

# SECURITY-POLICY MONITORING AND ENFORCEMENT WITH JAVAMOP

**SOHA HUSSEIN<sup>1</sup>**

**PATRICK MEREDITH<sup>2</sup>**

**GRIGORE ROSU<sup>3</sup>**

<sup>1</sup>UNIV. OF AIN SHAMS,  
EGYPT.

<sup>2</sup>UNIV. OF ILLINOIS AT URBANA-  
CHAMPAIGN, USA.

**PLAS'12**



# INTRODUCTION & MOTIVATION

## ■ INLINED SECURITY ENFORCEMENT MECHANISMS:

- DESIGNED FOR ENFORCING SECURITY PROPERTIES.
- EXECUTION MONITORS, REWRITING MODEL, EDIT AUTOMATA
- SASI, NACCIO, PoET/PSLANG, POLYMER AND SPoX.

VS.

## ■ RUNTIME MONITORING AND VERIFICATION:

- DESIGNED TO BE GENERIC.
- USED TO ENFORCE FUNCTIONAL CORRECTNESS OF POST-PRODUCTION PROGRAMS OR DEBUGGING AND TESTING PROGRAM DURING THE PRODUCTION PHASE.
- MOP, JPAX, MAC, J-LO, TRACEMATCHES, ETC.



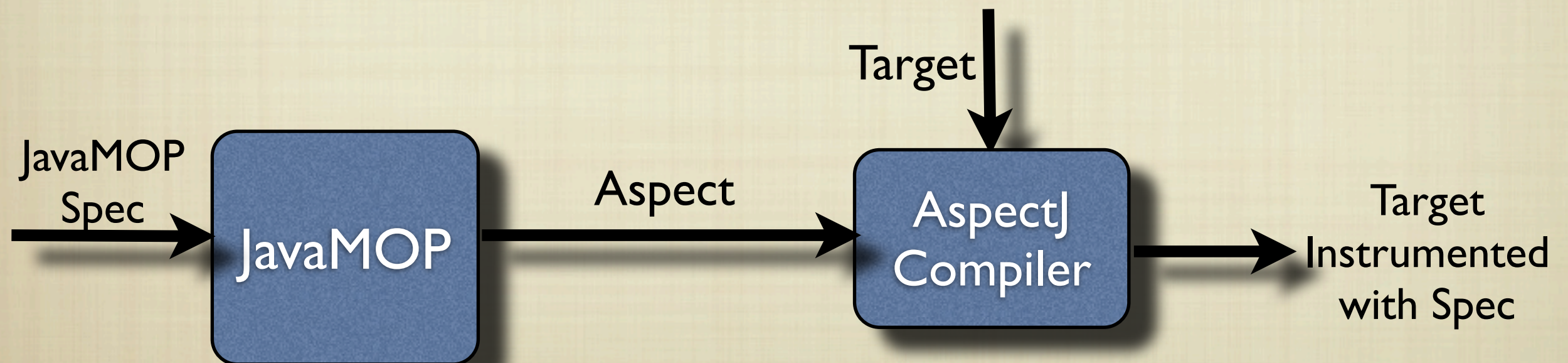
# MOTIVATION

- **USE RV-BASED SYSTEM (JAVAMOP) TO SPECIFY AND ENFORCE SECURITY POLICIES.**
- **PROVIDE A MEANS TO SUPPORT COMPOSITION AND CONFLICTS AMONG POLICIES.**
- **MEASURE PERFORMANCE BETWEEN JAVAMOP AND OTHER SECURITY-ENFORCEMENT SYSTEMS.**



# RV SYSTEM: JAVAMOP

- MONITORING ORIENTED PROGRAMMING (MOP) IS A **FORMALISM-GENERIC** RUNTIME VERIFICATION AND MONITORING FRAMEWORK.
- JAVAMOP IS THE **JAVA INSTANCE** FOR MOP.
- OUTPUT OF JAVAMOP IS AN **ASPECTJ FILE**.





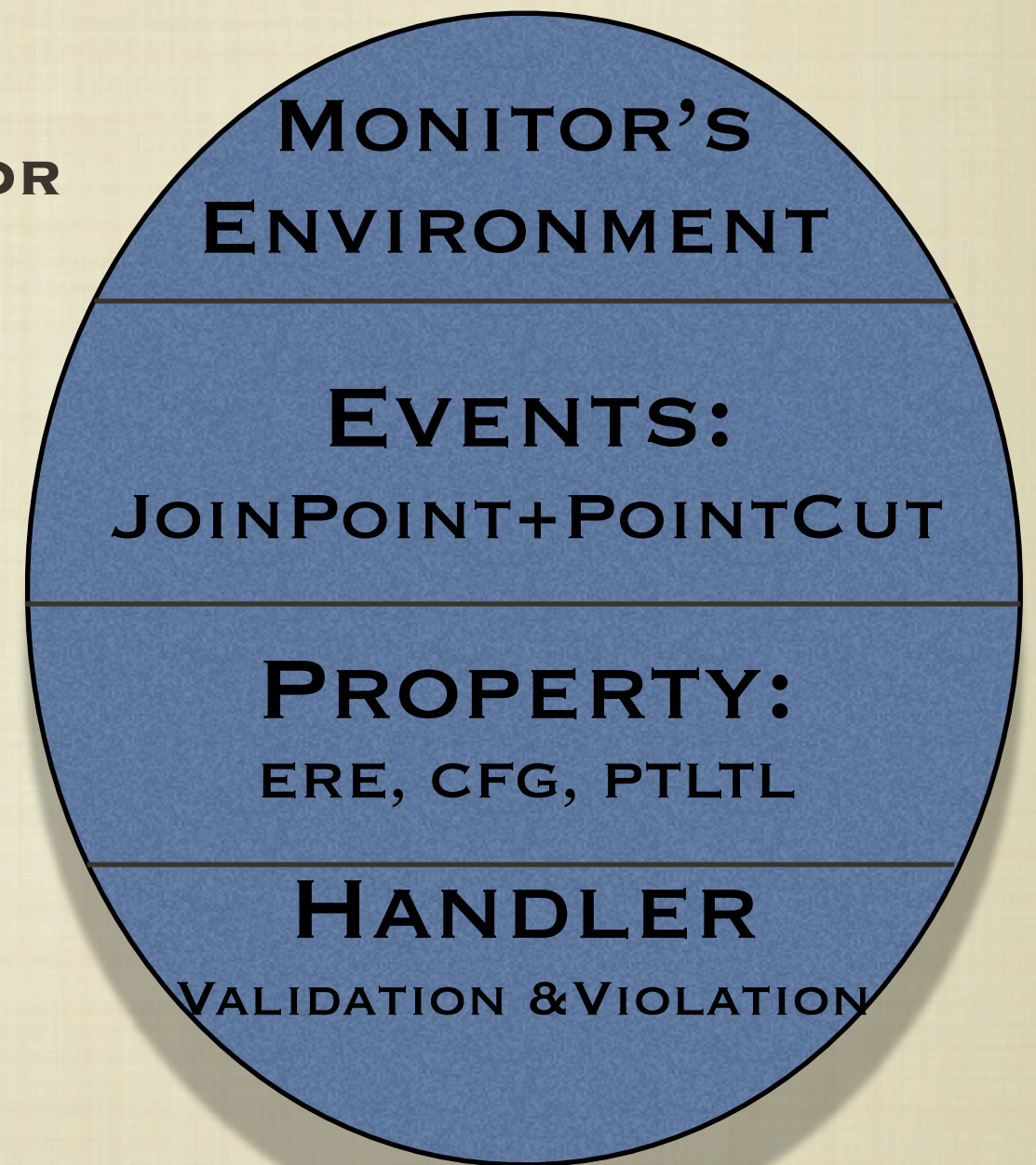
# MOP GENERAL TERMS & FEATURES

■ **GENERIC  
FORMALISM**

**MOP MONITOR**

■ **CATCHING  
VALIDATION OR  
VIOLATIONS**

■ **PARAMETRIC  
PROPERTIES.**





# ENFORCING SECURITY POLICIES

- ACCESS CONTROL POLICIES.
  - SAFE LOCK. §2.
  - DISABLE SYSTEM CALLS. §3.1.
- SQL INJECTION. §3.3.
- SEPARATION OF DUTIES. §3.2.
- VARIATIONS OF THE CHINESE WALL. §3.4.
  - FILE NETWORK WALL
  - ORIGINAL CHINESE WALL



# CHINESE WALL

- THE POLICY ATTEMPTS TO KEEP USERS IN THE SYSTEM FROM ACCESSING OBJECTS OF DIFFERENT DATASETS THAT ARE IN THE SAME CONFLICT CLASS.

IN THE MONITORING WORLD THIS IS TRANSLATED  
To:

- WE WANT TO CHECK EACH AND EVERY ACCESS OF OBJECTS IN THE SUBJECT AND MAKE SURE THAT THE OBJECT BEING ACCESSED DOES NOT LIE IN CONFLICT WITH PREVIOUSLY ACCESSED OBJECTS, FOR EACH SUBJECT INSTANCE.



# CHINESE WALL IN JAVAMOP

EVENT ACCESS BEFORE  
(OBJ O): CALL(\* OBJ.READ  
( )) && TARGET(O)

WHAT WE WANT IS:

1. A PARAMETRIC SPECIFICATION.
2. MEANS TO TRACK THE CALL STACK OF SUBJECTS.
3. CARRY THE NECESSARY CHECKS WHEN MATCHED.

SUBJECT S  
CALL STACK

EVENT METHODCALL BEFORE  
(SUBJECT S):  
CALL(\* SUBJECT.\*(..)) && TARGET(S)

EVENT METHODRETURN AFTER  
(SUBJECT S) :  
CALL(\* SUBJECT.\*(..))&&TARGET(S)



# CHINESE WALL IN JAVAMOP

```
ChineseWall(Subject S) {  
    SubjectWall monitoredSubjectWall;  
    Obj readObject;  
    event methodCall before(Subject S): call(* Subject.*(..)) && target(S) {  
        if (monitoredSubjectWall == null)  
            monitoredSubjectWall = new SubjectWall(S); }  
  
    event methodReturn after(Subject S) : call(* Subject.*(..))&&target(S) {}  
  
    event access before(Obj O): call(* Obj.Read()) && target(O) {  
        readObject = O; }  
  
    cfg: S -> S access | S M | epsilon,  
        M -> M methodCall M methodReturn  
        | epsilon  
    @fail{  
        SubjectWall sw = __MONITOR.monitoredSubjectWall;  
        Obj o = __MONITOR.readObject;  
        if(sw.conflictClassContains(o) && !sw.dataSetContains(o)){  
            System.out.println("Chinese Wall is violated. Halting..");  
            Runtime.getRuntime().halt(1); }  
        sw.addToConflictClass(o);  
        sw.addToDataSet(o); }  
}
```



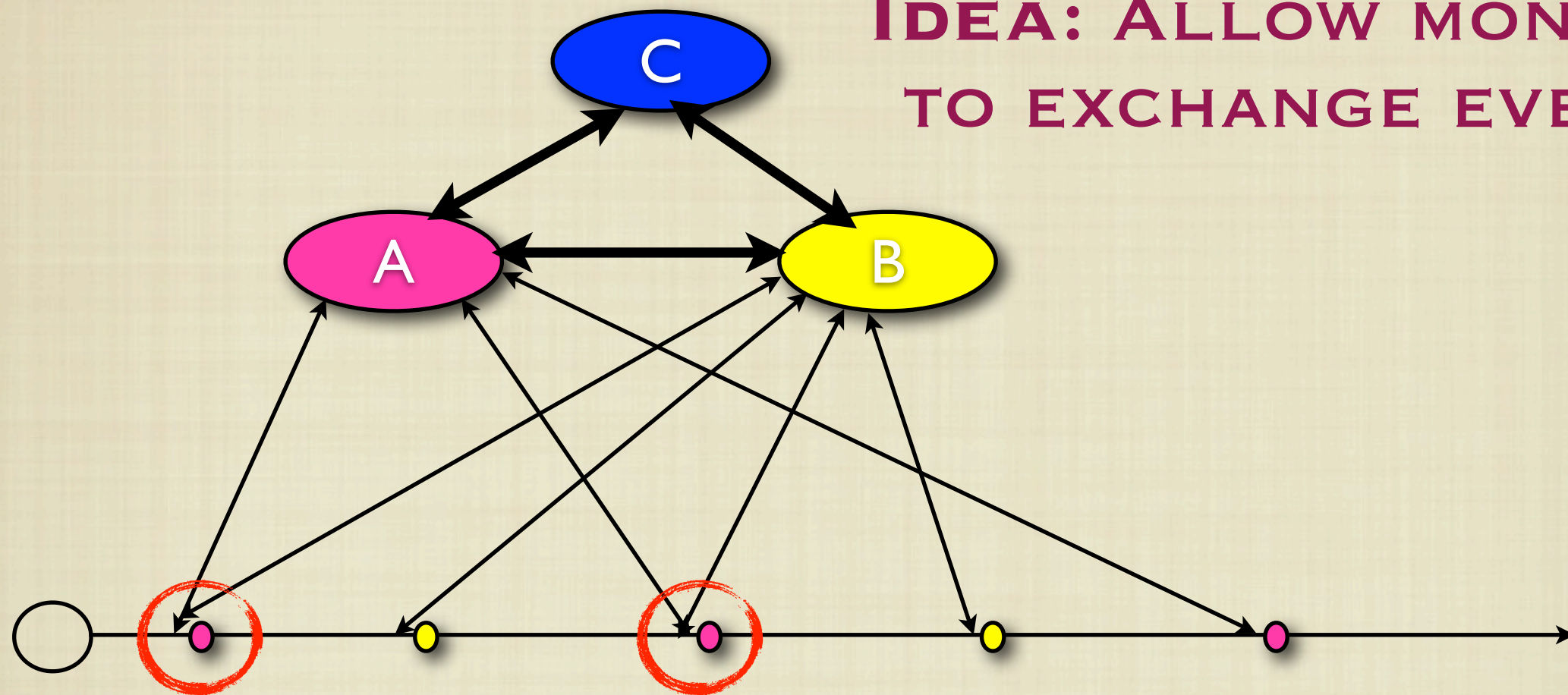
# POLICY COMPOSITION AND CONFLICT RESOLUTION IN JAVAMOP

- JAVAMOP, BY DEFAULT, ALLOWS MULTIPLE POLICIES TO COEXIST WITHIN A GIVEN TARGET PROGRAM.
- HOWEVER IT MAKES NO GUARANTEES ON HOW THEY WILL OPERATE TOGETHER IF THEIR EVENTS INTERFERE WITH EACH OTHER, THAT IS, IF THEY HAPPEN TO SELECT SOME OF THE SAME PROGRAM POINTS.



# COMPOSITION AND CONFLICT PROBLEM

**IDEA: ALLOW MONITORS TO EXCHANGE EVENTS.**

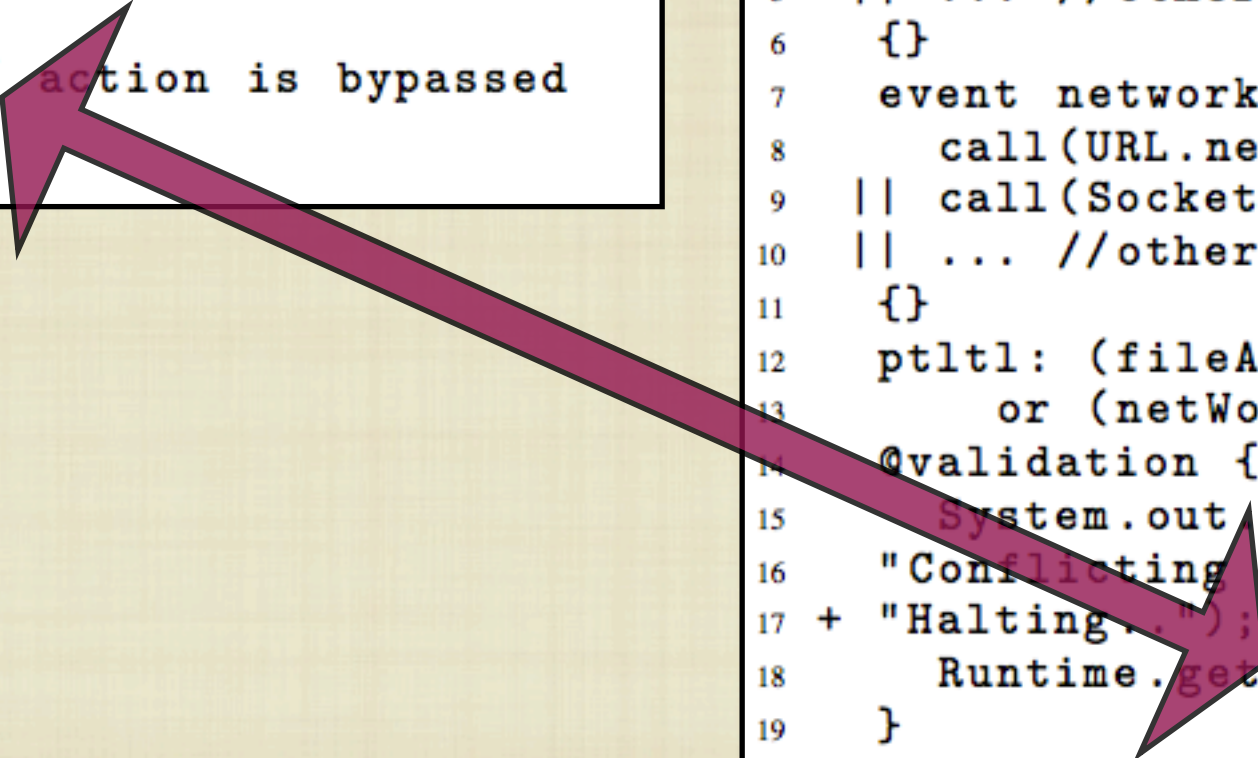


		A			
		Halt	Skip	Proceed	Exec
B	Halt	Halt	?	?	?
	Skip	?	Skip	?	?
	Proceed	?	?	Proceed	?
	Exec	?	?	?	?



# POLICY COMPOSITION AND CONFLICT PROBLEM IN JAVAMOP

```
1 RestrictSystemCalls() {  
2   event systemCalls Object around():  
3     call(* Runtime.exec(...)){}  
4   ere : systemCalls+  
5   @match {  
6     __SKIP; // action is bypassed  
7   }  
8 }
```



```
1 FileNetworkWall() {  
2   event fileAccess before():  
3     call(* Runtime.exec(...))  
4   || call(* File.createNewFile())  
5   || ... //other file access methods  
6   {}  
7   event networkAccess before():  
8     call(URL.new(...))  
9   || call(Socket.new(...))  
10  || ... //other network access methods  
11  {}  
12  ptl1: (fileAccess => <*> netWorkAccess)  
13        or (netWorkAccess => <*> fileAccess)  
14  @validation {  
15    System.out.println(  
16      "Conflicting resource access. "  
17    + "Halting.");  
18    Runtime.getRuntime().halt(1)  
19  }  
20 }
```

■ THE PROBLEM IS THAT EVERY MONITOR HERE IS ACTUALLY FIRING ACTION WITHOUT COORDINATION. WHAT WE NEED IS **COORDINATION**.



# MONITORING OF MONITORS

```
1 ConflictManager() {
2   event haltFN
3     after():call(void Manager haltFN()) {}
4   event haltNF
5     after():call(void Manager haltNF()) {}
6   event proceed
7     after():call(void Manager proceed()) {}
8   event skip
9     after():call(void Manager skip()) {}
10  ptl1: haltFN => <*>proceed
11  @validation{
12    System.out.println(
13      "FileNetworkWall violated..Halting.");
14    Runtime.getRuntime().halt(1);
15  }
16  ptl1: haltNF => (*)proceed
17  @validation{
18    System.out.println(
19      "FileNetworkWall violated..Halting.");
20    Runtime.getRuntime().halt(1);
21  }
22 }
```

```
1 FileNetworkWall() {
2   Manager M1 = new Manager();
3
4   event fileAccess before():
5     call(* Runtime.exec(..)) ||
6     call(* File.createNewFile())
7     || ... //other file access methods
8   {}
9   event networkAccess before():
10    call(URL.new(..)) || call(Socket.new(..))
11    || ... //other network access methods
12  {}
13  ptl1 : networkAccess => <*>fileAccess
14  @match{ __MONITOR.M1 haltFN(); }
15
16  ptl1: fileAccess => <*>networkAccess
17  @match{ __MONITOR.M1 haltNF(); }
18
19
20 }
```



# OTHER SECURITY CONCERNS §4

- **MONITOR'S INTEGRITY.**
- **RESTRICTING JAVA REFLECTION.**
- **ASPECTJ CORRECTNESS.**



# EXPERIMENTS

- THREE EXPERIMENTS WERE CARRIED OUT.
- THE FIRST IS SPECIALIZED TO TEST THE CHINESEWALL POLICY.
- THE SECOND AND THE THIRD EXPERIMENTS USE THE DAcAPO BENCHMARK SUITE (VERSION 9.12-BACH) AND SEVERAL JAVA API SECURITY POLICIES SPECIFIED USING JAVAMOP AND LATER USING OTHER IRM SYSTEMS.



# CHINESE WALL IN JAVAMOP

#Subjects	#Datasets	#Conflict	%Over-head	#Method call/return	#Access	Total #events	#Trigger
100	1000	10	6	2000	1000	3000	51500
200	4000	20	9	8000	4000	12000	406000
300	9000	30	5	18000	9000	27000	1363500
400	16000	40	3	32000	16000	48000	3224000
500	25000	50	6	50000	250000	75000	6287500



# JAVAMOP PERFORMANCE ON DACAPO BENCHMARK

	HiddenFileAccess			DisableNetwork			FileCreation			FileNetworkWall			DisSysCalls		AllPolicies		
avro	3	64	64	1	0	0	2	14	14	0	1388	0	1	0	2	1416	56
batik	-1	122	122	1	685	685	-1	0	0	0	1692	685	-1	0	1	2071	1542
eclipse	-1	642	642	1	438	438	2	28	28	1	1958	439	1	0	1	3047	1542
fop	1	121	121	0	0	0	0	0	0	1	667	0	1	0	1	548	49
h2	2	15	15	0	0	0	-1	0	0	1	73	0	-1	0	-1	70	9
ivy	0	2726	2726	2	0	0	-1	2	2	1	7347	0	-2	0	-1	10049	2720
lucene	1	25	25	0	0	0	1	256	256	0	24534	0	-1	0	-2	16475	176
lucene	-2	1549	1549	0	0	0	0	0	0	0	3807	0	0	0	0	3670	1033
pmd	-1	3138	3138	-1	0	0	-1	0	0	-3	6288	0	0	0	-1	7182	2242
sunflow	0	13	13	1	0	0	1	0	0	2	72	0	1	0	0	67	9
tomcat	1	37	37	2	3	3	1	0	0	2	20044	3	1	0	1	14680	39
tradebeans	0	13	13	0	0	0	1	0	0	0	3430	0	0	0	1	3427	9
tradesoap	0	15	15	0	0	0	1	0	0	0	2788	0	0	0	-1	2787	11
xalan	-1	23826	23826	0	1	1	-1	0	0	1	47741	1	-2	0	0	51144	17022



# JAVAMOP VS. SPOX & POLYMER ON DACAPO

	DisableSystemCalls			FileNetworkWall			LimitOpenedFiles			NoWriteAfterClose	
avroora	1	3	1	3	2	2	2	2	2	2	1
batik	-1	0	-1	1	-1	-1	-1	1	0	-1	-1
eclipse	-1	1	-8	1	1	-3	0	-5	-1	1	-2
fop	0	16	19	5	17	15	0	16	13	1	16
h2	1	1	-	2	0	-	-1	-1	-	-1	-1
jython	1	-2	-2	-1	-2	2	0	-2	1	-1	-4
luindex	0	3	-1	0	3	14	0	0	13	1	5
lusearch	0	0	0	0	1	2	-1	1	0	1	1
pmd	-2	-2	10	-2	-2	128	-2	-2	39	-1	-2
sunflow	1	0	0	0	0	1	1	0	-1	1	0
tomcat	2	0	7	1	0	126	1	-1	44	2	-1
tradebeans	1	1	2	1	0	1	1	0	1	1	1
tradesoap	0	-1	1	0	1	9	0	1	3	-1	0
xalan	-1	-1	4	-1	-3	63	1	-3	22	-1	-2



# WRAP-UP

- **IRM CAN BE CONSIDERED AS A SPECIFIC INSTANCE OF RV. SPECIFICALLY, WE DEMONSTRATED HOW JAVAMOP, AN RV SYSTEM, IS ABLE TO EFFECTIVELY AND EFFICIENTLY SPECIFY AND MONITOR SECURITY POLICIES.**
- **WE SHOWED HOW JAVAMOP CAN BE USED TO RESOLVE POTENTIAL CONFLICTS OR COMPOSITION AMONG MONITORS.**
- **OUR EXPERIMENTS WHICH SHOWED THAT JAVAMOP YIELDS A BETTER PERFORMANCE RESULTS WHEN COMPARED TO SPoX AND POLYMER.**
- **A FORMAL FRAMEWORK FOR THE COMPOSITION OF JAVAMOP SPECIFICATIONS IS A DIRECTION FOR FUTURE RESEARCH.**



# SECURITY-POLICY MONITORING AND ENFORCEMENT WITH JAVAMOP

**SOHA HUSSEIN<sup>1</sup>**

**PATRICK MEREDITH<sup>2</sup>**

**GRIGORE ROSU<sup>3</sup>**

<sup>1</sup>UNIV. OF AIN SHAMS,  
EGYPT.

<sup>2</sup>UNIV. OF ILLINOIS AT URBANA-  
CHAMPAIGN, USA.

**PLAS'12**