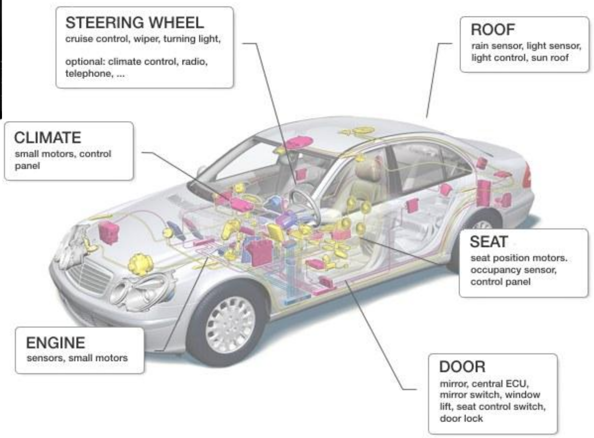
Automated Analysis of Industrial Embedded Software

Moonzoo Kim
Provable Software Lab
KAIST, South Korea

Thanks to Hotae Kim and Yoonkyu Jang
Samsung Electronics, South Korea

Strong IT Industry in South Korea



Embedded Software in Two Different Classes

	Consumer Electronics	Safety Critical Systems
Examples	Smartphones, flash memory platforms	Nuclear reactors, avionics, cars
Market competition	High	Low
Life cycle	Short	Long
Development time	Short	Long
Model-based development	None	Yes
Important value	Time-to-market	Safety

Personal Research Roadmap

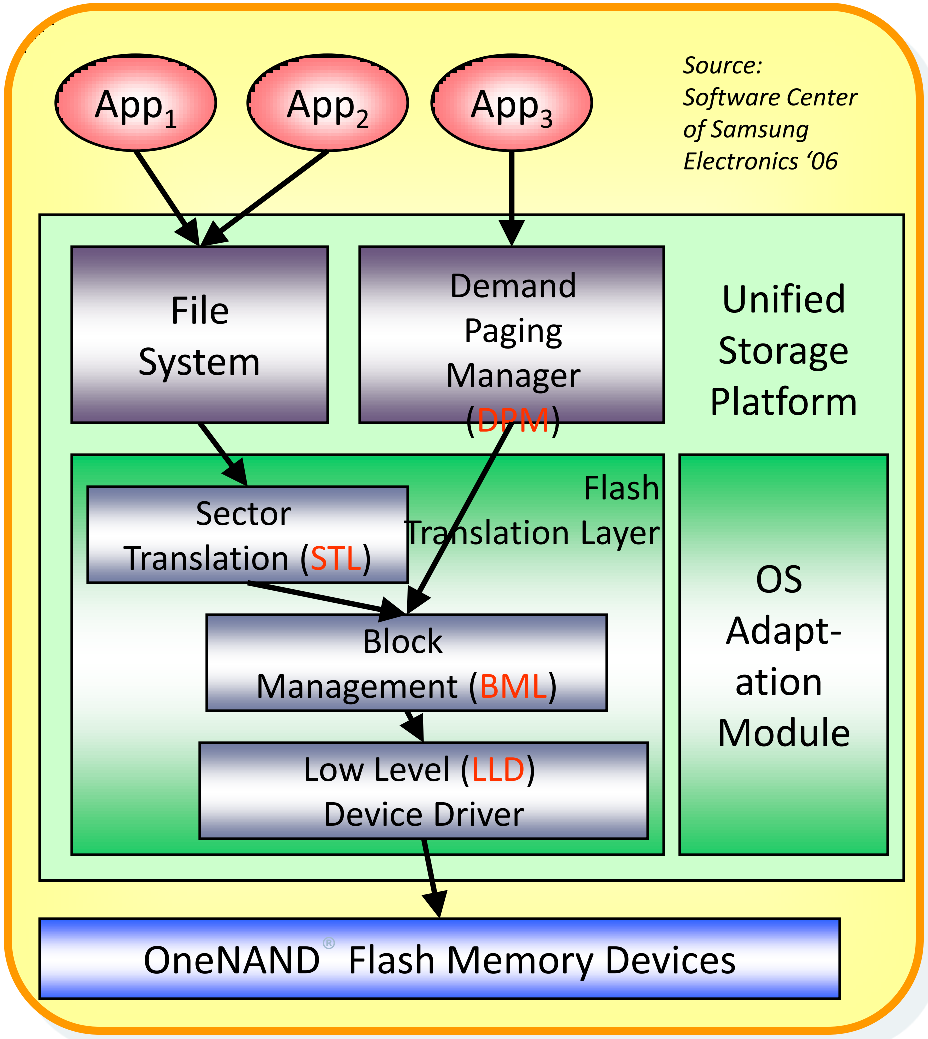
- Past: RV (dynamic) && MC (static)
  - Runtime Verification — FMSD'04
  - Model Checking — Spin'08
  - Software Model Checking — ASE '08, TSE'11
- Current: Extended Concolic Testing
  - Concolic Testing — ICST12a, FACJ'12, FSE'11a
  - Distributed Concolic Testing — ICST'12b, FSE'11b, ICTAC'10
  - Hybrid Algorithm (i.e., w/ Genetic Alg) — ISSRE '11, FSE'10
- Future: Concolic Testing with Intelligence
  - Statistic Inference + Machine Learning + User Assistance

Better Industrial Application

Part I: Experience from SW Model Checking

Target system: Samsung Unified Storage Platform (USP) for OneNAND® flash memory (around 30K lines of C code)

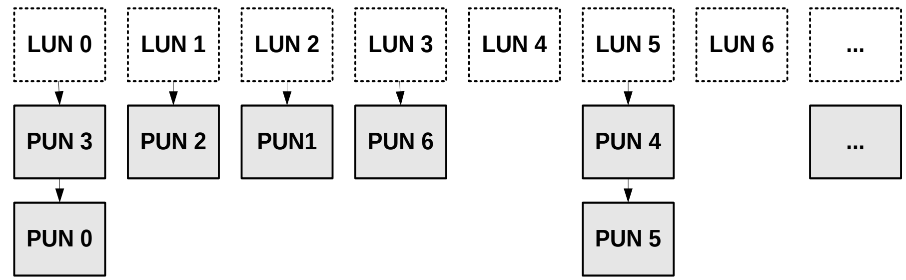
- Characteristics of OneNAND® flash mem
  - Each memory cell can be written limited number of times only
    - Logical-to-physical sector mapping
    - Bad block management, wear-leveling, etc
  - Concurrent I/O operations
    - Synchronization among processes is crucial
  - XIP by emulating NOR interface through demand-paging scheme
    - binary execution has a highest priority
  - Performance enhancement
    - Multi-sector read/write
    - Asynchronous operations
    - Deferred operation result check



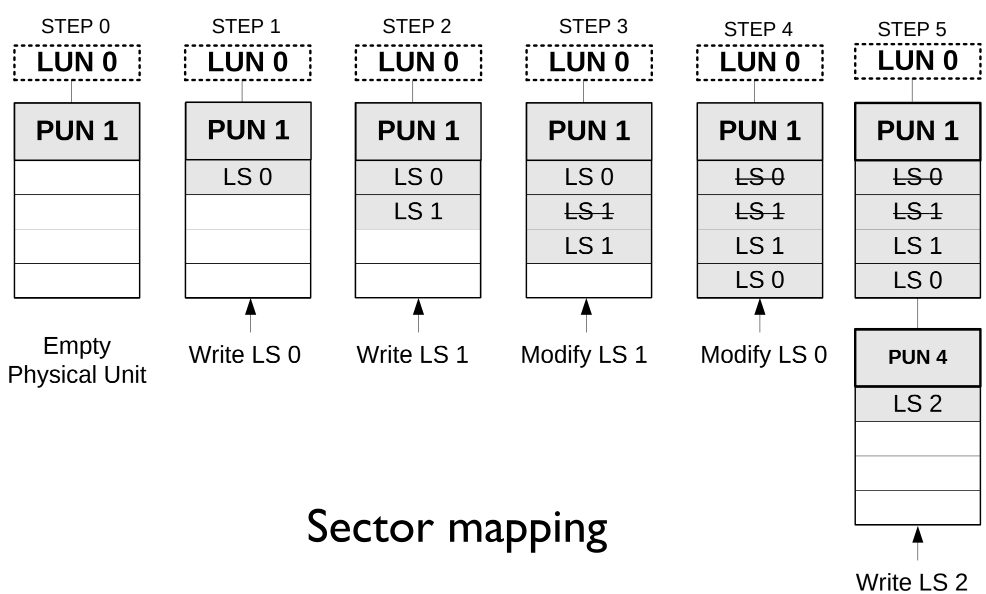
Results of Unit Analysis through CBMC and BLAST [TSE'11]

- Demand paging manager (234 LOC)
  - Detected a bug of not saving the status of suspended erase operation
- Concurrency handling
  - Confirmed that the BML semaphore was used correctly in all 14 BML functions (150 LOC on average)
  - Detected a bug of ignoring BML semaphore exceptions in a call sequence from STL (2500 LOC on average)
- Multi-sector read operation (MSR) (157 LOC)
  - Provided high assurance on the correctness of MSR
    - no violation was detected even after exhaustive analysis (at least with a small number of physical units(~10))
- In addition, we evaluated and compared pros and cons of CBMC and BLAST empirically

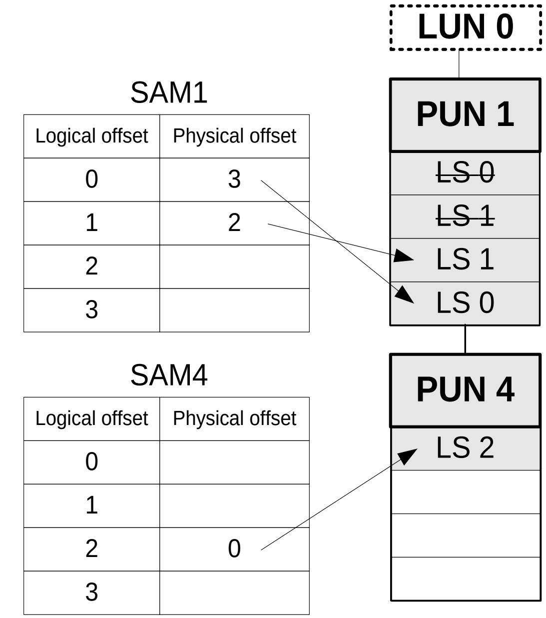
Logical to Physical Sector Mapping



1:N mapping from a LUN to PUNs



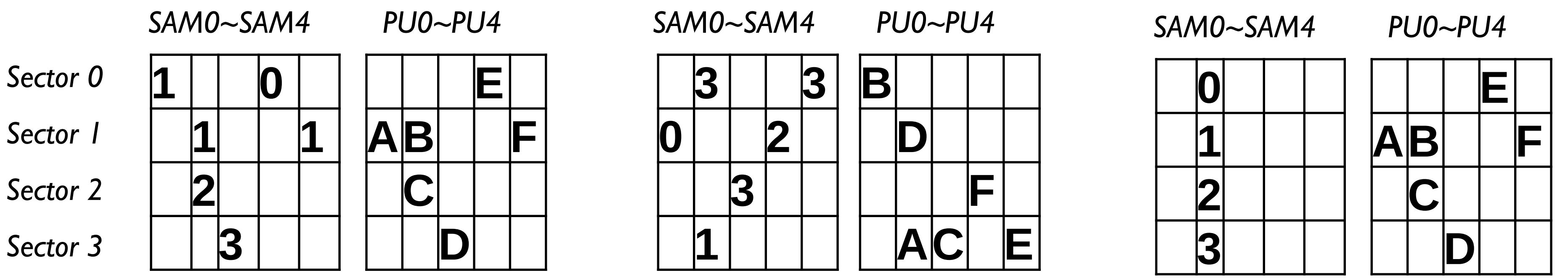
Sector mapping



Sector Allocation Map (SAM)

- In flash memory, logical data are distributed over physical sectors.

Multi-sector Read Operation (MSR)

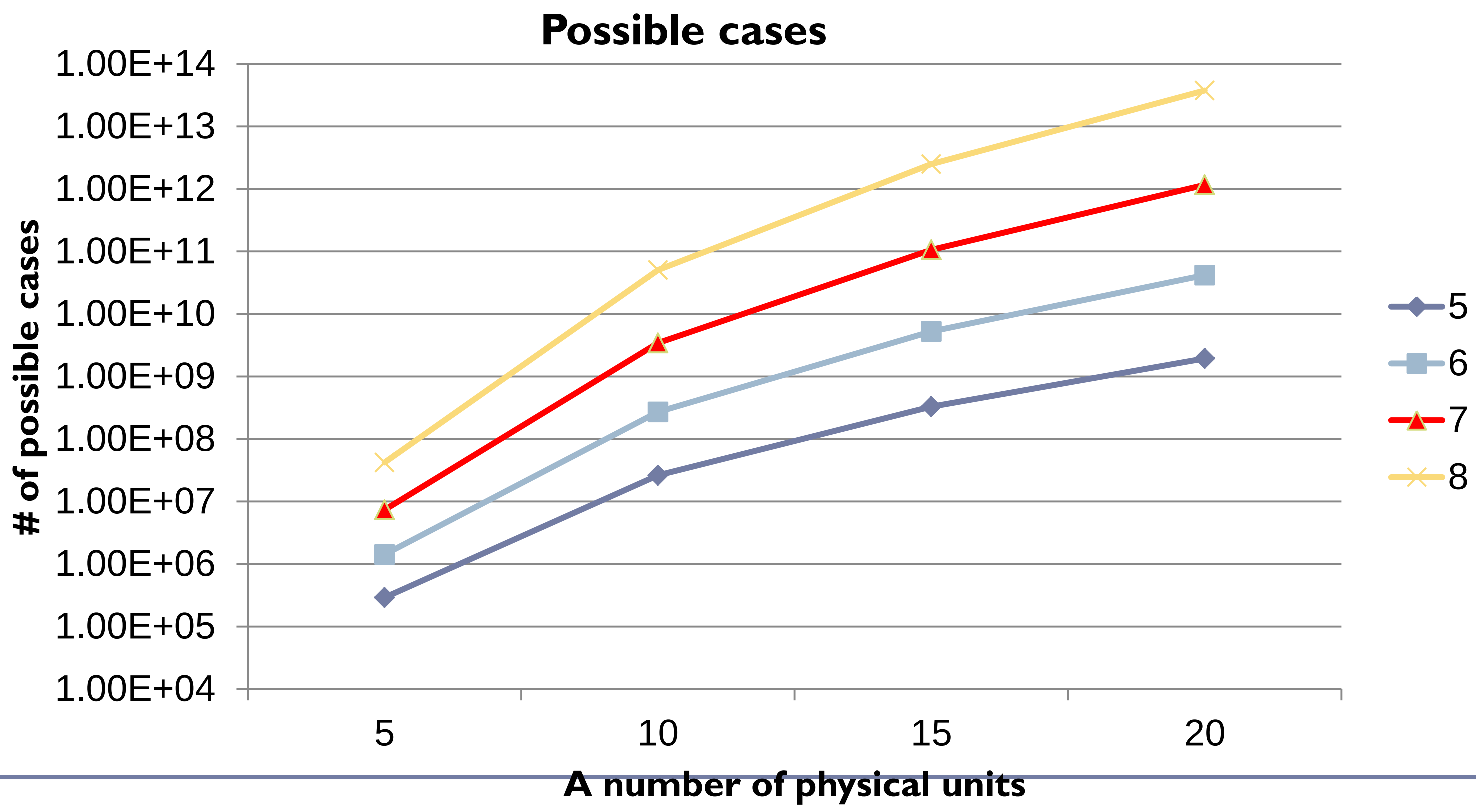


a) A distribution of "ABCDEF" b) Another distribution of "ABCDEF" c) Invalid distribution of "ABCDEF"

- MSR reads adjacent multiple physical sectors once in order to improve read speed
  - MSR is 157 lines long, but highly complex due to its 4 level loops
  - 4 parameters to specify logical data to read (from, to, how long, read flag)
- The requirement property is to check
  - after_MSR -> (∀ i. logical_sectors[i] == buf[i])
- We built a **verification environment model** for MSR

Exponential Increase of Distribution Cases

$$\sum_{i=1}^{n-1}\left(_{(4\times i)}C_4 \times 4!\right) \times \left(_{(4\times(n-i))}C_{(l-4)} \times (l-4)!\right)$$



Environment Modeling

1. **One PU is mapped to at most one LU**
2. **Valid correspondence between SAMs and PUs:**

   If the *i* th LS is written in the *k* th sector of the *j* th PU, then the *i* th offset of the *j* th SAM is valid and indicates the k'th PS ,

   Ex> 3rd LS ('C') is in the 3rd sector of the 2nd PU, then SAM1[2] ==2

   i=2 k=2 j=1

3. **For one LS, there exists only one PS that contains the value of the LS:**

   The PS number of the *i* th LS must be written in only one of the (*i* mod *4*) th offsets of the SAM tables for the PUs mapped to the corresponding LU.

$$
\begin{aligned}
\forall i,j,k\ (LS[i] = PU[j].sect[k] \rightarrow (&SAM[j].valid[i\ mod\ m] = true \\
&\&\ SAM[j].offset[i\ mod\ m] = k \\
&\&\ \forall p.(SAM[p].valid[i\ mod\ m] = false) \\
&\text{where } p \neq j \text{ and } PU[p] \text{ is mapped to} \lfloor \frac{i}{m} \rfloor_{th}\ LU))
\end{aligned}
$$

	SAM0~SAM4					PU0~PU4				
Sector 0	1			0					E	
Sector 1		1			1	A	B			F
Sector 2		2					C			
Sector 3			3					D		

Loop Structure of MSR

```
curLU = LU0;
while(curLU != NULL ) {
    readScts = # of sectors to read in the current LU
    while(readScts > 0 ) {
        curPU = LU->firstPU;
        while(curPU != NULL ) {
            while(...) {
                conScts = # of consecutive PS's to read in curPU
                offset = the starting offset of these consecutive PS's in curPU
            }
            BML_READ(curPU, offset, conScts);
            readScts = readScts - conScts;
            curPU = curPU->next;
        }
    }
    curLU = curLU->next;
}
```

Loop1: iterates over LUs

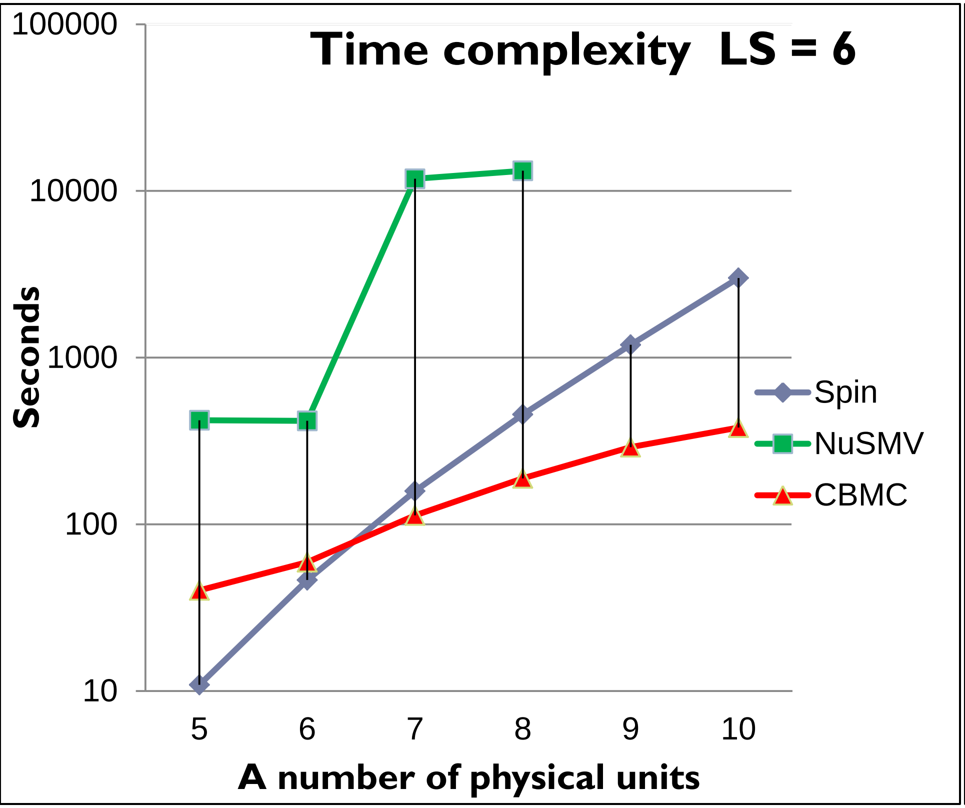
Loop2: iterates until the current LU is read completely

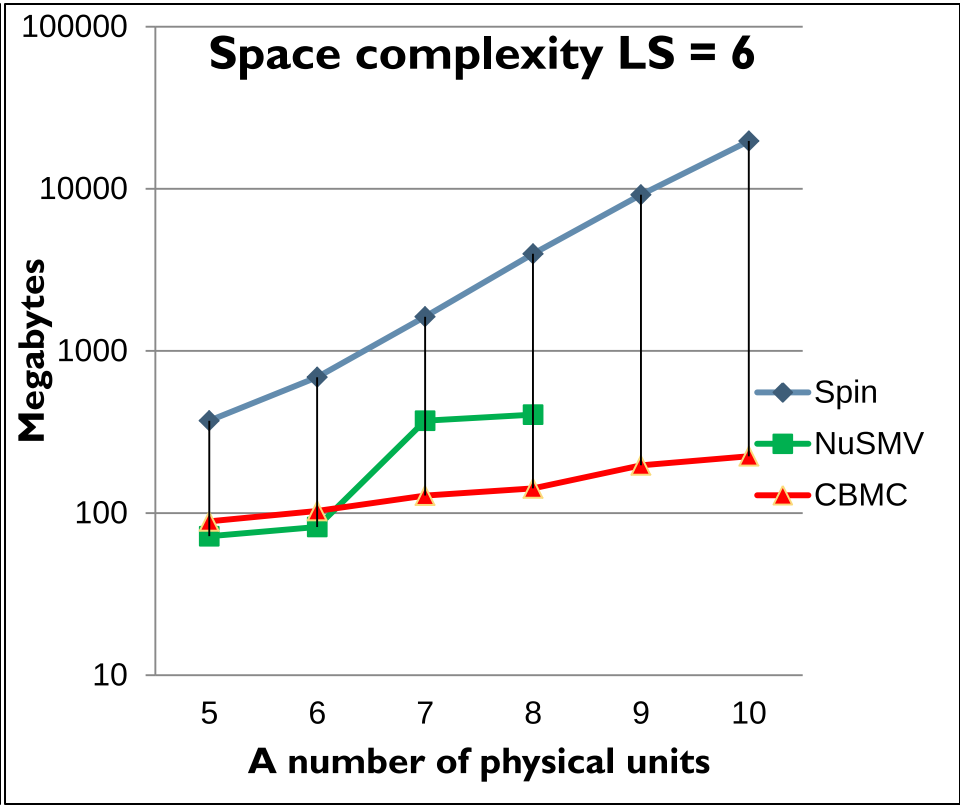
Loop3: iterates over PUs linked to the current LU

Loop4: identify consecutive PS's in the current PU

Model Checking Results of MSR [Spin'08, TSE'11]

- Verification of MSR by using NuSMV, Spin, and CBMC
- No violation was detected within |LS|<=8, |PU| <=10
  - 10^{10} configurations were exhaustively analyzed for |LS|=8, |PU|=10





Feedbacks from Samsung Electronics

Main challenge :

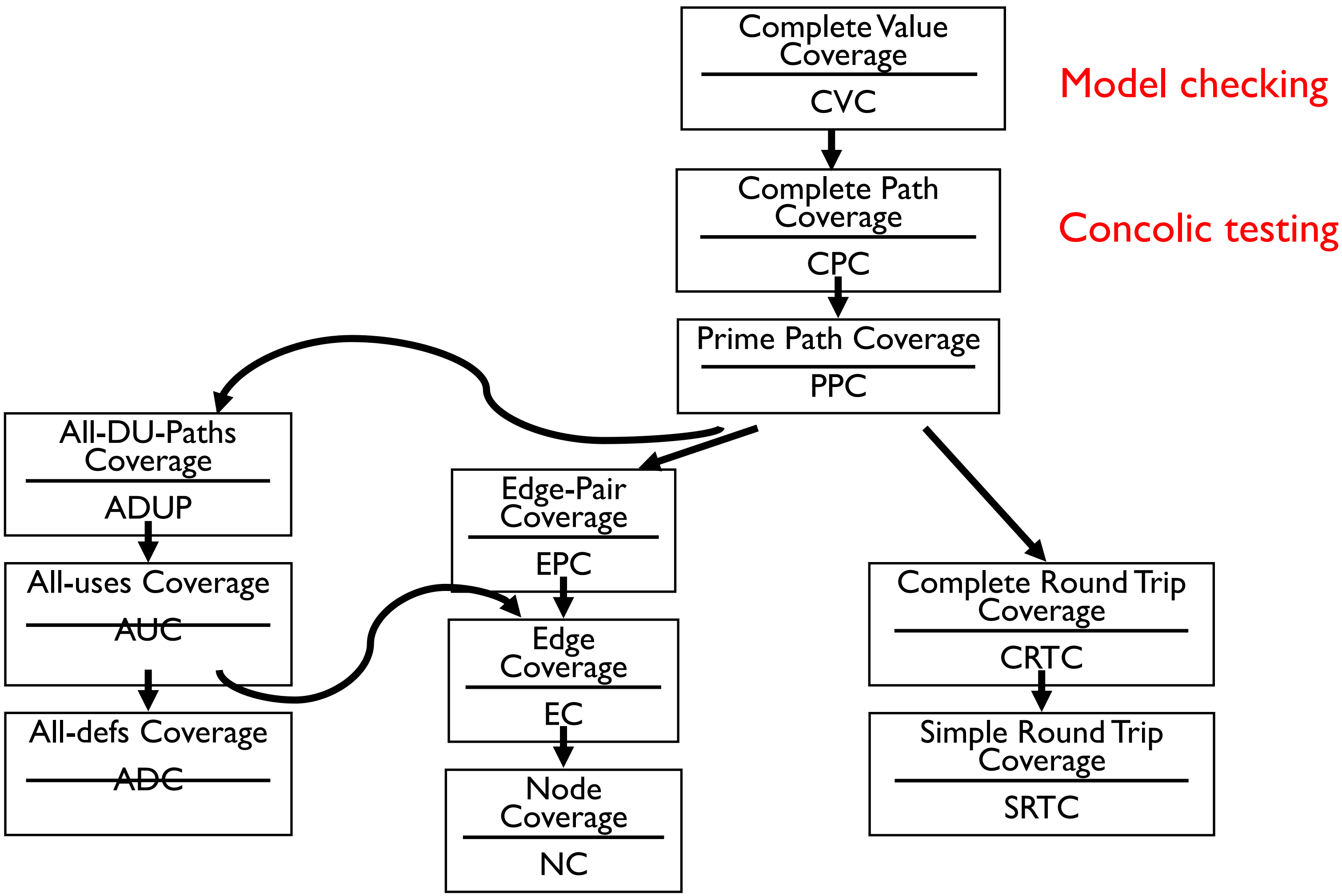
- IT industry is not mature enough to conduct unit testing

1. Current SW development of Samsung is not ready to apply unit testing
   - Tight project deadline does not allow defining detailed asserts and environment models
2. Needs large scalability even at the cost of accuracy
   - Rigorous automated tools for small unit (i.e., SW model checker) is of limited practical value
3. Many embedded SW components have dependency on external libraries
   - Pure analysis methods on source code only are of limited value
4. It is desirable to generate test cases as a result of the analysis.
   - Current SW V&V practice operates on test cases

Background on Concolic Testing

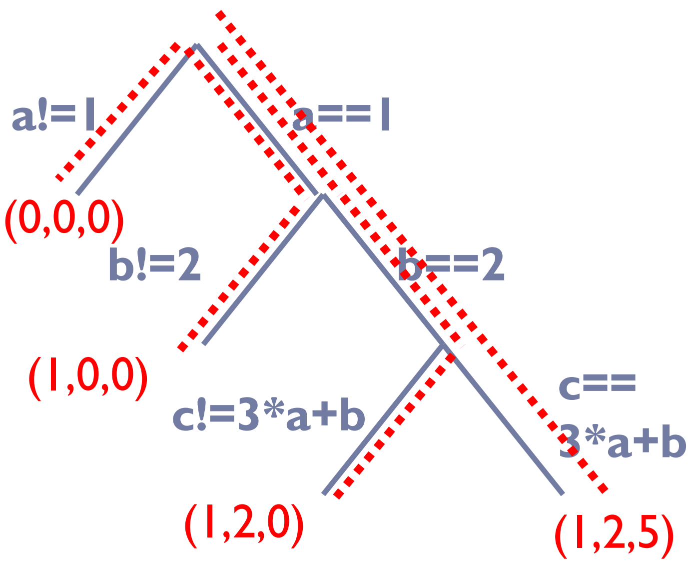
- **Conc**rete runtime execution guides symb**olic** path analysis
  - a.k.a. dynamic symbolic execution (DSE), white-box fuzzing
- Automated test case (TC) generation technique
  - Applicable to a large target program (no memory bottleneck)
  - Applicable to testing stages seamlessly
  - External binary library can be handled (partially)
- Explicit path model checker
  - All possible execution paths are explored based on the generated TCs
  - Anytime algorithm
    - User can get partial analysis result (i.e., TCs) anytime
  - Analysis of each path is independent from each other
    - Parallelization for linear speed up
    - Ex. Scalable Concolic testing for Reliability (SCORE) framework [ICST'12a]

Hierarchy of SW Coverages



Concolic Testing Example

```
// Test input a, b, c
void f(int a, int b, int c) {
    if (a == 1) {
      if (b == 2) {
          if (c == 3*a + b) {
              target();
} }  } }
```



- Random testing
  - Probability of reaching Error( ) is extremely low
- Concolic testing generates the following 4 test cases
  - (0,0,0): initial random input
    - Obtained symbolic path formula (SPF) ϕ: a!=1
    - Next SPF ψ generated from ϕ: !(a!=1)
  - (1,0,0): a solution of ψ (i.e. !(a!=1))
    - SPF ϕ: a==1 && b!=2
    - Next SPF ψ: a==1 && !(b!=2)
  - (1,2,0)
    - SPF ϕ: a==1 && (b==2) && (c!=3*a +b)
    - Next SPF ψ: a==1 && (b==2) && !(c!=3*a +b)
  - (1,2,5)
    - Covered all paths and target() reached

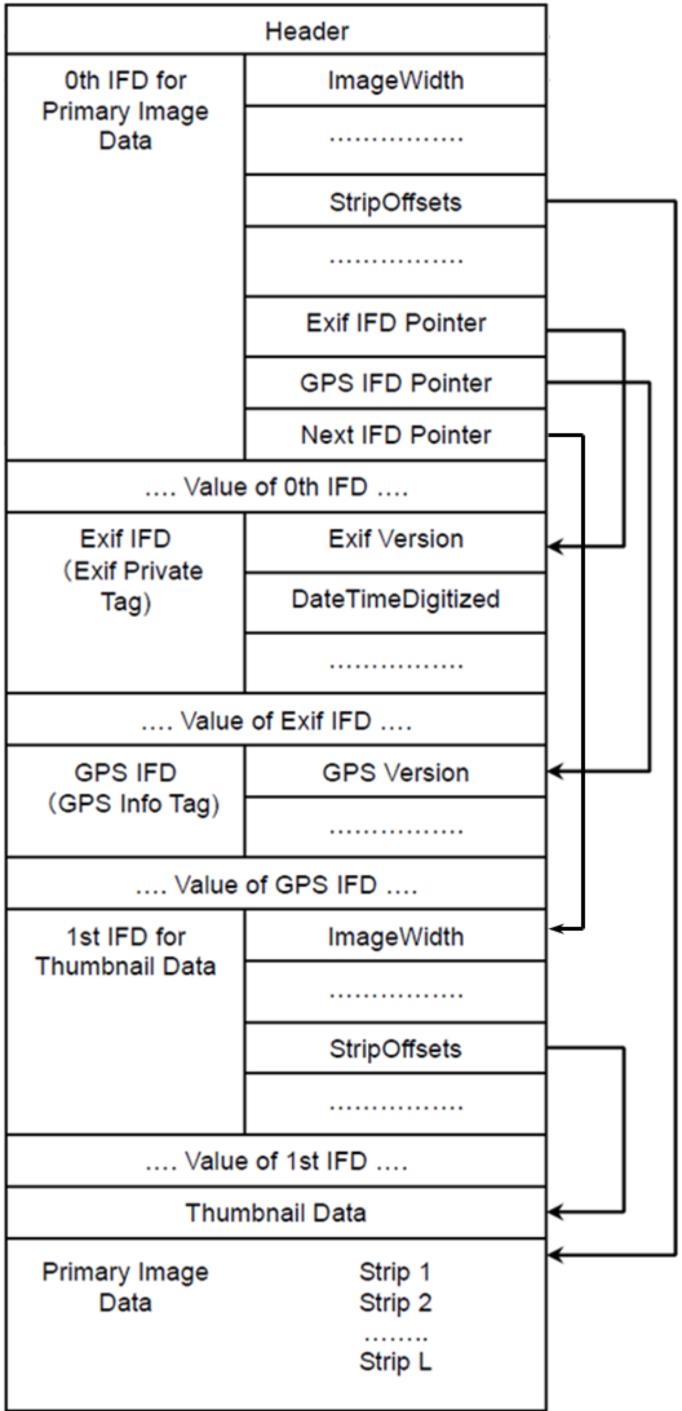
Part II: Experience from Concolic Testing using CREST

Target system: Samsung Smartphone Platform

- Unit-level testing
  1. Busybox ls (1100 LOC)
     - 98% of branches covered and 4 bugs detected
  2. Samsung security library (2300 LOC)
     - 73% of branches covered and a memory violation bug detected
- System level testing
  1. Samsung Linux Platform (SLP) file manager
     - detected an infinite loop bug
  2. 10 Busybox utilities
     - Covered 80% of the branches with 40,000 TCs in 1 hour
     - A buffer overflow bug in grep was detected
  3. Libexif
     - 300,000 TCs in 4 hours
     - 1 out-of-bound memory access bug, 1 null pointer dereferences, and 4 divide-by-0 bugs were detected

LibEXIF (Exchangeable Image File Format)

- libexif contains 238 functions in C (14KLOC)
- An IFD consists of
  - a 2 byte counter to indicate a number of tags in the IFD, tag arrays, 4 byte offset to the next IFD.
- Each tag consists of
  - tag id (2 bytes), type (2 bytes), count (i.e., a number of values) (4 bytes), value (or offset to the value if the value is larger than 4 bytes) (4 bytes).
- Manufacturer note tag is used for manufacturers of EXIF writers to record any desired information
  - Camera manufactures define a large number of their own maker note tags
  - maker note tags are not specified in the official EXIF specification.
  - Ex. Canon defines more than 400 maker note tags.



Testing Strategies

- Open source oriented approach
  - Focusing on runtime failure bugs only
    - Null-pointer dereference, divide-by-0, out-of-bound memory accesses
- Baseline concolic testing
  - Input EXIF tag size fixed at 244 bytes
  - Full symbolic
- Focus on the maker note tags w/ concrete image files.
  - 5 among 10 largest functions are for maker notes
  - These 5 functions takes 27% of total branches
- Compare two popular Concolic testing tools
  - CREST-BV and KLEE
- Comparison with Coverity Prevent

Testing Result 1

Table I
STATISTICS ON THE BASELINE CONCOLIC TESTING EXPERIMENTS BY USING KLEE

Time option (sec)	DFS			Random path			Random search			Covering new			DFS + covering new			Total of the 5 search strategies			# of bugs dete-cted	TC gen. speed (#/sec)
	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)		
900	1001	1289	8.1	2577	2280	11.1	2294	3192	11.1	2574	3072	11.1	1954	2022	11.1	10400	11855	11.1	1	1.1
1800	1903	2450	8.1	5530	4121	11.1	4832	5277	11.1	4944	4083	19.7	4928	3089	19.7	22137	19020	19.7	1	0.9
3600	3705	4868	8.1	10506	7084	11.1	9406	9945	19.1	12609	4543	20.4	10018	7685	19.7	46244	34125	20.4	1	0.7

Table II
STATISTICS ON THE BASELINE CONCOLIC TESTING EXPERIMENTS BY USING CREST-BV

Correspon-ding KLEE time option (sec)	DFS			Random path			Control flow graph (CFG) based			Total of the 3 search strategies			# of bugs dete-cted	TC gen. speed (#/sec)
	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)		
900	1001	12671	20.2	2577	100934	9.3	900	25191	21.8	4478	138796	22.3	1	31.0
1800	1903	22317	20.2	5530	171531	9.3	1800	25752	21.8	9233	219600	22.3	1	23.8
3600	3705	42499	20.3	10506	259625	10.3	3600	65644	21.8	17811	367768	22.3	1	20.6

- Out-of-bound memory access bug detected
  - exif_data_load_data () of exif-data.c as follows (line 2):

```
1:if (offset + 6 + 2 > ds) { return; }
2:n = exif_get_short(d+6+offset, ...)
```

Testing Result 2

Table III
STATISTICS ON THE CONCOLIC TESTING WITH FOCUS ON MAKER NOTE TAGS WITH 6 IMAGE FILES BY USING KLEE

Time option (sec)	DFS (Sum on the 6 files)			Random path (Sum on the 6 files)			Random search (Sum on the 6 files)			Covering new (Sum on the 6 files)			DFS+covering new (Sum on the 6 files)			Total of the 5 strategies on the 6 files each			# of bugs dete-cted	TC gen. speed (#/sec)
	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)		
900	5424	15804	44.5	5526	4800	44.3	5592	6684	44.7	5994	20454	44.7	5880	23424	44.7	28416	71166	49.2	1	2.5
1800	10830	24936	44.7	11010	8172	44.7	11154	10758	44.7	11646	24492	44.7	11652	34890	44.7	56292	103248	49.2	1	1.8
3600	21642	39270	44.7	21996	11342	44.7	22416	15378	45.0	23142	29988	45.0	23310	48270	45.0	112506	144248	49.5	1	1.3

Table IV
STATISTICS ON THE CONCOLIC TESTING WITH FOCUS ON MAKER NOTE TAGS WITH 6 IMAGE FILES BY USING CREST-BV

Correspon-ding KLEE time option (sec)	DFS (Sum on the 6 files)			Random path (Sum on the 6 files)			CFG based (Sum on the 6 files)			Total of the 3 strategies on the 6 files each			# of bugs dete-cted	TC gen speed (#/sec)
	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)	time (sec)	# of TC	Br.cov. (%)		
900	5424	93645	48.7	5526	130387	58.7	5400	98800	56.1	16350	322832	66.2	5	19.7
1800	10830	174173	48.7	11010	245873	59.3	10800	181362	56.5	32640	601408	67.1	5	18.4
3600	21642	309931	48.7	21996	433570	59.6	21600	325261	57.4	65238	1068762	68.1	5	16.4

- KLEE detected 1 null-pointer-dereference
- CREST-BV detected 4 divide-by-0 bugs in addition

- Null-pointer-dereference bug

```
for(i=0;i<sizeof(table)/sizeof(table[0]);i++)
  //t is a maker note tag read from an image
  if (table[i].tag==t) {
    //Null-pointer dereference occurs!!!
    if(!*table[i].description)
            return "";
```

- Divide-by-0 bug

```
vr=exif_get_rational(...);
//Added for concolic testing
assert(vr.denominator!=0);
a = vr.numerator / vr.denominator
```

Testing Result 3

- Comparison with Coverity Prevent
  - Prevent detected the following null-pointer dereference bug, which KLEE/CREST-BV did not detect
    - because test-mnote.c does not call the buggy function.

```
if(!loader||(loader->data_format ... ) {
  exif_log(loader->log, ...);
```

- However, no bugs detected by concolic testing was detected by Prevent
  - Not surprising
    - (Prevent spent only 5 minutes to analyze libexif)

Lessons Learned from Real-world Application

- Practicality of Concolic testing
  - 1 null-pointer dereference, 1 out-of-bound memory access, and 4 divide-by-0 in reasonable time
  - Note that
    - libexif is very popular OSS tampered by millions of users
    - we did not have background on LIBEXIF!!!
- Importance of Testing Methodology/Strategy
- Still state space explosion is a big obstacle
  - Average length of symbolic path formula = 300

    => In theory, there exist 2^{300} test cases to test

Conclusion and Future Work

- Formal verification techniques really work in IT industry !
  - Model checking and concolic testing detected hidden bugs in industrial embedded software
- To alleviate the limitations of concolic testing
  - External function summaries through dynamic invariance generation
  - Develop a new search strategy for fast branch coverage
- Data mining on a huge set of runtime execution information
  - (semi) Automated oracle generation through dynamic invariant generation
  - Automated debugging
- Technical papers can be downloaded at http://pswlab.kaist.ac.kr