Model Checking Distributed Software

Sagar Chaki
September 19, 2014

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Model Checking and Me

1997 : Ed visits IIT Kharagpur

- Just finished 2nd year undergrad
- Couldn't understand most of the talks

1998-99: Final year UG

- Developed symbolic model checker, read papers, found a typo

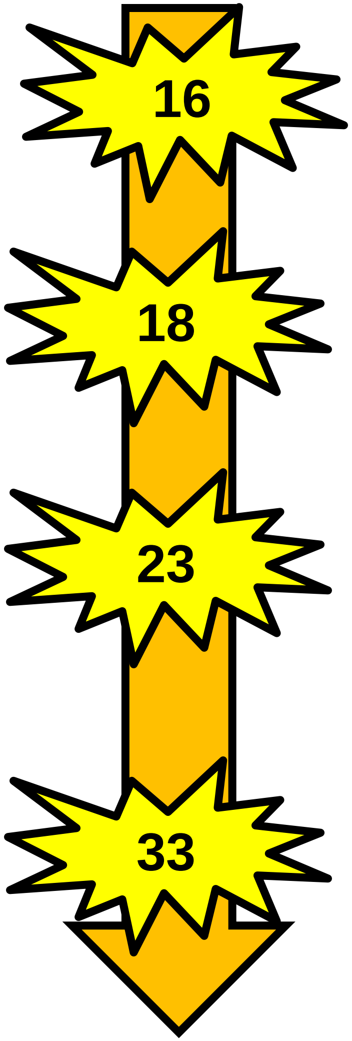
1999-2004: CMU PhD

- Some of the best years of my life
- Did not coin the term "CEGAR"
- Ed, Martha, Pankaj, Somesh, Orna, Helmut, Daniel, Joel, …

2004-Present: SEI

- Verifying Cyber Physical Systems
- Meetings & Lunch in my office responding to email

EMC (Era of Model Checking)



Motivation

Distributed algorithms have always been important

- File Systems, Resource Allocation, Internet, …

Increasingly becoming safety-critical

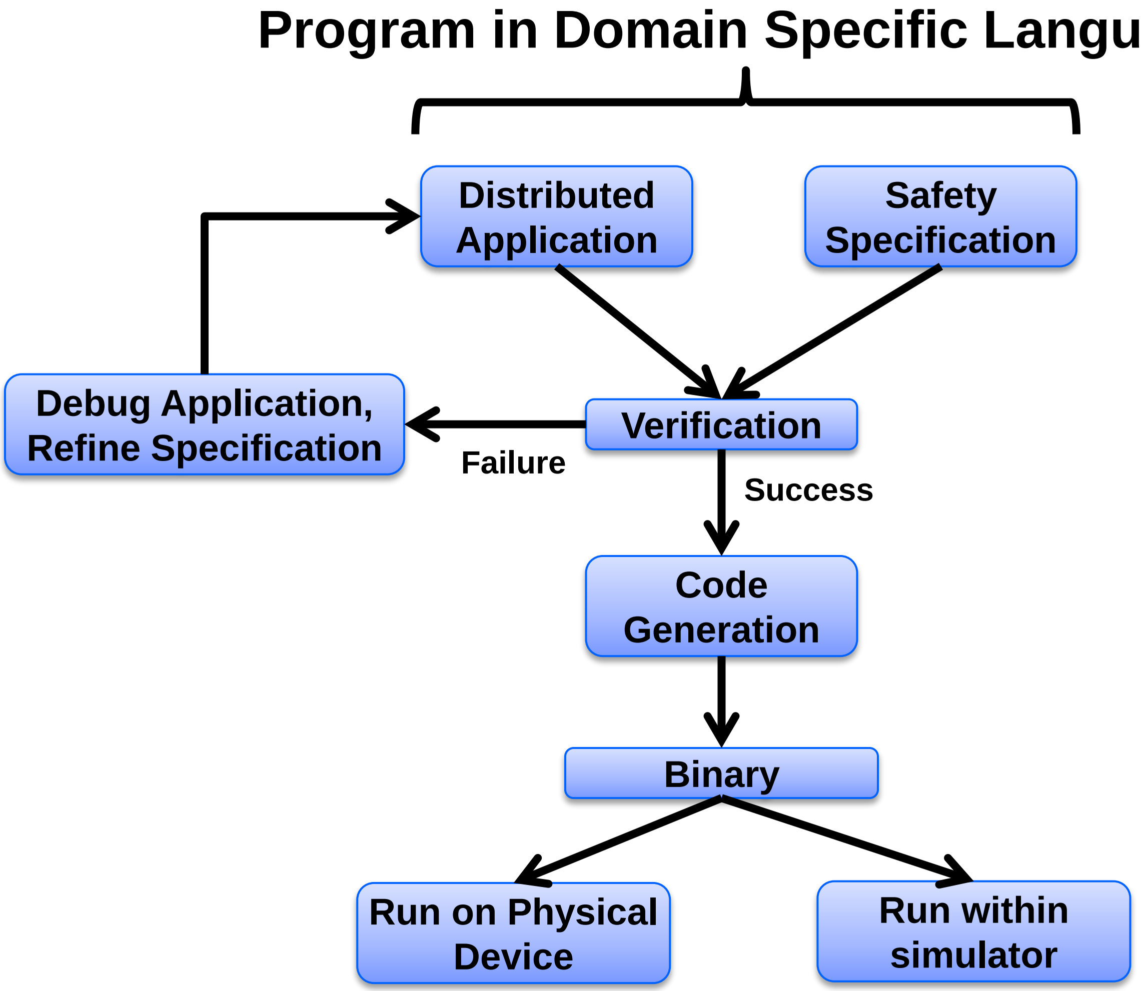
- Robotic, transportation, energy, medical

Prove correctness of distributed algorithm implementations

- Pseudo-code is verified manually (semantic gap)
- Implementations are heavily tested (low coverage)

Model-Driven Verifying Compilation of Synchronous Distributed Applications, Sagar Chaki, James Edmondson, Proc. of MODELS 2014, to appear

Approach : Verification + Code Generation



Program in Domain Specific Language

Distributed Application

Safety Specification

Verification

Debug Application, Refine Specification

Failure

Success

Code Generation

Binary

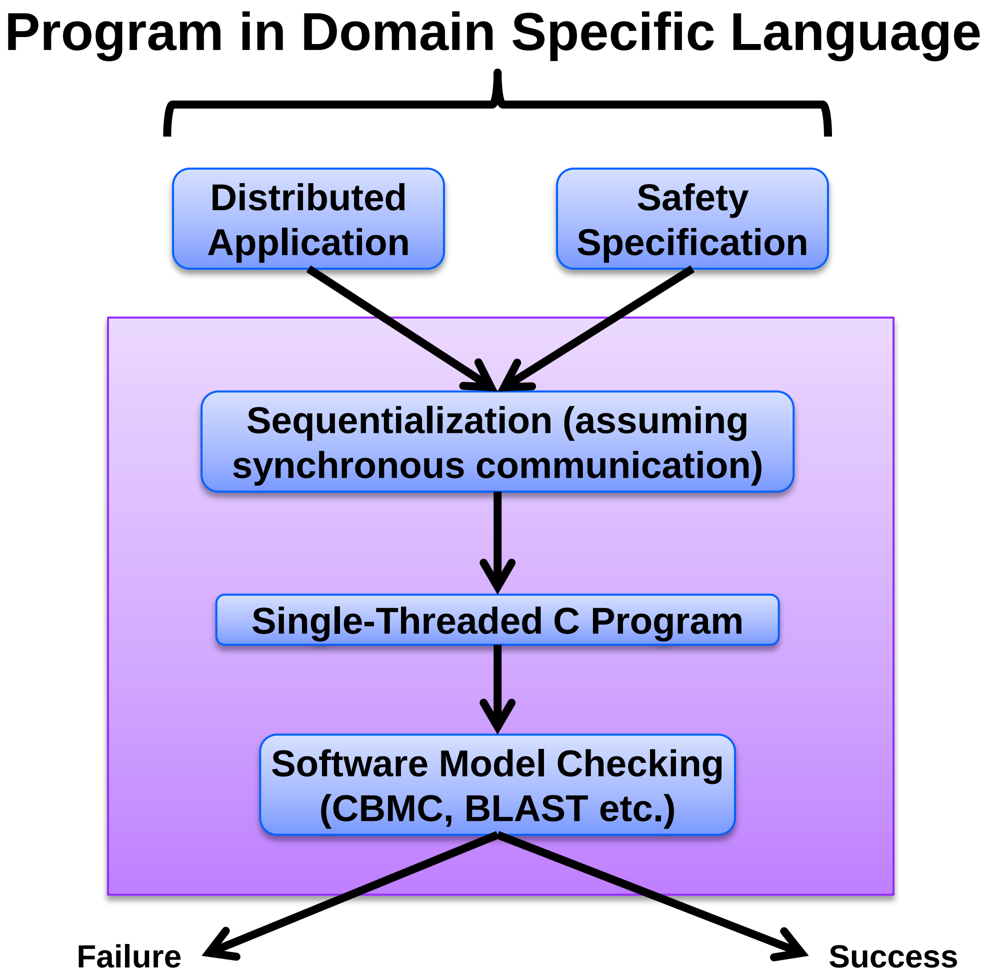
Run on Physical Device

Run within simulator

The Verifying Compiler: A Grand Challenge for computing research

Tony Hoare

Verification


Program in Domain Specific Language

Distributed Application

Safety Specification

Sequentialization (assuming synchronous communication)

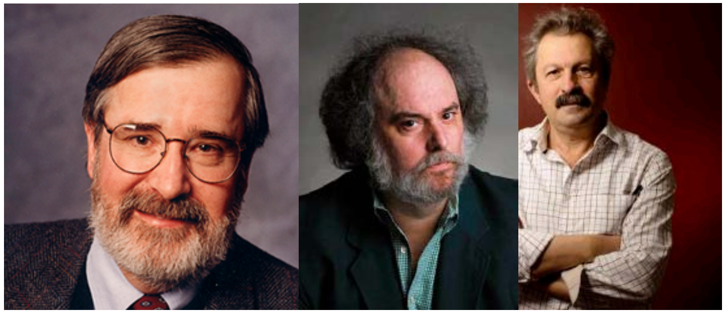
Single-Threaded C Program

Software Model Checking (CBMC, BLAST etc.)

Failure

Success

Model Checking



Automatic verification technique for finite state concurrent systems.

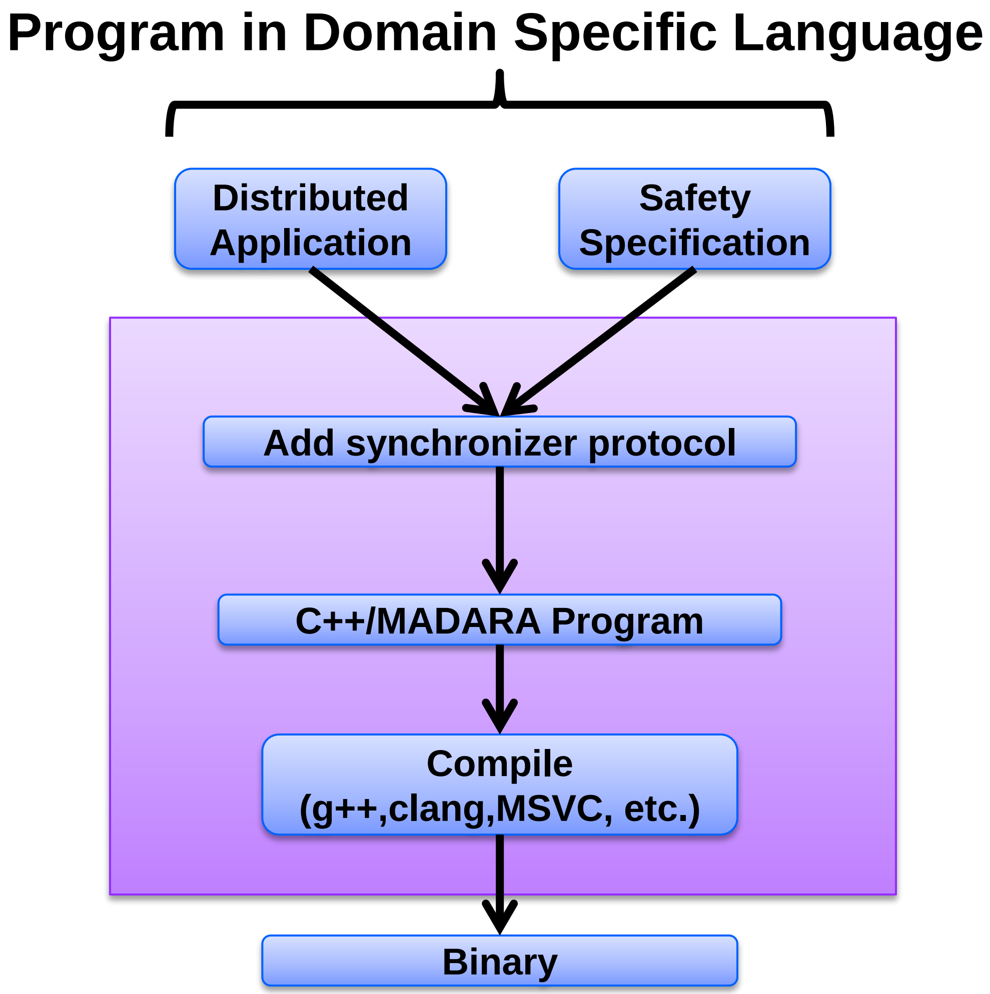
- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980's.
- ACM Turing Award 2007

Specifications are written in propositional temporal logic. (Pnueli 77)

- Computation Tree Logic (CTL), Linear Temporal Logic (LTL), …

Verification procedure is an intelligent exhaustive search of the state space of the design

Code Generation


Program in Domain Specific Language
Distributed Application
Safety Specification
Add synchronizer protocol
C++/MADARA Program
Compile (g++,clang,MSVC, etc.)
Binary

MADARA Middleware

A database of facts: $DB = Var \mapsto Value$

Node i has a local copy: DB_i

- update DB_i arbitrarily
- publish new variable mappings
  - Immediate or delayed
  - Multiple variable mappings transmitted atomically

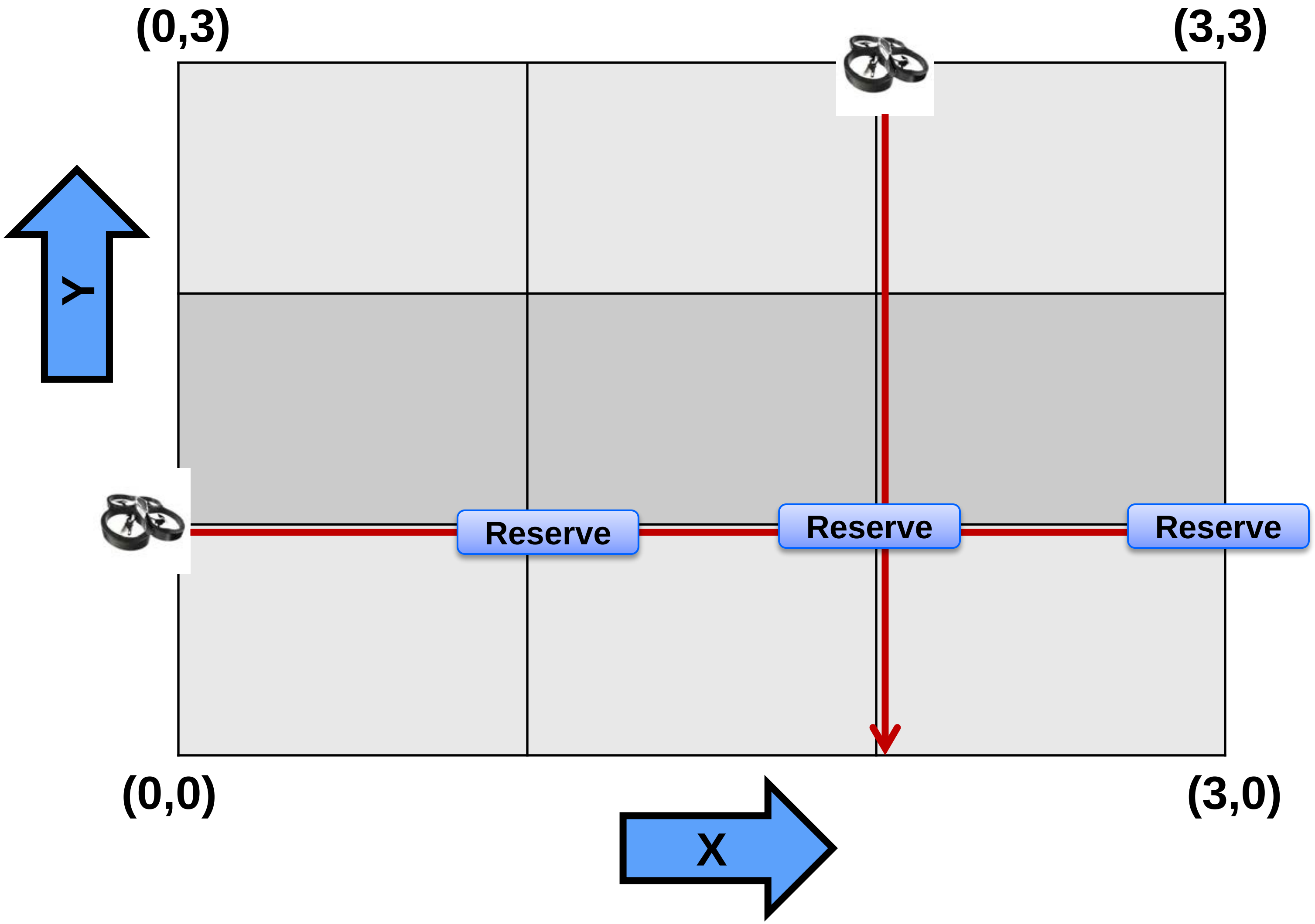
Implicit “receive” thread on each node

- Receives and processes variable updates from other nodes
- Updates ordered via Lamport clocks

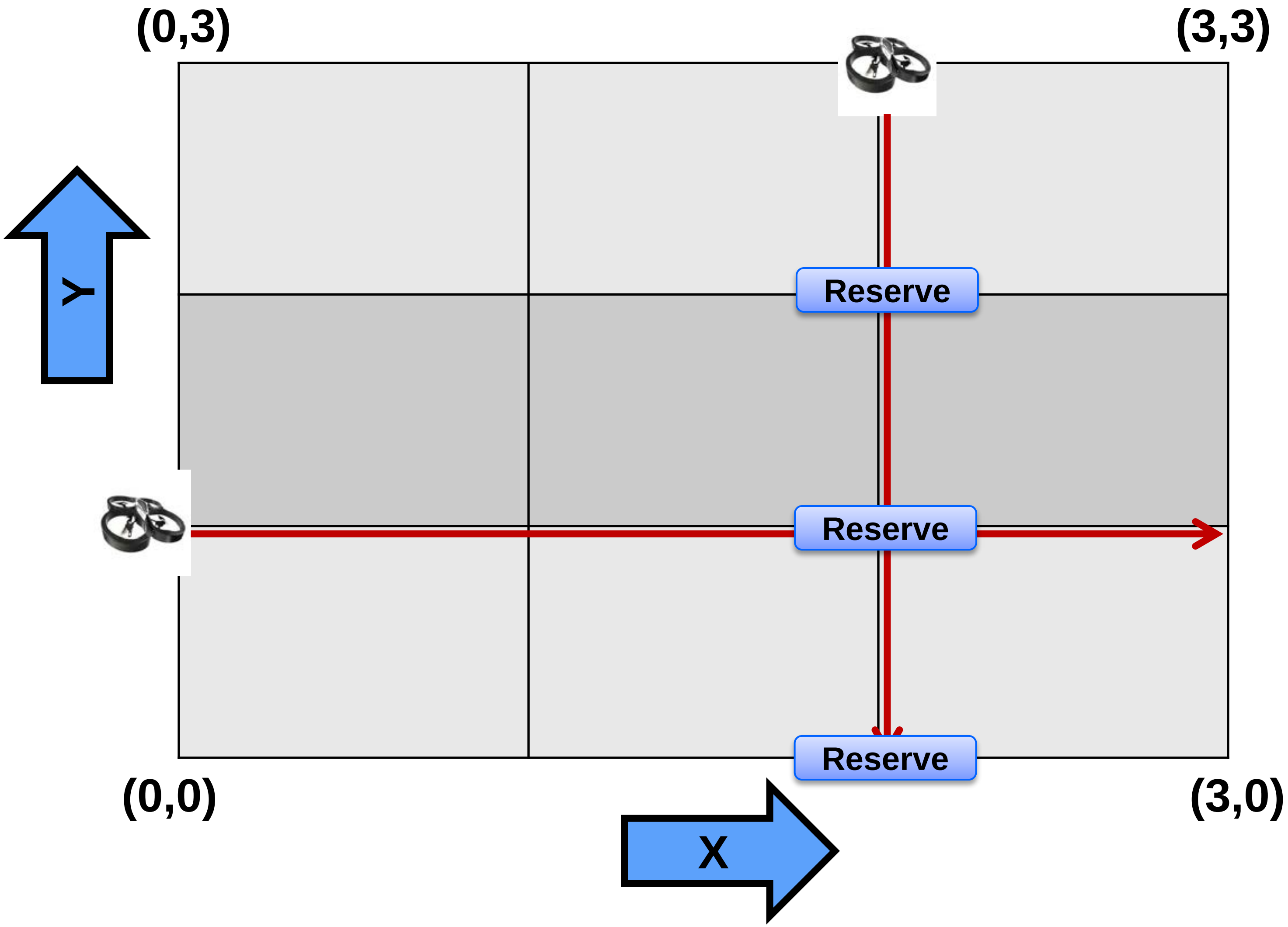
Portable to different OSes (Windows, Linux, Android etc.) and networking technology (TCP/IP, UDP, DDS etc.)

Case Study: Synchronous Collision Avoidance

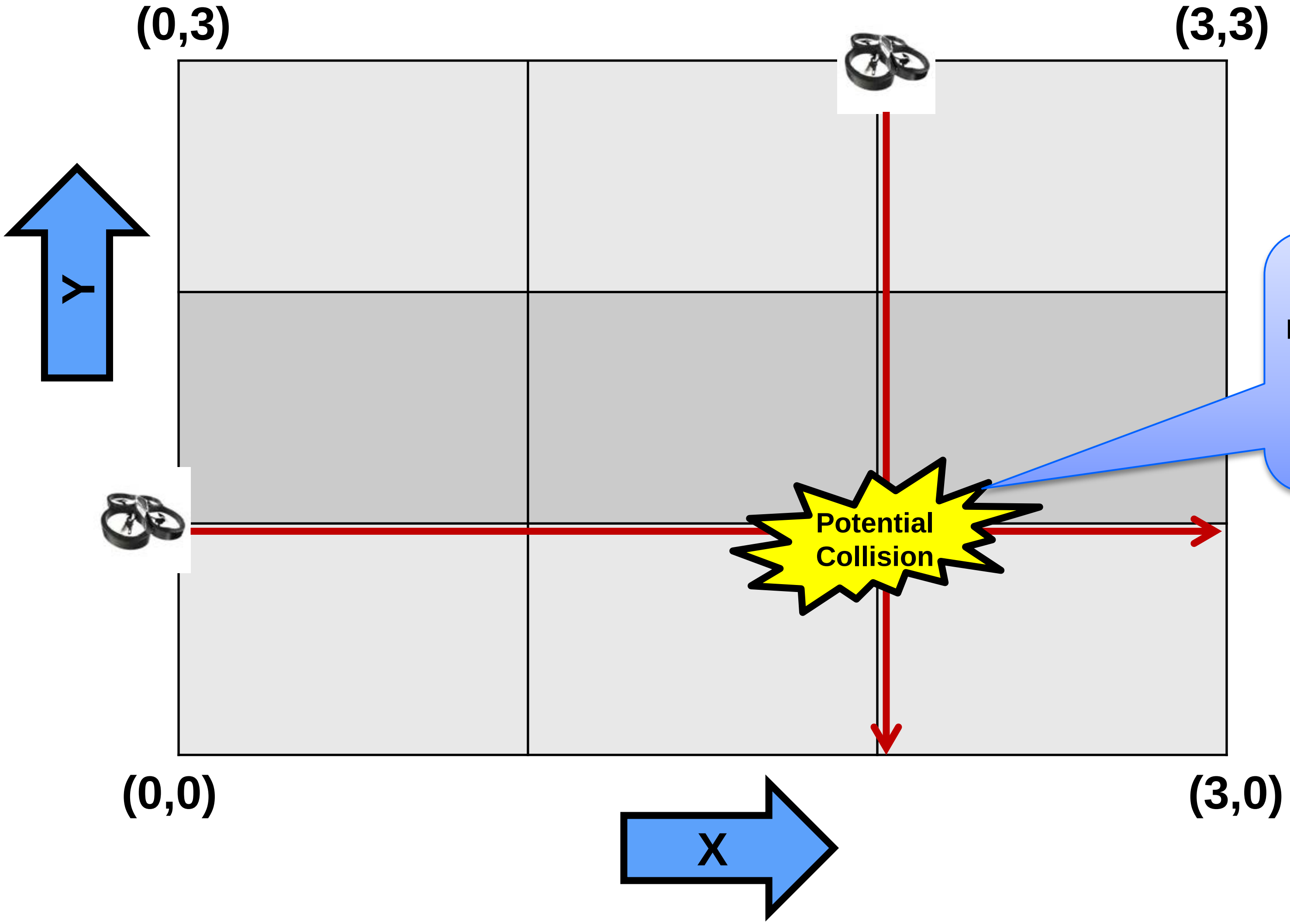
Example: Synchronous Collision Avoidance



Example: Synchronous Collision Avoidance

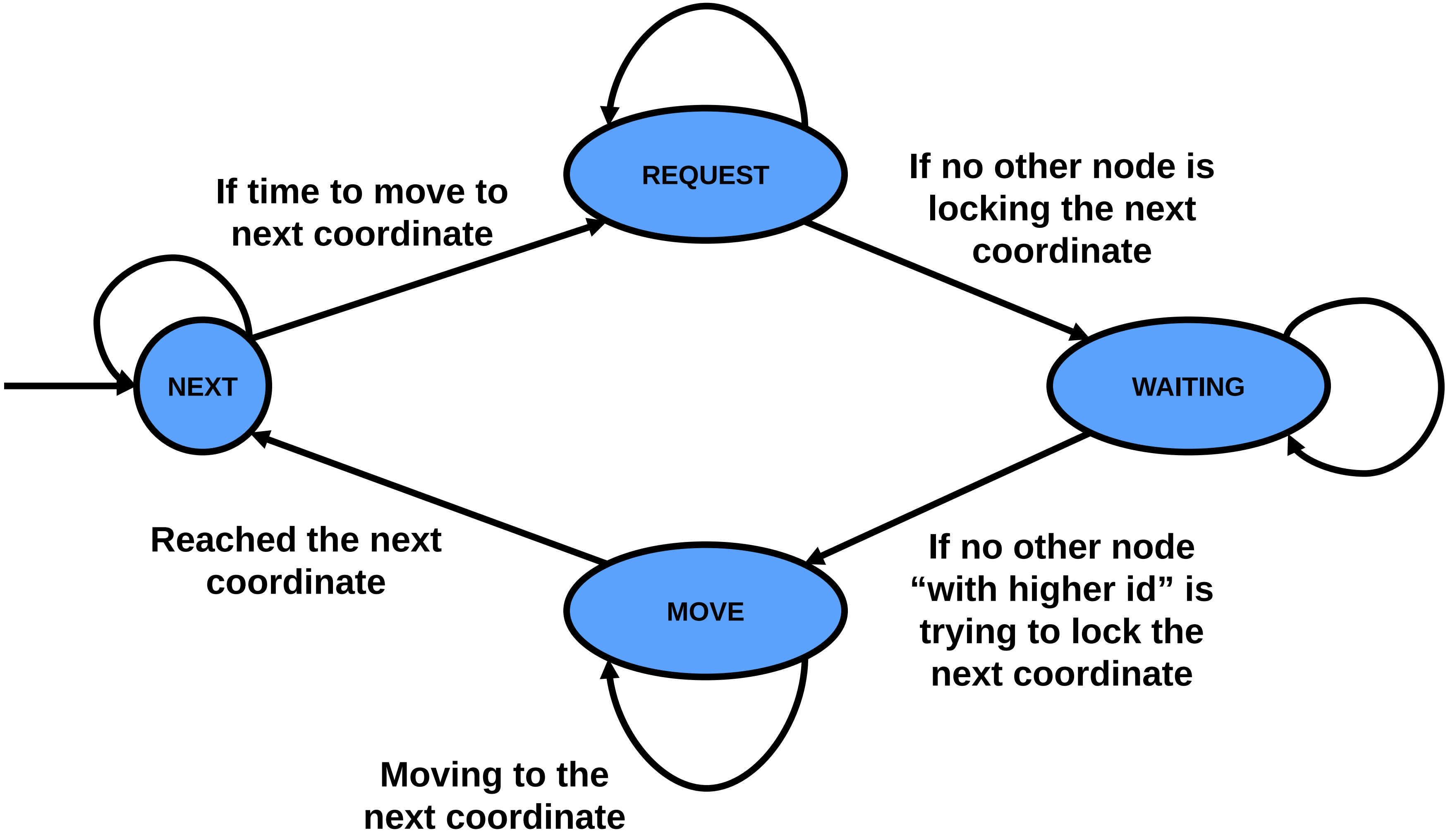


Example: Synchronous Collision Avoidance



Reservation Contention Resolved based on Node ID. No collision possible if no over-booking.

Collision Avoidance Protocol


REQUEST
If time to move to next coordinate
If no other node is locking the next coordinate
NEXT
WAITING
Reached the next coordinate
MOVE
If no other node “with higher id” is trying to lock the next coordinate
Moving to the next coordinate

Synchronous Collision Avoidance Code

```
MOC_SYNC;

CONST X = 4; CONST Y = 4;
CONST NEXT = 0;
CONST REQUEST = 1;
CONST WAITING = 2;
CONST MOVE = 3;

EXTERN int
MOVE_TO (unsigned char x,
         unsigned char y);

NODE uav (id) { … }

void INIT () { … }

void SAFETY { … }
```

```
NODE uav (id)
{
 GLOBAL bool lock [X][Y][#N];
 LOCAL int state,x,y,xp,yp,xf,yf;
 void NEXT_XY () { … }
 void ROUND () {
  if(state == NEXT) { …
   state = REQUEST;
  } else if(state == REQUEST) { …
   state = WAITING;
  } else if(state == WAITING) { …
   state = MOVE;
  } else if(state == MOVE) { …
   state = NEXT;
  } } }
```

```
INIT
{
 FORALL_NODE(id)
  state.id = NEXT;
  //assign x.id and y.id non-deterministically
  //assume they are within the correct range
  //assign lock[x.id][y.id][id] appropriately

 //nodes don't collide initially
 FORALL_DISTINCT_NODE_PAIR (id1,id2)
  ASSUME(x.id1 != x.id2 || y.id1 != y.id2);
}

SAFETY {
 FORALL_DISTINCT_NODE_PAIR (id1,id2)
  ASSERT(x.id1 != x.id2 || y.id1 != y.id2);
}
```

Synchronous Collision Avoidance Code

```
if(state == NEXT) {
  //compute next point on route
  if(x == xf && y == yf) return;
  NEXT_XY();
  state = REQUEST;
} else if(state == REQUEST) {
  //request the lock but only if it is free
  if(EXISTS_OTHER(idp,lock[xp][yp][idp] != 0)) return;
  lock[xp][yp][id] = 1;
  state = WAITING;
} else if(state == WAITING) {
  //grab the lock if we are the highest
  //id node to request or hold the lock
  if(EXISTS_HIGHER(idp, lock[xp][yp][idp] != 0)) return;
  state = MOVE;
}
```

```
else if(state == MOVE) {
  //now we have the lock on (xp,yp)
  if(MOVE_TO()) return;
  lock[x ][y][id] = 0;
  x = xp; y = yp;
  state = NEXT;
}
```

Tool Usage

Project webpage (http://mcda.googlecode.com)

- Tutorial (https://code.google.com/p/mcda/wiki/Tutorial)

Verification

- daslc --nodes 3 --seq --rounds 3 --seq-dbl --out tutorial-02.c tutorial-02.dasl
- cbmc tutorial-02.c (takes about 10s to verify)

Code generation & simulation

- daslc --nodes 3 --madara --vrep --out tutorial-02.cpp tutorial-02.dasl
- g++ …
- mcda-vrep.sh 3 outdir ./tutorial-02 …

Demonstration: Synchronous Collision Avoidance

Questions?

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM-0001371

Contact Information Slide Format

Sagar Chaki
Principal Researcher
SSD/CSC
Telephone: +1 412-268-1436
Email: chaki@sei.cmu.edu

Web
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations
Email: info@sei.cmu.edu

Telephone:	+1 412-268-5800
SEI Phone:	+1 412-268-5800
SEI Fax:	+1 412-268-6257