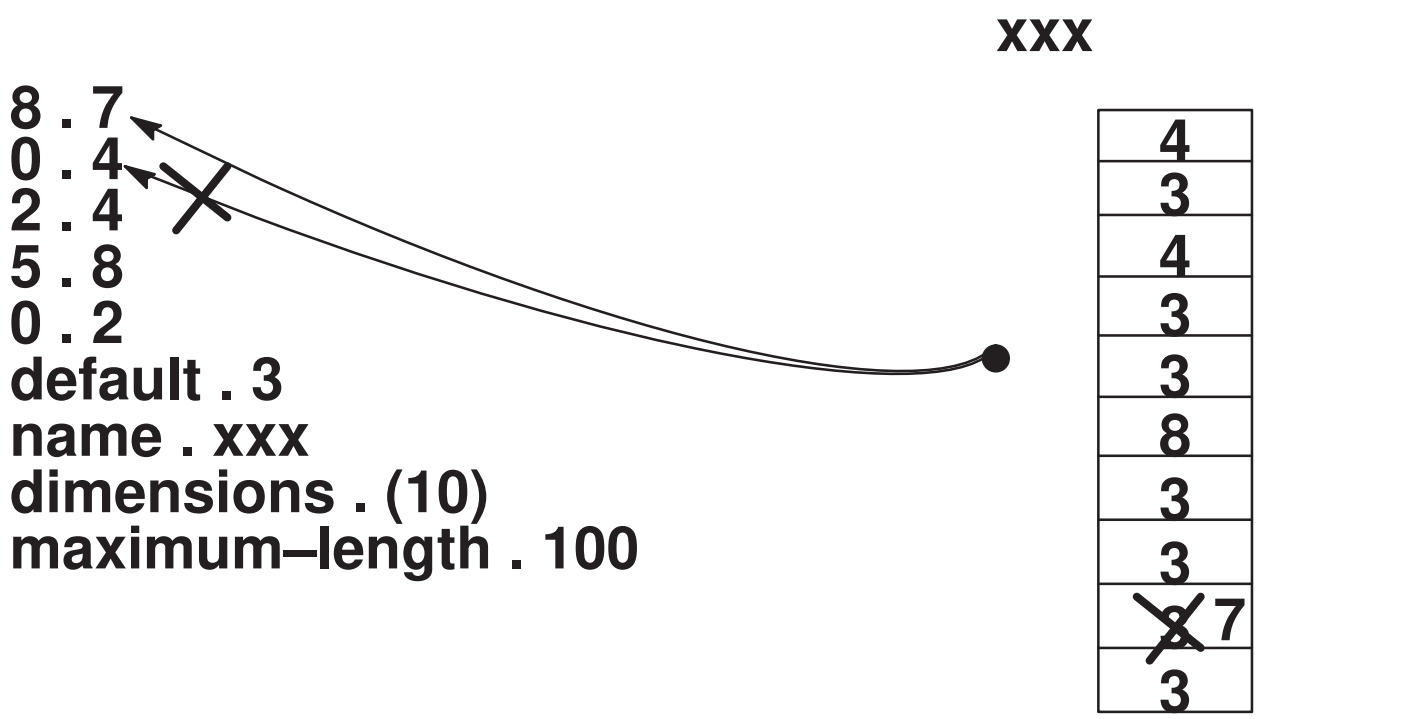
Unfortunately, we see no efficient way to represent 32−bit arithmetic operations in ACL2. We introduce 32−bit arithmetic in ACL2 and replace "under the hood" the compiled ACL2 operations with fast C arithmetic.

```
(defun +bv32 (x y)
  (declare (type (signed–byte 32) x y))
  (bv32 (nat–to–int32 (logand (+ (int–to–nat32 x) (int–to–nat32 y))
                              (1– (expt 2 32))))))
```

Standard ACL2 Arrays

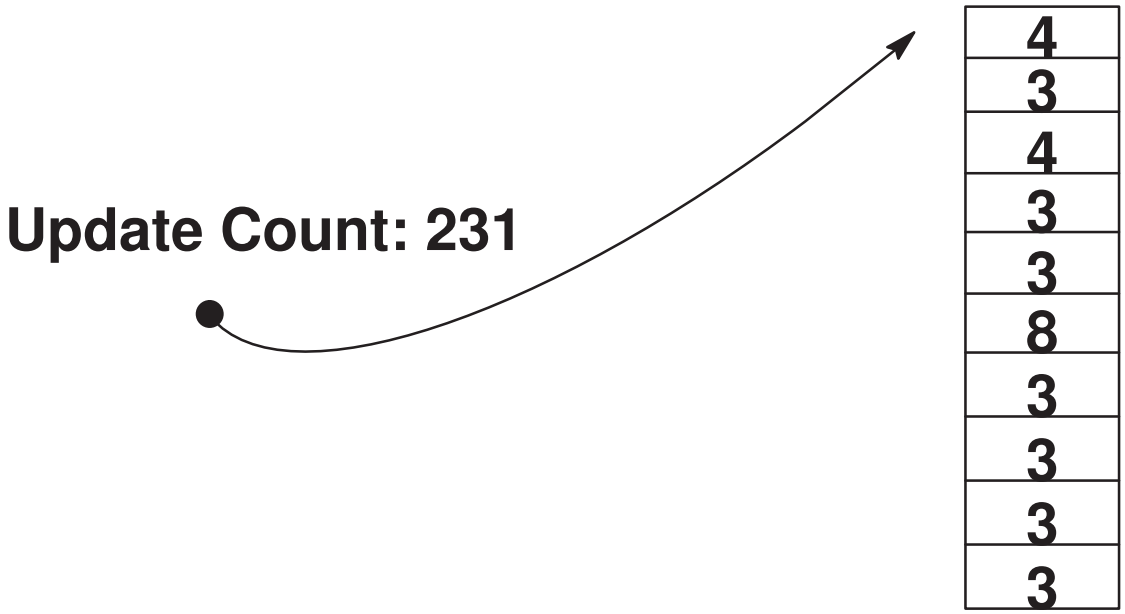


ACL2 arrays provide relatively fast array operations. Operations on stale arrays or arrays created without the special ACL2 array functions, use the list representation.



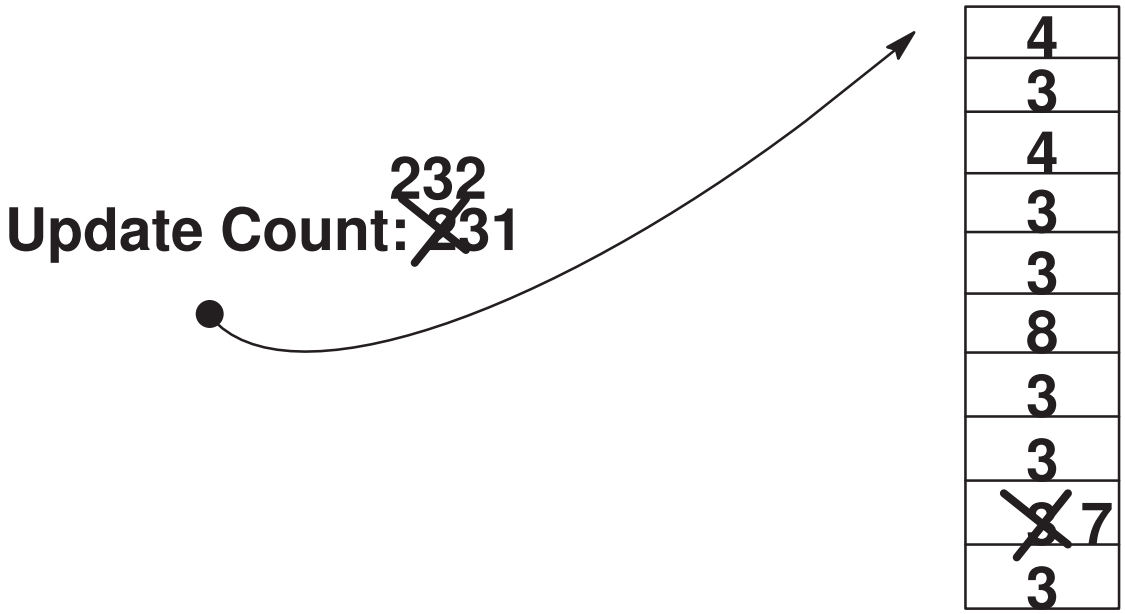
ACL2 array **(and under the hood)**

“Fast Arrays”

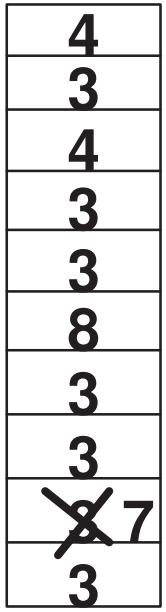


We replaced the special ACL2 array functions with our own versions written in C.

Stale array updates and references signal an error. Functions other than ACL2 array operations must not be applied to fast arrays.


Update Count: 232 231
4
3
4
3
3
8
3
3
7
3

“Inlined Arrays”



A third approach is to replace the ACL2 array functions with inlined C array functions. This implementation does not check for staleness and assumes only the array functions are used with arrays.

Correctness of Optimizations

We need our ACL2 models to run quickly and correctly. How can we justify the optimizations we have made?

Arithmetic operation optimizations partly justified.

- **Arithmetic optimizations proved within ACL2.**
- **Implementation of 32–bit operations replaced.**

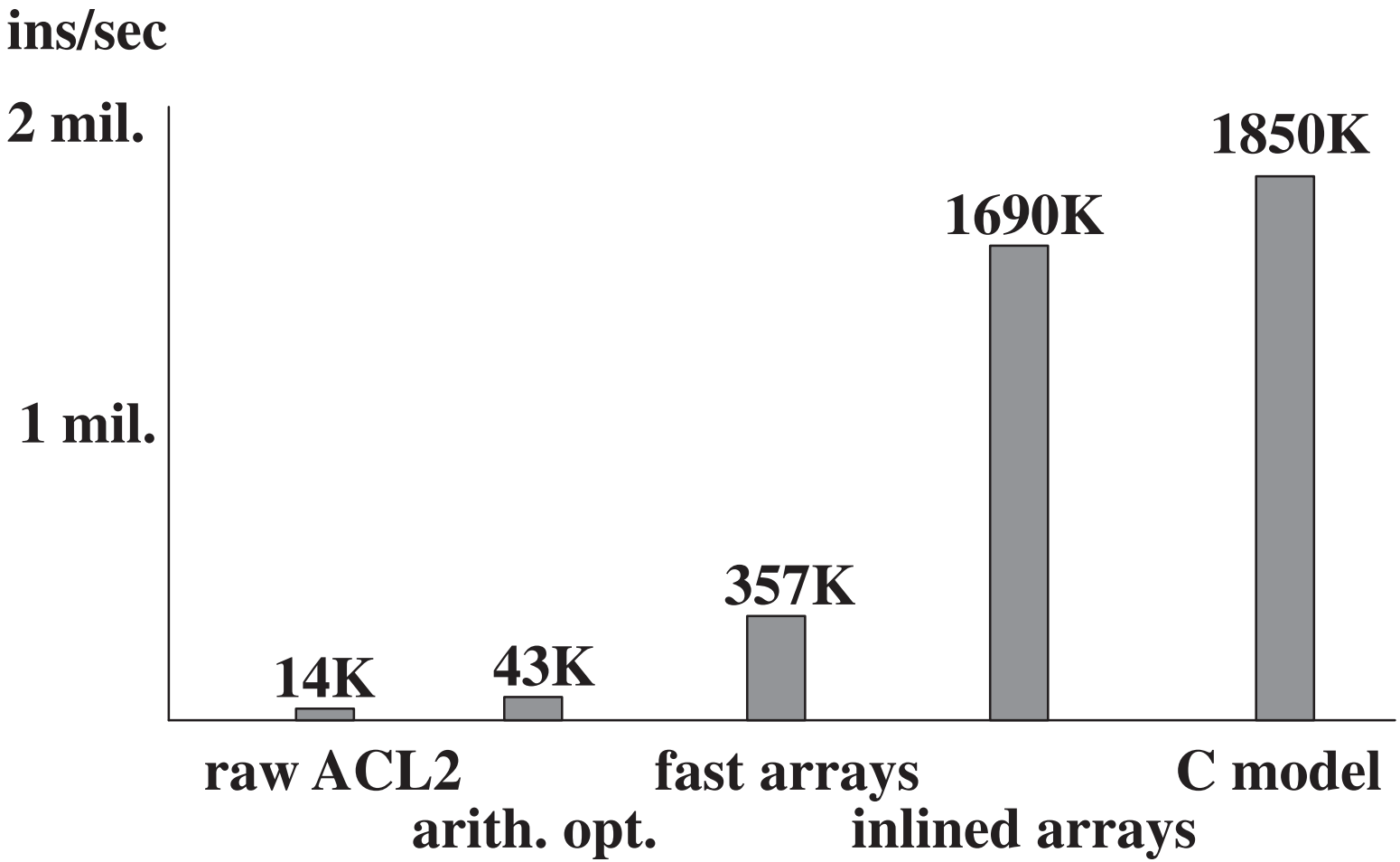
Array optimizations can not currently be justified within ACL2.

- **model needs to be validated anyway, but...**
- **ACL2 should support checks for array optimizations to allow quick execution of models.**

Tiny Execution Benchmarks

- C model
- raw ACL2
- ACL2 with arithmetic optimizations
- ACL2 with arithmetic optimizations and fast arrays
- ACL2 with arithmetic optimizations and inlined array references

Tiny Benchmark



ACL2 2.0 on GCL 2.1 on a Sparc 5

Rockwell Avionics
Collins

Future Work

PMU Verification

- **More realistic designs**
- **invariant performance on PMU–equipped boxcar architected system**
- **examination of relationship between partitioning and fault–tolerance**

Executable Formal Models

- **JEM1 model building using these techniques started this month**
- **simple proofs about prototype**
- **correctness of array optimizations**
- **integration of simulation into JEM design process**
- **integration of symbolic simulation into JEM design process**
  - regression testing
  - interface issues
- **formal connection to JVM, or other high–level specification**