

Model-Driven Verifying Compilation of Synchronous Distributed Applications

Sagar Chaki, James Edmondson
October 1, 2014

MODELS'14, Valencia, Spain



Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 01 OCT 2014		2. REPORT TYPE N/A		3. DATES COVERED -	
4. TITLE AND SUBTITLE Model-Driven Verifying Compilation of Synchronous Distributed Applications				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Sagar Chaki James Edmondson				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, distribution unlimited					
13. SUPPLEMENTARY NOTES The original document contains color images.					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT SAR	18. NUMBER OF PAGES 24	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

Copyright 2014 Carnegie Mellon University

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.

This material has been approved for public release and unlimited distribution except as restricted below.

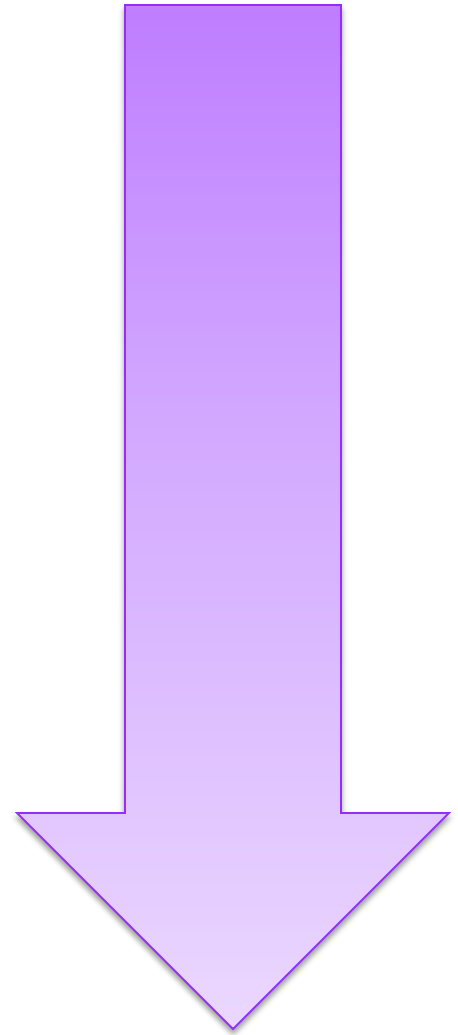
This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0001691



Outline

- Motivation
- Approach
- Sequentialization : SEQSEM & SEQDBL
- Examples
- Experimental Results
- Synchronizer Protocol : 2BSYNC
- Tool Overview & Demo
- Future Work



Motivation

Distributed algorithms have always been important

- File Systems, Resource Allocation, Internet, ...



Increasingly becoming safety-critical

- Robotic, transportation, energy, medical

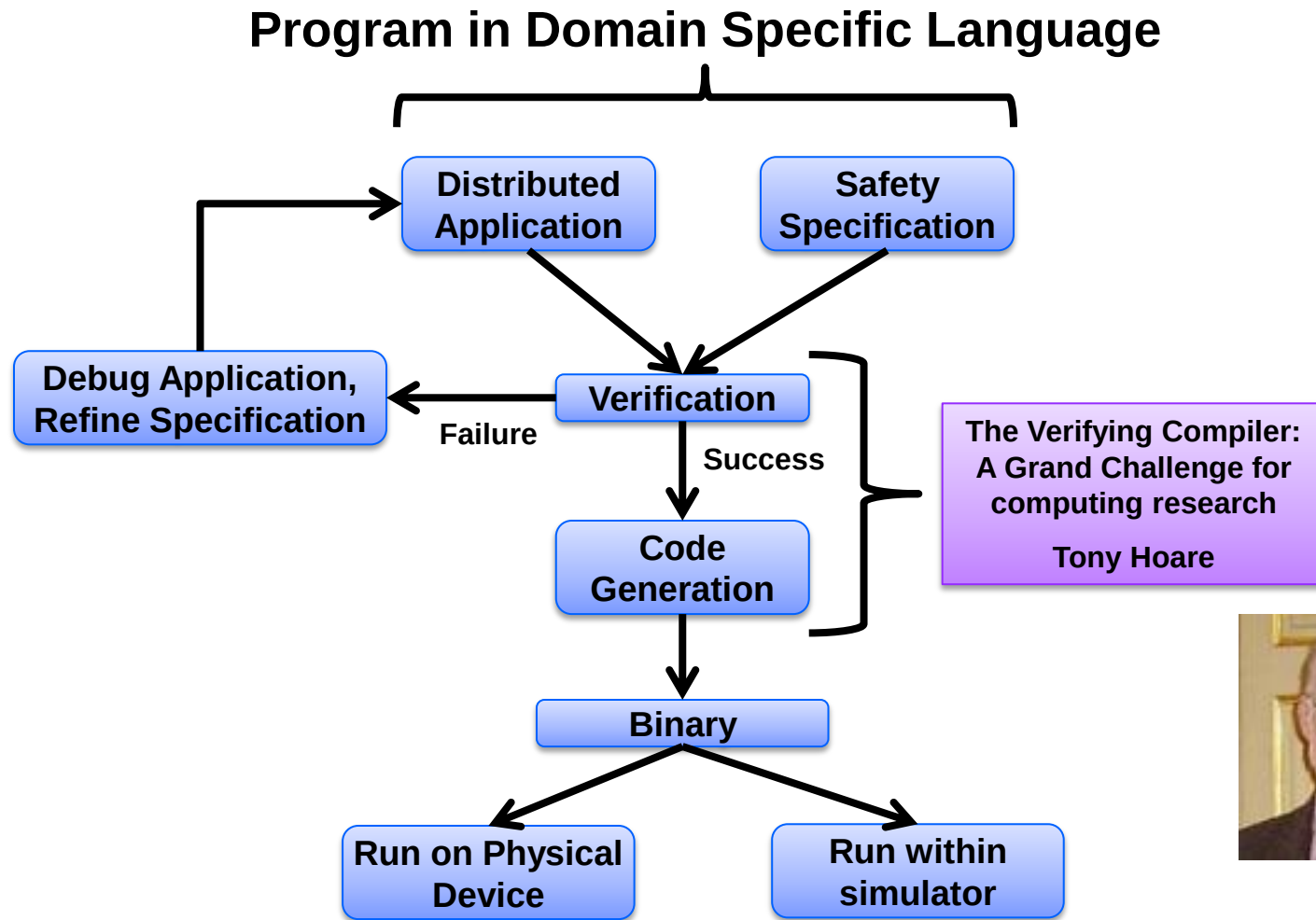


Prove correctness of distributed algorithm implementations

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

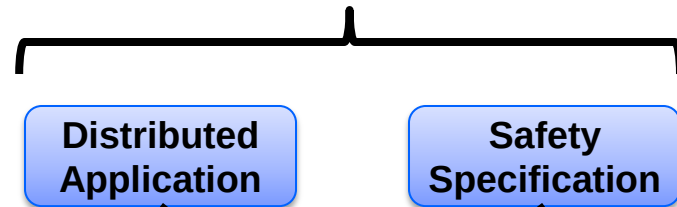


Approach : Verification + Code Generation



Verification

Program in Domain Specific Language



Sequentialization

Single-Threaded
C Program

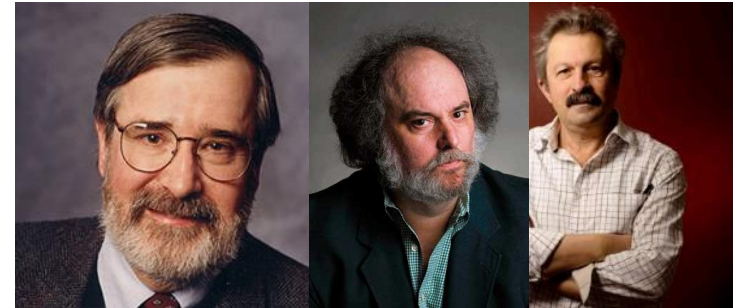
Software Model Checking
(CBMC, BLAST etc.)

Assume
Synchronous
Model of
Computation

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)

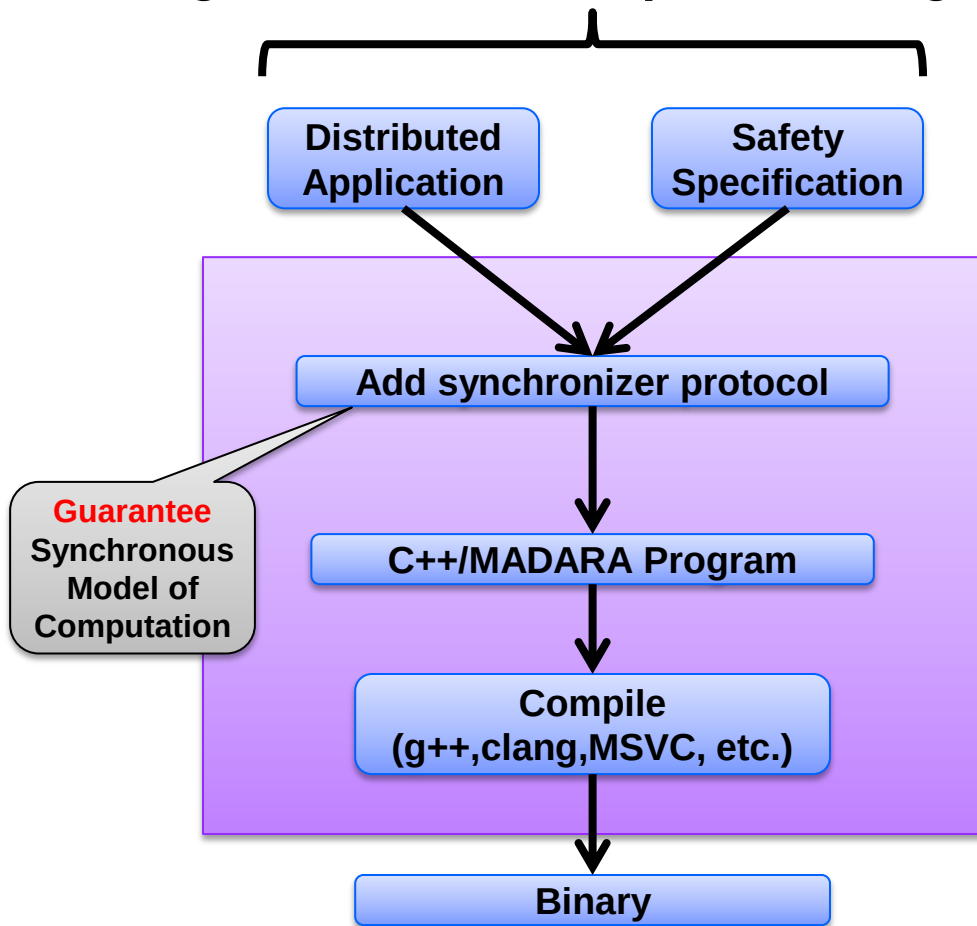
- Computation Tree Logic (CTL), Linear Temporal Logic (LTL), ...

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



Code Generation

Program in Domain Specific Language



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.)

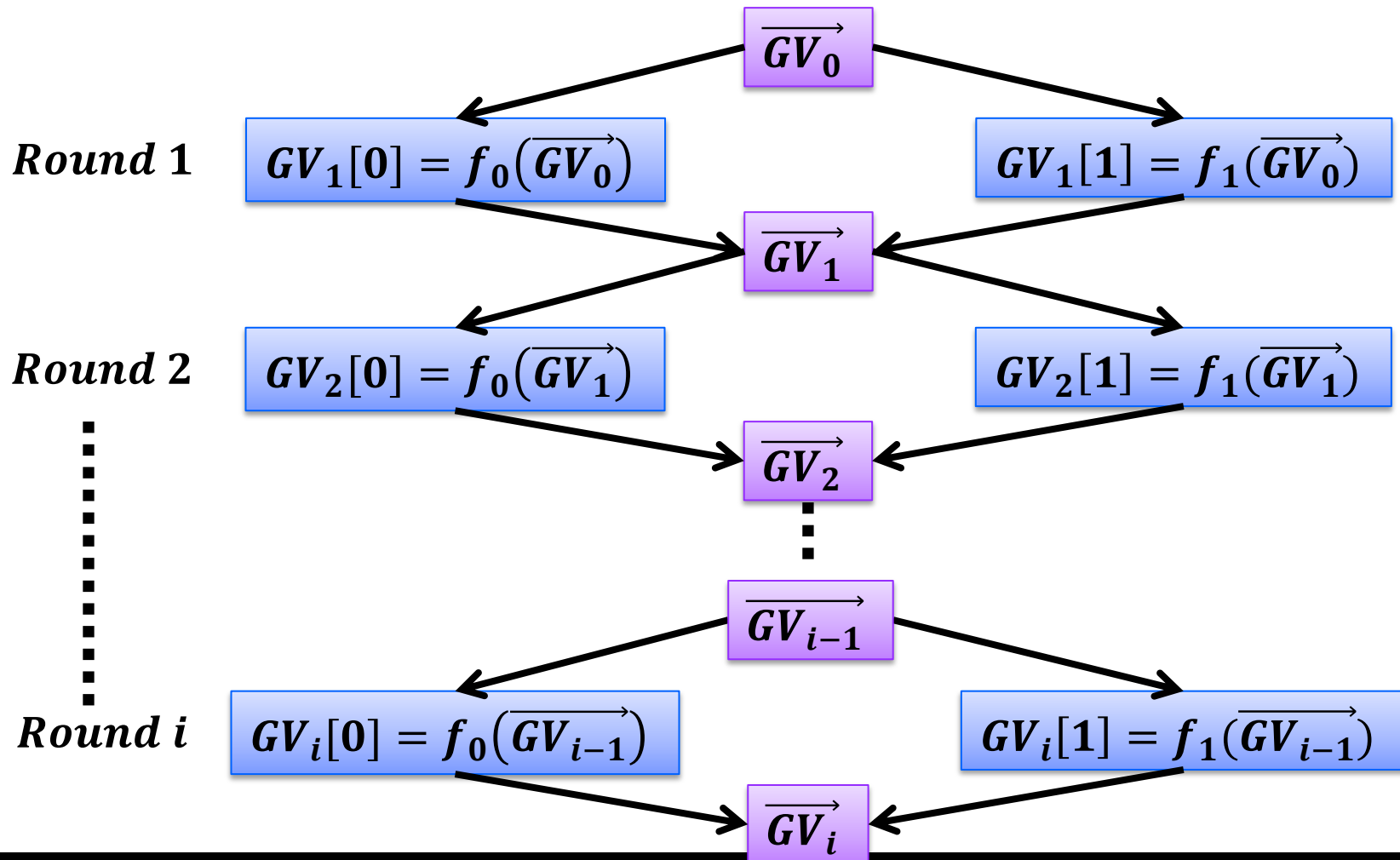


Synchronous Distributed Application (SDA)

Node 0 = $f_0()$

Shared Variables: $\overrightarrow{GV} = GV[0], GV[1]$

Node 1 = $f_1()$



SDA Verification

Program with n nodes : $P(n)$

- Each node has a distinct $id \in [1, n]$
- Array GV has n elements, $GV[i]$ writable only by node with id i
- Each element of GV is drawn from a finite domain

In each round, node with id id executes function ρ whose body is a statement

$stmt := skip \mid lval = exp \quad (assignment)$
 $\quad \mid ITE(exp, stmt, stmt) \quad (if, then, else)$
 $\quad \mid ALL(IV, stmt) \quad (iterate\ over\ nodes : use\ to\ check\ existence)$
 $\quad \mid \langle stmt^+ \rangle \quad (iteration\ of\ statements)$
 $lval := GV[id][w] \quad (lvalues)$
 $exp := \top \mid \perp \mid lval \mid GV[iv][w] \mid id \mid IV \mid \diamond (exp^+) \quad (expressions)$

Initial states and “ERROR” states of the program are define

- State $\equiv$ value assigned to all variables

Verification $\equiv$ decide if there is an execution of the program that starts in an initial state and ends in an ERROR state



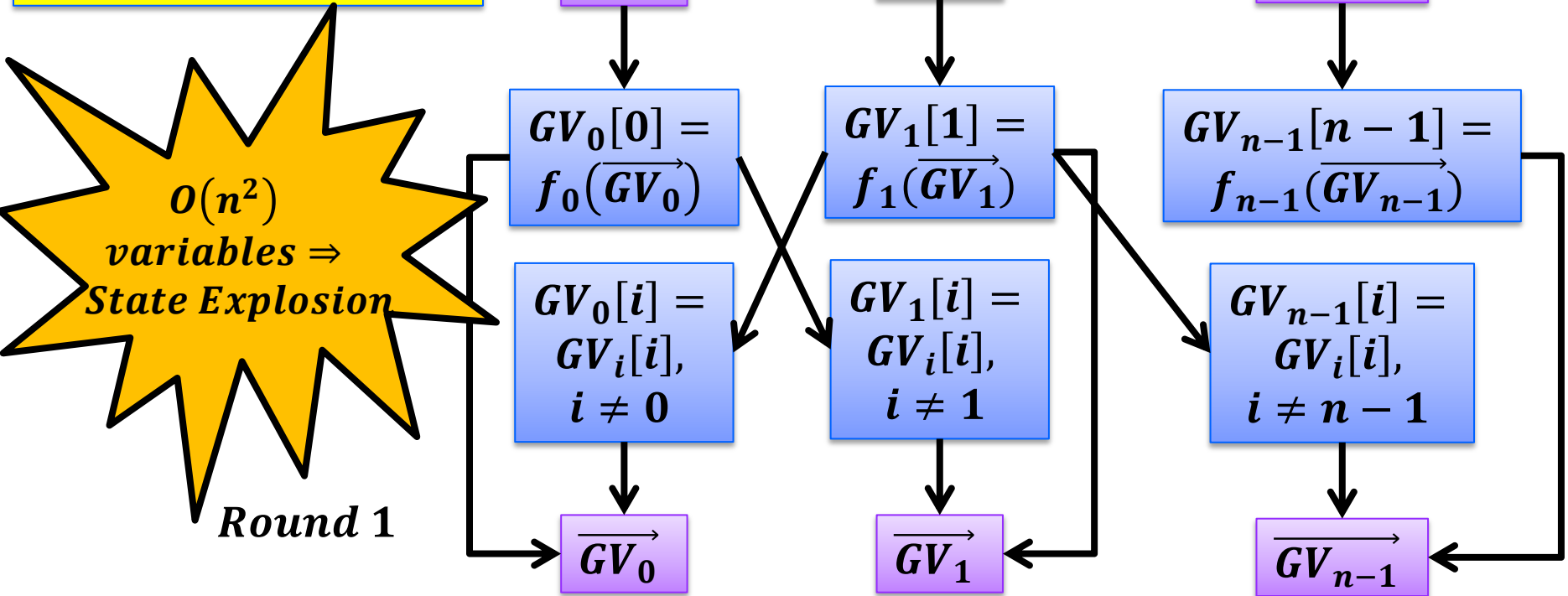
Semantic Sequentialization: SEQSEM

Node 0 = $f_0()$

Shared Variables: $\overrightarrow{GV} = GV[0], GV[1]$

Node 1 = $f_1()$

Assume n nodes
Use n copies of $\overrightarrow{GV}$



Operations have independence $\Rightarrow$ reordered sequentially.



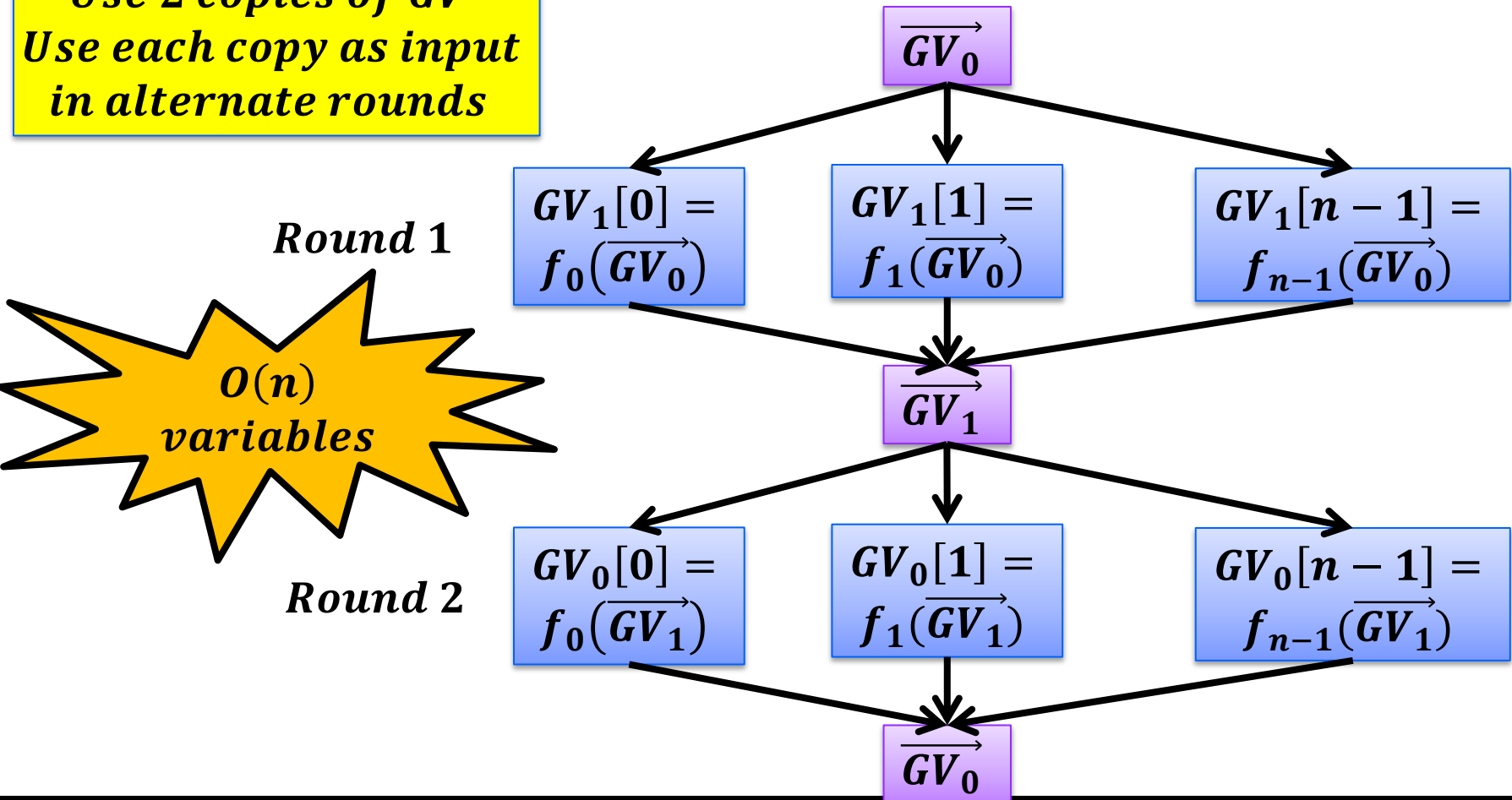
Double Buffering Sequentialization: SEQDBL

Node 0 = $f_0()$

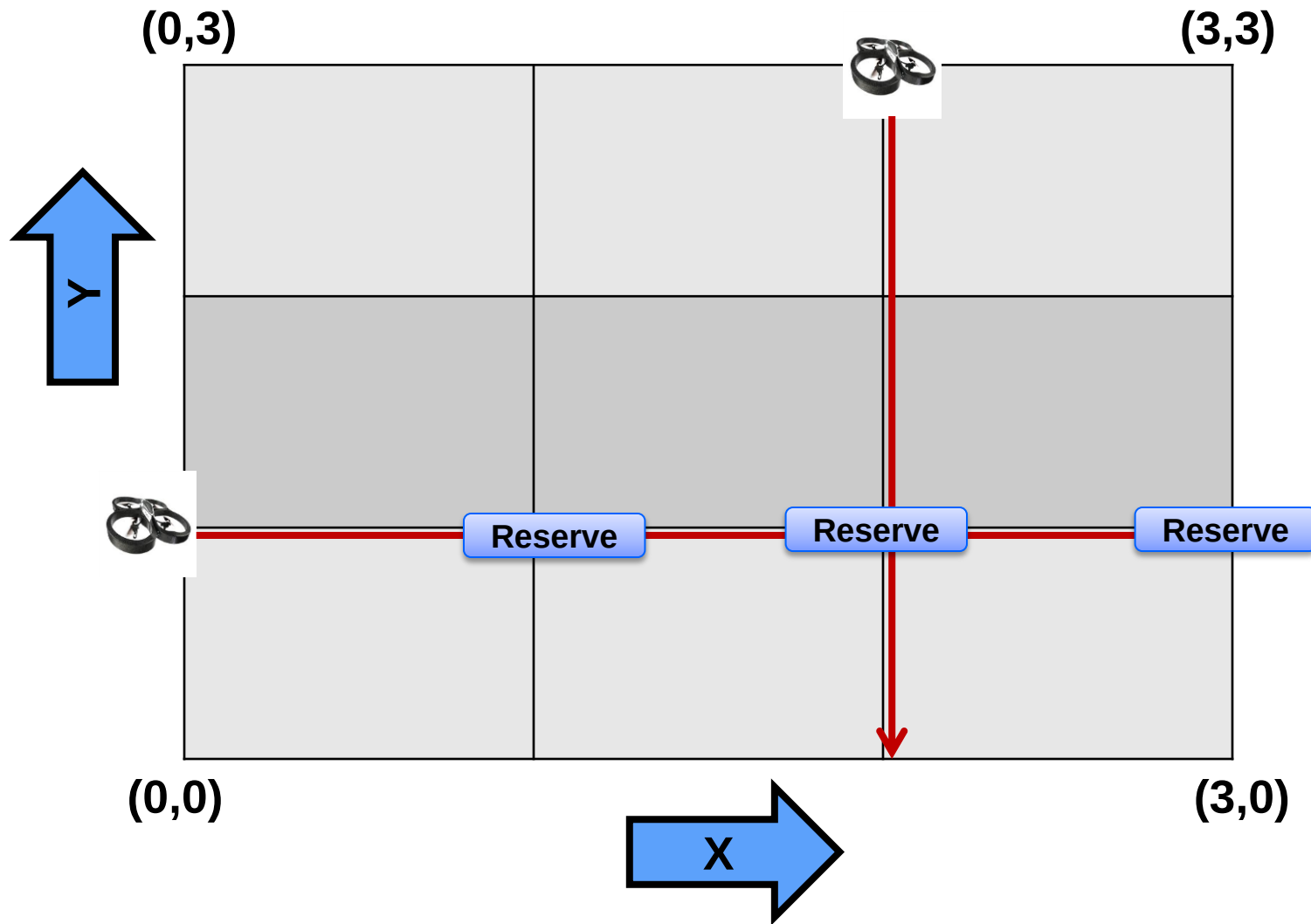
Shared Variables: $\overrightarrow{GV} = GV[0], GV[1]$

Node 1 = $f_1()$

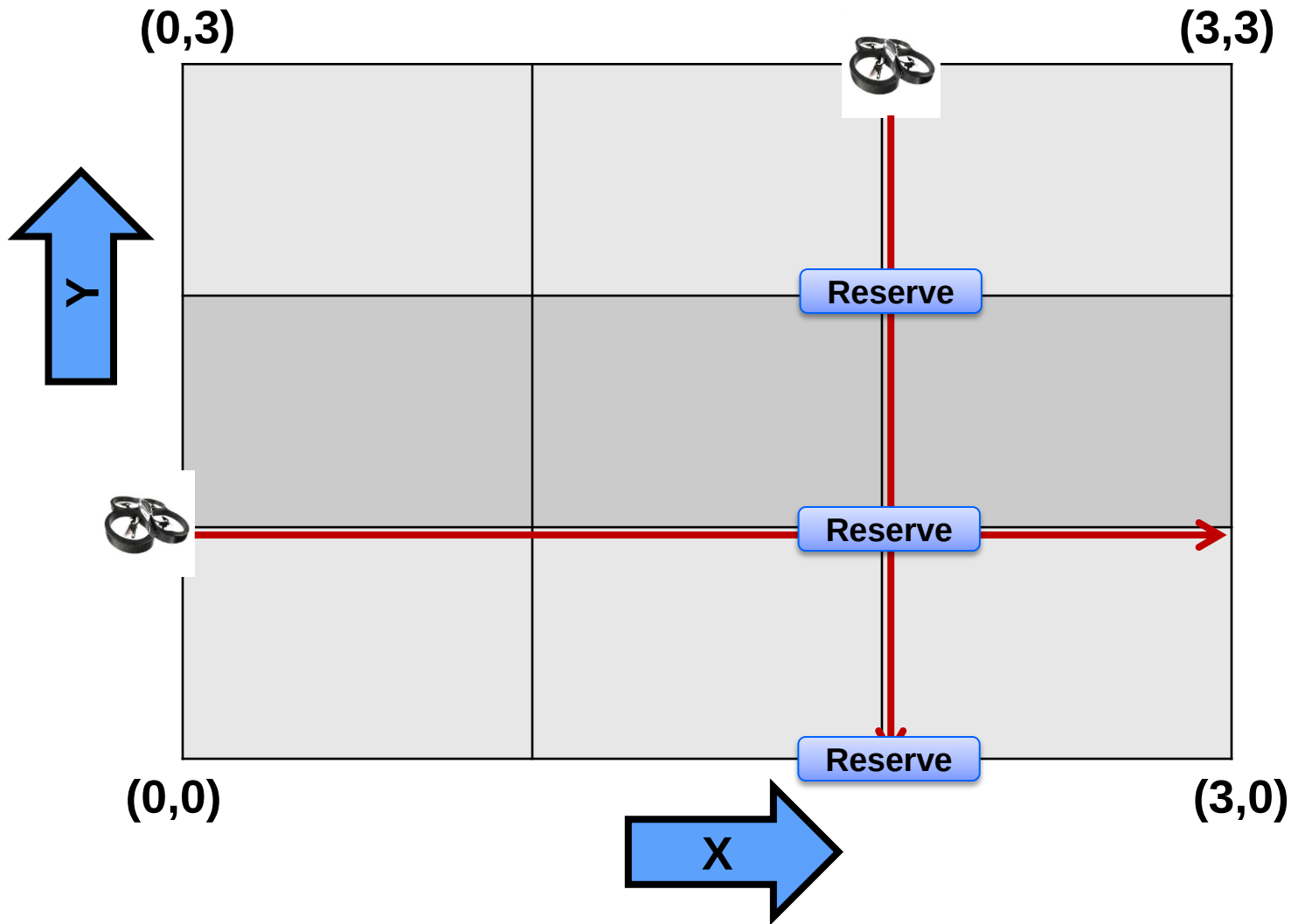
*Use 2 copies of $\overrightarrow{GV}$
Use each copy as input
in alternate rounds*



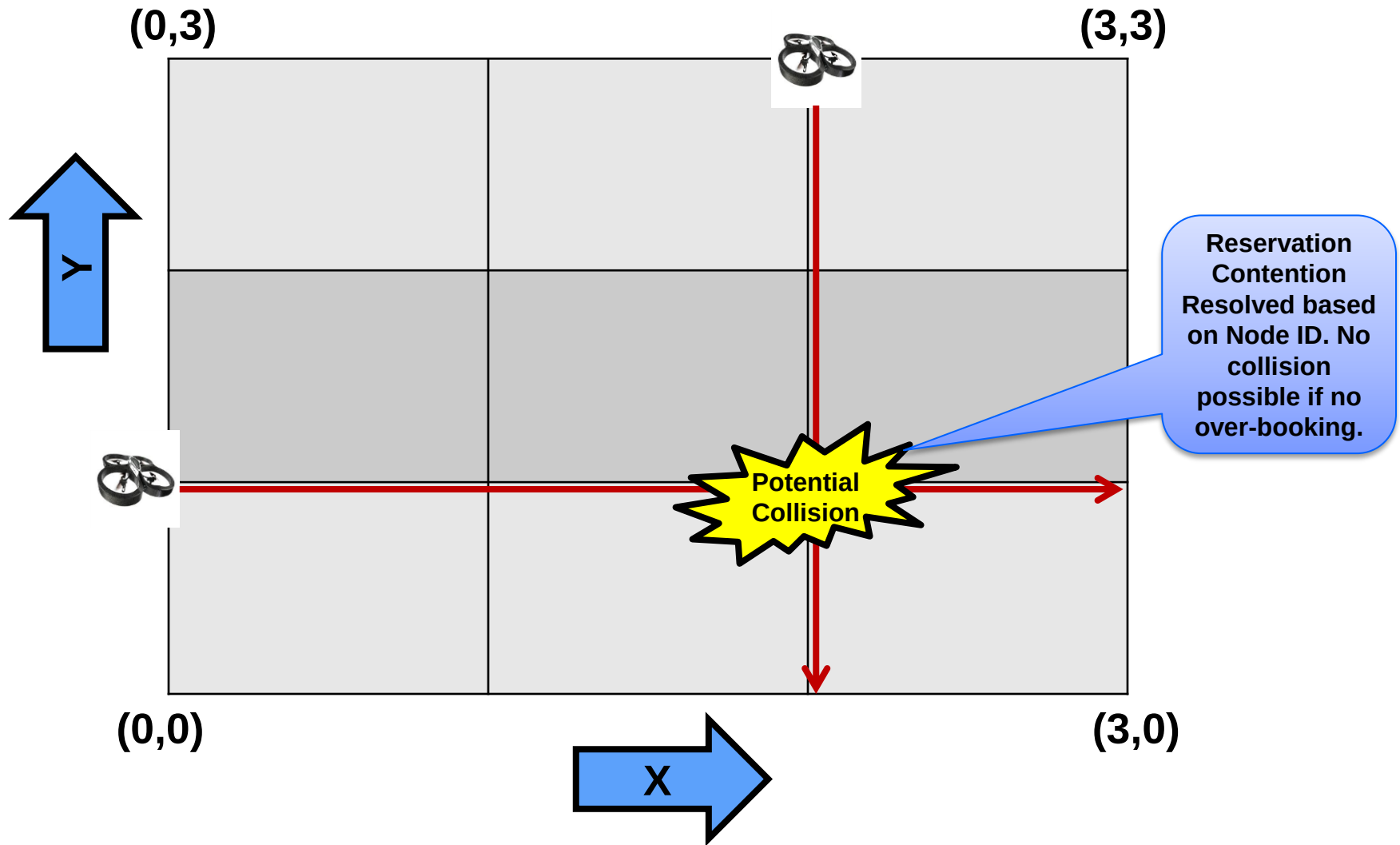
Example: 2D Synchronous Collision Avoidance



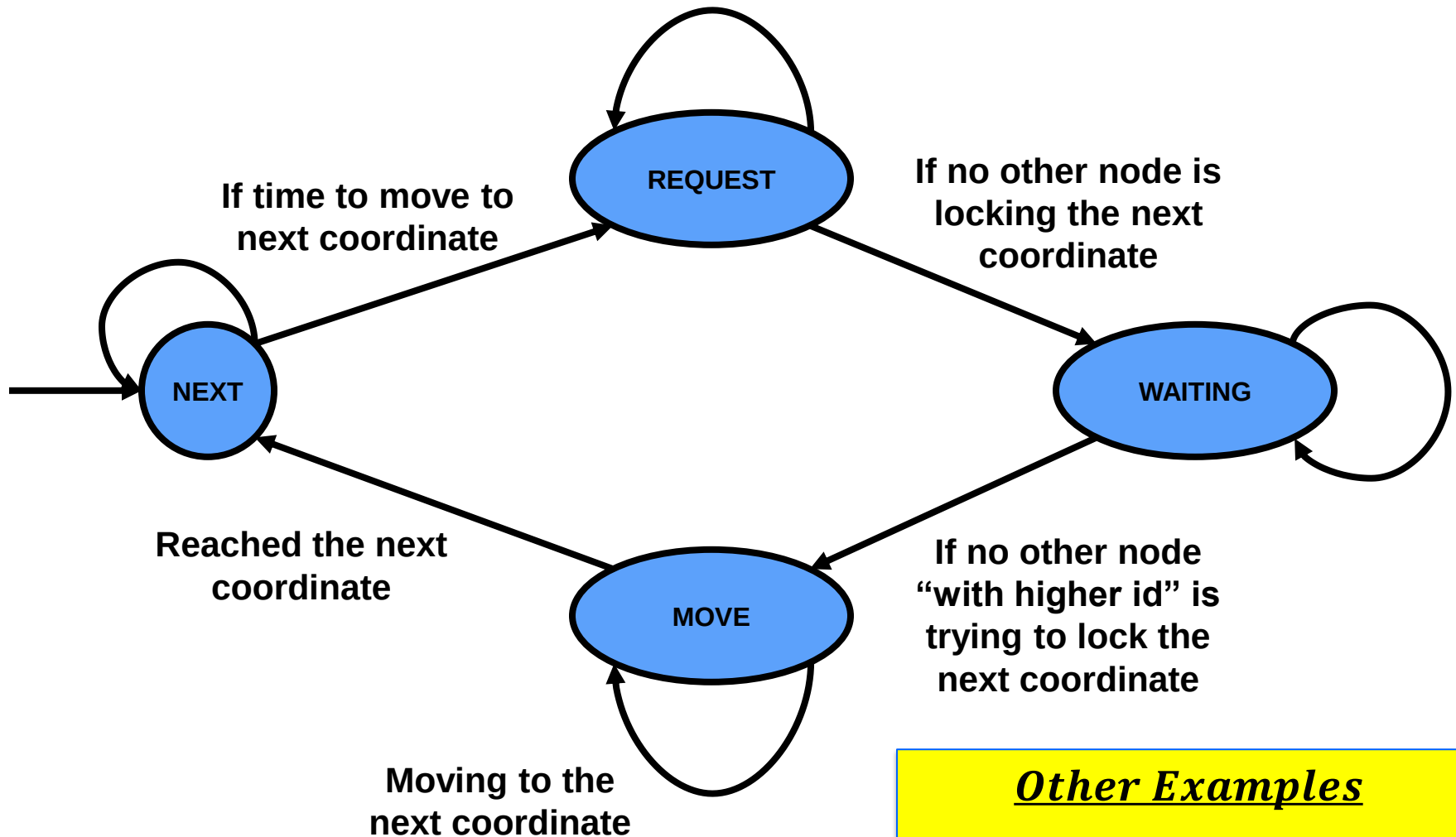
Example: 2D Synchronous Collision Avoidance



Example: 2D Synchronous Collision Avoidance



2D Collision Avoidance Protocol



Other Examples

3D Collision Avoidance
Mutual Exclusion



Results: 3D Collision Avoidance

3DCOLL-OK-4x4						
R	T_S	T_D	T_S	T_D	T_S	T_D
	$n = 2$		$n = 4$		$n = 6$	
10	13	10	59	40	219	96
20	37	31	351	123	1014	480
30	48	48	406	202	—	—
	$\mu=2.213 \ \sigma=0.715$					

3DCOLL-OK-7x7					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 2$		$n = 4$		$n = 6$	
31	35	323	148	1099	323
73	72	1262	401	—	—
142	113	—	—	—	—
$\mu=2.294 \ \sigma=0.763$					

3DCOLL-BUG-4x4					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 2$		$n = 4$		$n = 6$	
8	9	49	36	123	96
24	36	119	101	410	210
42	44	206	155	—	—
$\mu=1.615 \ \sigma=0.425$					

3DCOLL-BUG-7x7					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 2$		$n = 4$		$n = 6$	
22	23	194	114	—	—
57	76	—	—	—	—
117	134	—	—	—	—
$\mu=1.514 \ \sigma=0.344$					

SEQDBL
is better

T_S, T_D = model checking time with SEQSEM, SEQDBL

μ, σ = Avg, StDev of $\frac{T_S}{T_D}$

n = #of nodes R = #of rounds $G \times G$ = grid size



Software

Compilation
2014

© 2014 Carnegie Mellon University

Results: 2D Collision Avoidance

2DCOLL-OK-4x4						
R	T_S	T_D	T_S	T_D	T_S	T_D
	$n = 2$		$n = 4$		$n = 6$	
10	17	25	87	262	280	831
20	123	271	1474	2754	–	–
30	863	1301	–	–	–	–
	$\mu=0.446 \ \sigma=0.118$					

2DCOLL-BUG1-4x4					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 2$		$n = 4$		$n = 6$	
3	2	12	11	30	22
8	7	36	29	80	75
12	15	57	51	144	105
$\mu=1.282 \ \sigma=0.264$					

2DCOLL-BUG2-4x4					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 2$		$n = 4$		$n = 6$	
4	3	13	11	30	29
8	9	33	33	76	66
16	21	57	77	150	120
$\mu=1.056 \ \sigma=0.266$					

2DCOLL-OK-7x7						
R	T_S	T_D	T_S	T_D	T_S	T_D
	$n = 2$		$n = 4$		$n = 6$	
10	74	146	395	1016	1707	–
20	1726	3096	–	–	–	–
30	–	–	–	–	–	–
	$\mu=0.598 \ \sigma=0.202$					

2DCOLL-BUG1-7x7					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 2$		$n = 4$		$n = 6$	
7	7	32	24	101	70
15	22	94	55	345	150
40	35	180	91	–	223
$\mu=1.382 \ \sigma=0.517$					

2DCOLL-BUG2-7x7					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 2$		$n = 4$		$n = 6$	
5	10	26	36	188	113
19	22	71	113	207	166
46	68	124	295	416	235
$\mu=0.906 \ \sigma=0.393$					

Depends
on the
example

T_S, T_D = model checking time with SEQSEM, SEQDBL

μ, σ = Avg, StDev of $\frac{T_S}{T_D}$

n = #of nodes R = #of rounds $G \times G$ = grid size

Results: Mutual Exclusion

MUTEX-OK						
R	T_S	T_D	T_S	T_D	T_S	T_D
	$n = 6$		$n = 8$		$n = 10$	
60	406	396	1116	1051	2388	2268
80	850	806	2268	1967	4525	4249
100	1404	1381	3584	3452	7092	6764
	$\mu=1.040 \ \sigma=0.038$					

MUTEX-BUG1					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 6$		$n = 8$		$n = 10$	
184	175	517	439	1068	959
402	372	1013	925	2203	1812
734	686	1726	1566	3513	3287
$\mu=1.056 \ \sigma=0.060$					

MUTEX-BUG2					
T_S	T_D	T_S	T_D	T_S	T_D
$n = 6$		$n = 8$		$n = 10$	
233	216	637	553	1292	1167
500	462	1218	1112	2602	2139
890	838	2056	1860	4216	3742
$\mu=1.065 \ \sigma=0.056$					

SEQSEM
and
SEQDBL
similar

$T_S, T_D = \text{model checking time with SEQSEM, SEQDBL}$

$\mu, \sigma = \text{Avg, StDev of } \frac{T_S}{T_D}$

$n = \text{\#of nodes} \quad R = \text{\#of rounds} \quad G \times G = \text{grid size}$



Synchronizer Protocol: 2BSYNC

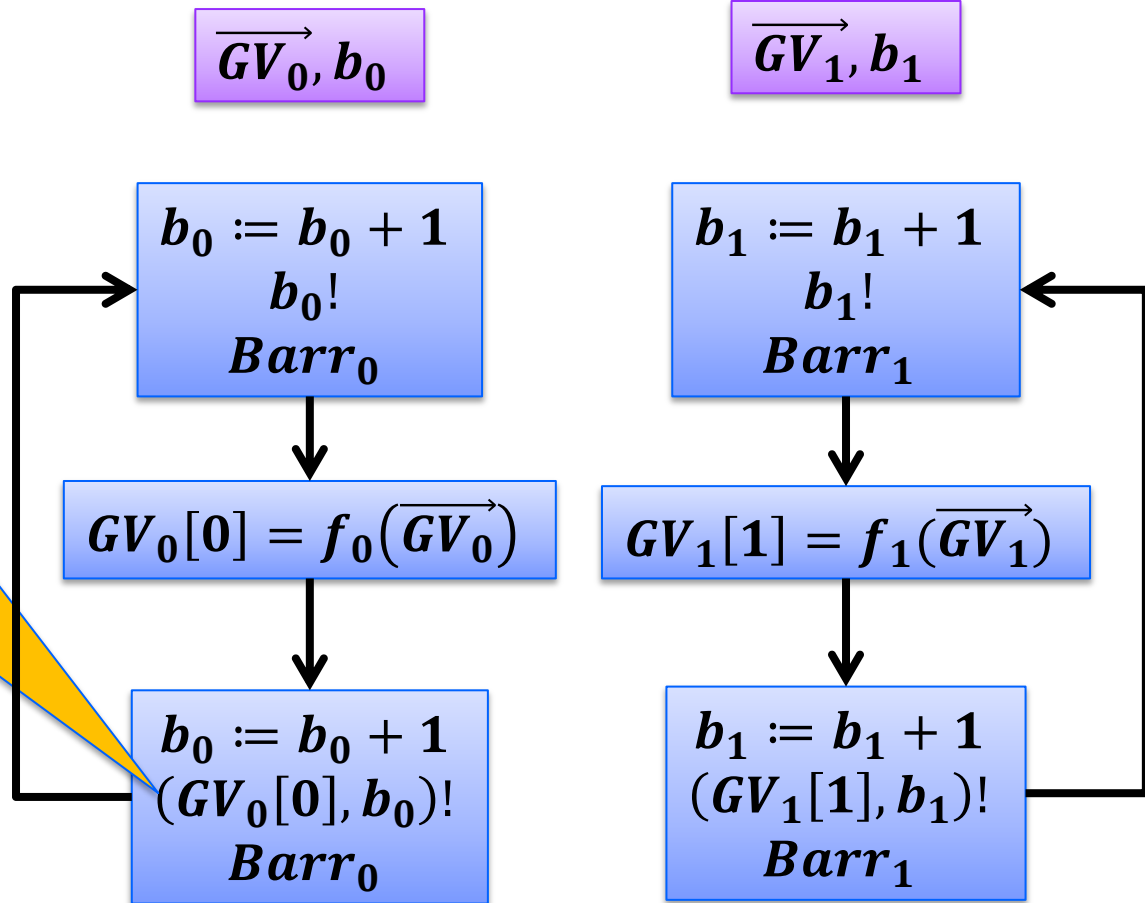
Node 0 = $f_0()$

Shared Variables: $\overrightarrow{GV} = GV[0], GV[1]$

Node 1 = $f_1()$

*Use barrier
variables: b_0, b_1
Initialized to 0*

Atomic Send. Either both $GV_0[0]$ and b_0 are received, or none is received. Can be implemented on existing network stack, e.g, TCP/IP



$Barr_0 \equiv \text{while}(b_1 < b_0) \text{ skip};$

*Proof of correctness
in paper*



Tool Overview

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 ...`



Future Work



Improving scalability and verifying with unbounded number of rounds

Verifying for unbounded number of nodes (parameterized verification)

- Paper at SPIN'2014 Symposium

Asynchronous and partially synchronous network semantics

Scalable model checking

- Abstraction, compositionality, symmetry reduction, partial order reduction

Fault-tolerance, uncertainty, ...

- Combine V&V of safety-critical and mission-critical properties



Contact Information Slide Format

Sagar Chaki

Principal Researcher

SSD/CSC

Telephone: +1 412-268-1436

Email: chaki@sei.cmu.edu

U.S. Mail

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

Web

www.sei.cmu.edu

www.sei.cmu.edu/contact.cfm

Customer Relations

Email: info@sei.cmu.edu

Telephone: +1 412-268-5800

SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257



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;  
}
```

