Formal Verification of a Flash Memory Device Driver
- an Experience Report

Moonzoo Kim, Yunho Kim

Provable Software Lab. CS Dept. KAIST

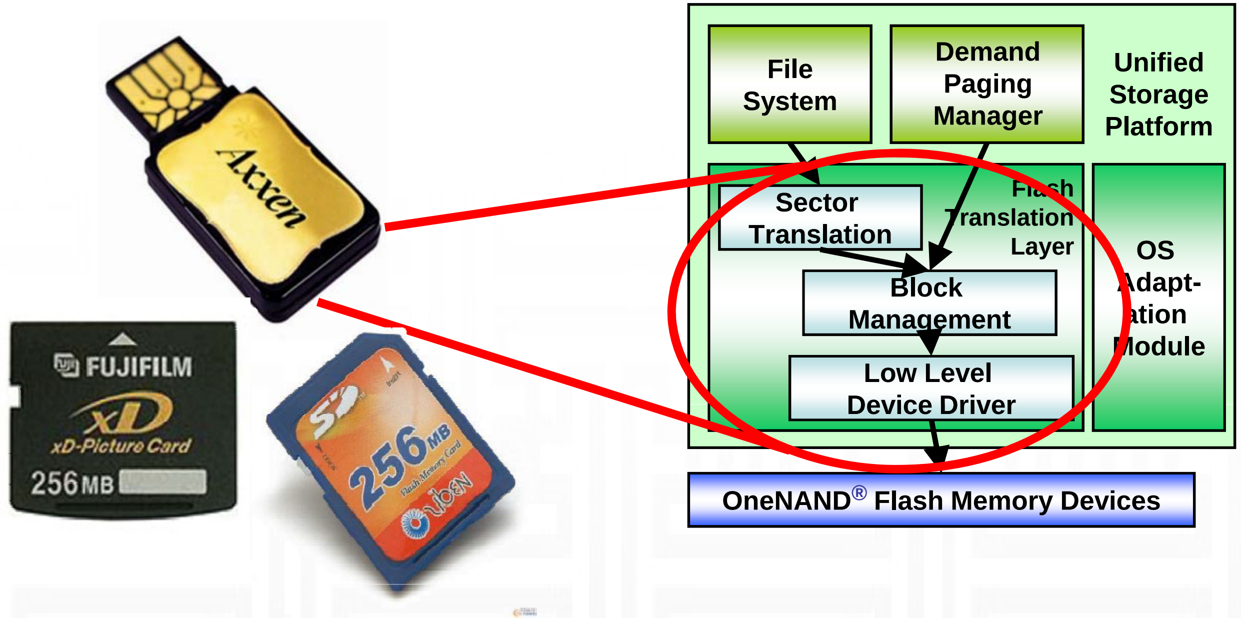
Yunja Choi

School of EECS, Kyungpook National Univ.

Hotae Kim

Samsung Electronics

Summary of the Talk



- **In 2007, Samsung requested to debug the device driver for the Samsung OneNAND™ flash memory, by using model checkers, for 6 months. This presentation describes a part of the result from the project.**

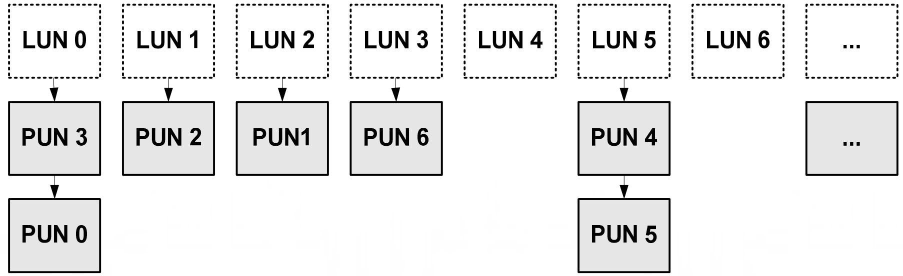
Overview

- **Background**
  - Overview of the Unified Storage Platform (USP)
  - Sector Translation Layer (STL)
  - Multi-Sector Read operation (MSR)
- **Model Checking MSR**
  - Reports on the following three aspects
    - Target system modeling
    - Environment modeling
    - Performance analysis on the verification
- **Three different types of model checkers are used**
  - BDD based symbolic model checking (NuSMV)
  - Explicit model checking (Spin)
  - C-bounded model checking (CBMC)

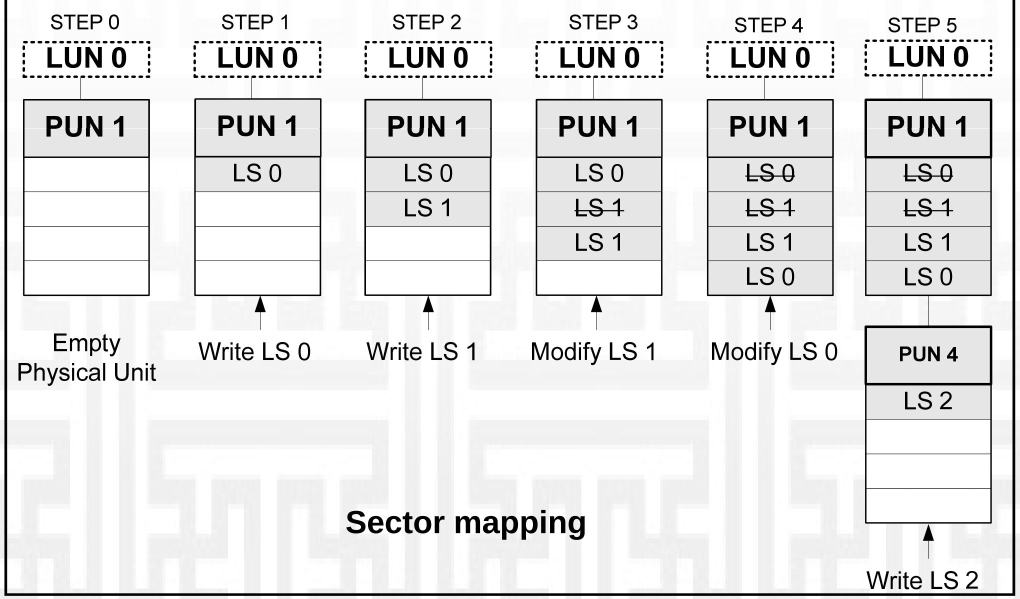
PART I: Background

- **Logical-to-physical sector translation**
  - Example of possible data distributions
- **Unified Storage Platform (USP)**
  - Block diagram
  - Code statistics
- **Multi-Sector Read operation (MSR)**
  - Pseudo structure

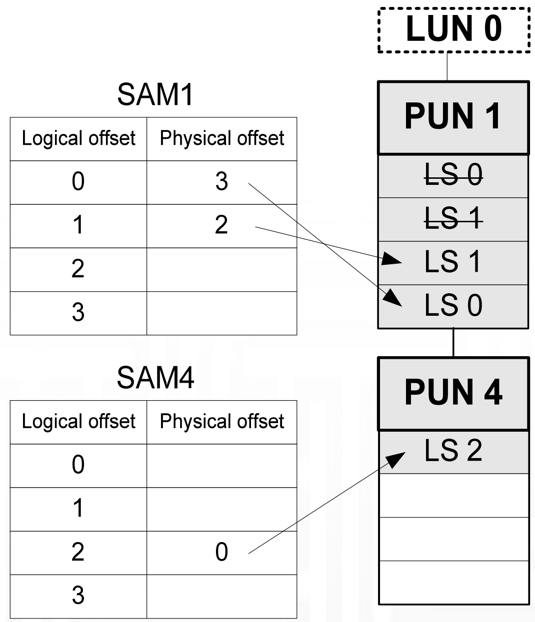
Logical to Physical Sector Mapping



1:N mapping from a LUN to PUNs



Sector mapping



Sector Allocation Map (SAM)

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

Examples of Possible Data Distribution

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

(a) A distribution of "ABCDEF"

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

(b) Another distribution of "ABCDEF"

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

(c) A distribution of "FEDCBA"

- **Assumptions**
  - there are 5 physical units
  - each unit has 4 sectors
  - each sector is 1 byte long

Environment Model

- **Environment model creation**
  - The environment of MSR (i.e., PUs and SAMs configurations) can be described by invariant rules. Some of them are
    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=3 k=3 j=2
    3. *For one LS, there exists only one PS that contains the value of the LS:*

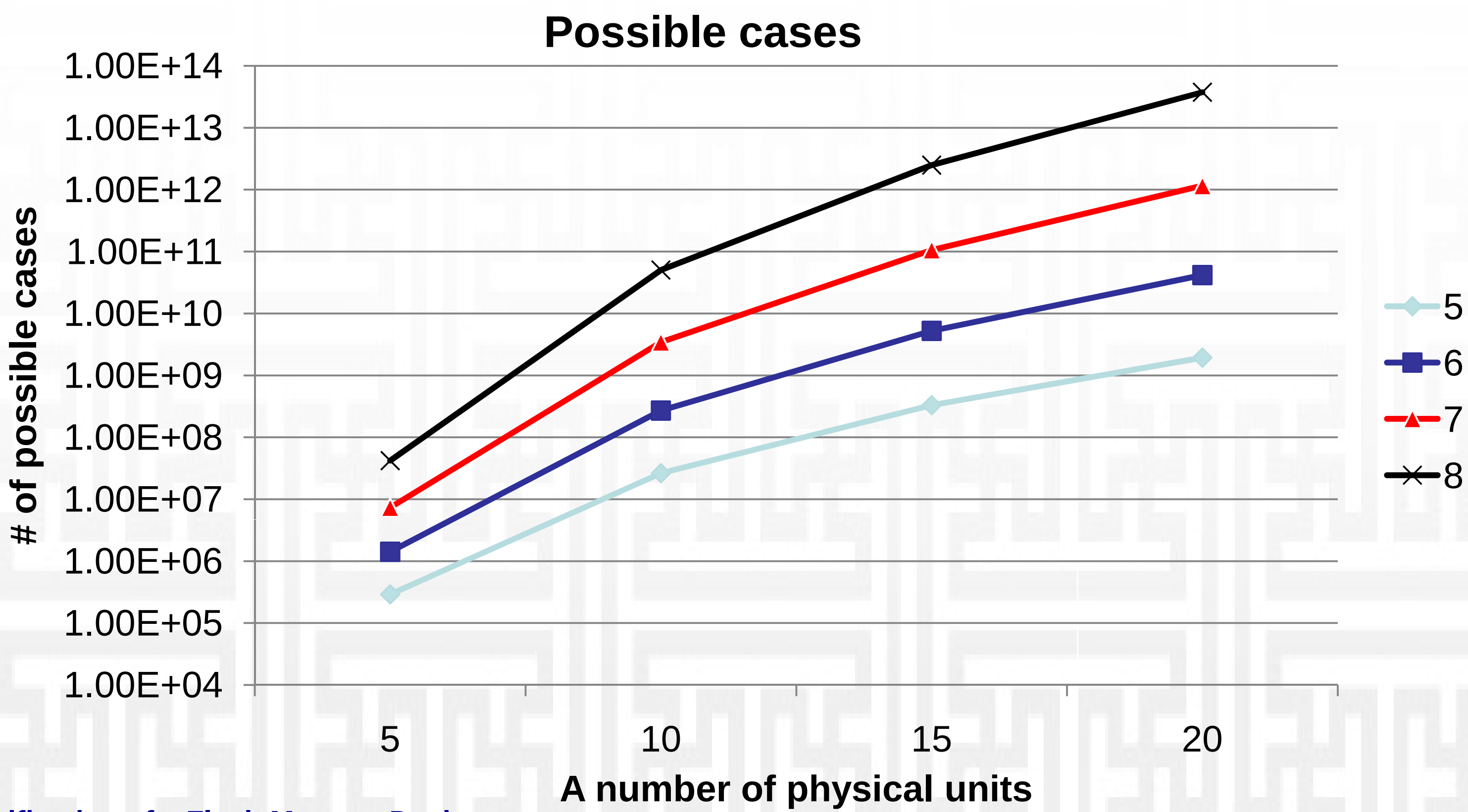
       The PS number of the *i* th LS must be written in only one of the (*i* mod *4*) th offsets of the SAM tables for the PUs mapped to the corresponding LU.

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

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

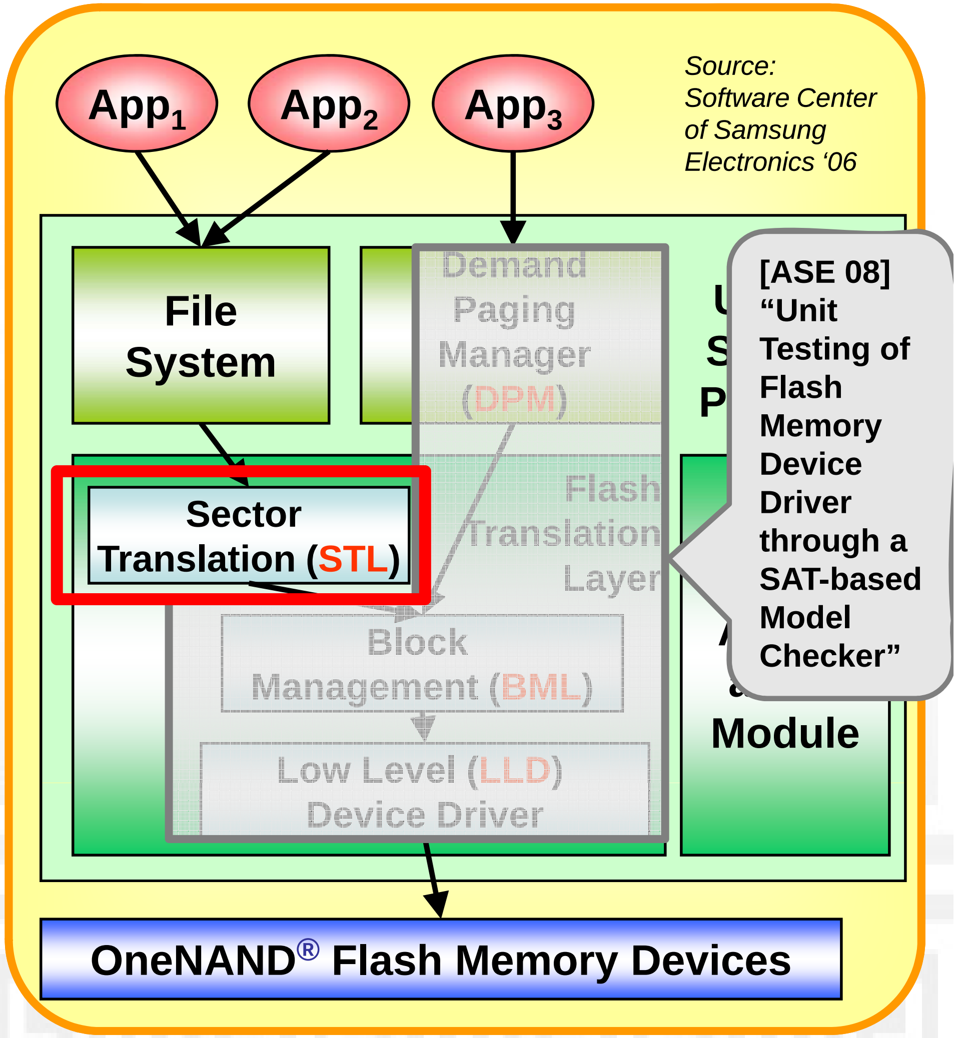
Exponential Increase of Distribution Cases

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



Overview of the OneNAND® Flash Memory

- **Characteristics of OneNAND®**
  - Each memory cell can be written limited number of times only
    - Logical-to-physical sector mapping
    - Bad block management
    - Wear-leveling
  - Performance enhancement
    - Multi-sector read/write
    - Asynchronous operations
    - Deferred operation result check



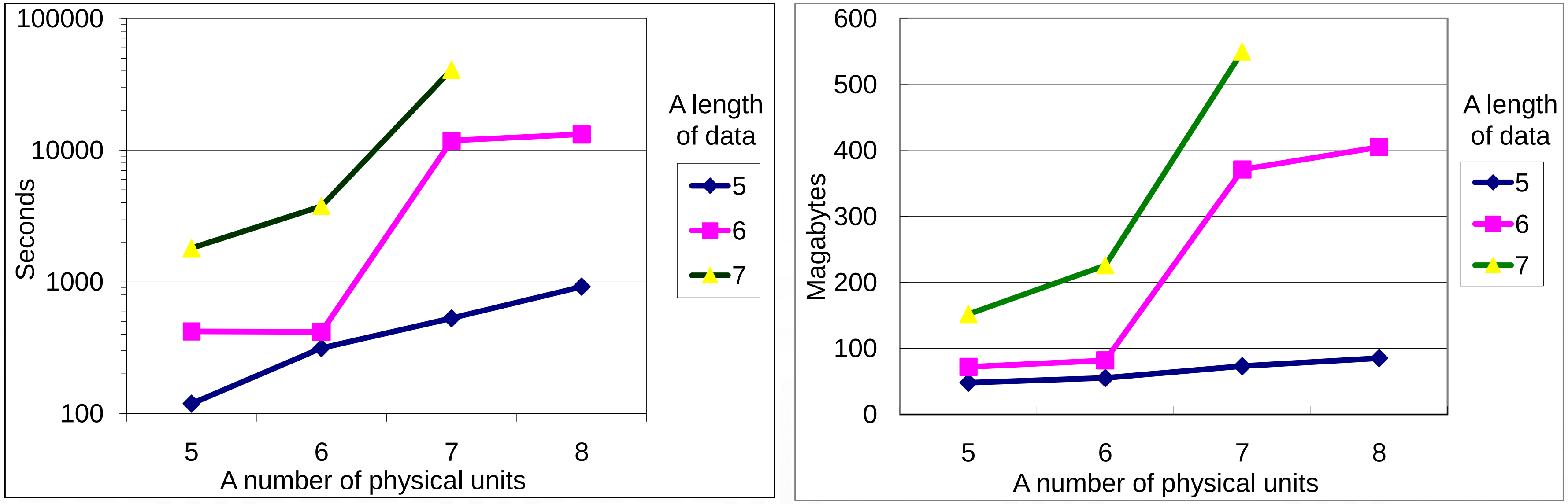
PART II: Model Checking Results

- **Verification of MSR by using NuSMV, Spin, and CBMC**
  - NuSMV: BDD-based symbolic model checker
  - Spin: Explicit model checker
  - CBMC: C-bounded model checker
- **The requirement property is to check**
  - after_MSR -> ($\forall$ i. logical_sectors[i] == buf[i])
- **We compared these three model checkers empirically**

Excerpts of the SMV Model

```
MODULE main

-- Variable declaration
VAR
   SAM    : array 0..4 of sam_type;
   PU     : array 0..4 of PU_type;
   buf    : array 0..4 of 0..5;
   nScts  : 0..5;

-- SPEC
INVARSPEC (after_first_do ->
PU[0].sect[0]=1 &
PU[0].sect[1]=2 &
PU[0].sect[2]=3 &
PU[0].sect[3]=4 &
PU[3].sect[0]=5)

init(buf[0]):=0;
--  if( pBuf==0 && 0 < nScts )
--       buf[0]= PU[PU_id].sect[nFirstOffset]
next(buf[0]):
  case after_fourth_do :
      case pBuf = 0 & 0 < nScts:  -- i=0
      case
          PU_id=0 & nFirstOffset=0: PU[0].sect[0];
          PU_id=0 & nFirstOffset=1: PU[0].sect[1];
          PU_id=0 & nFirstOffset=2: PU[0].sect[2];
          PU_id=0 & nFirstOffset=3: PU[0].sect[3];
          ...
          PU_id=4 & nFirstOffset=3 : PU[4].sect[3];
      esac;
  esac;

init(buf[1]):=0;
next(buf[1]):= ...
```

Verification Performance of NuSMV



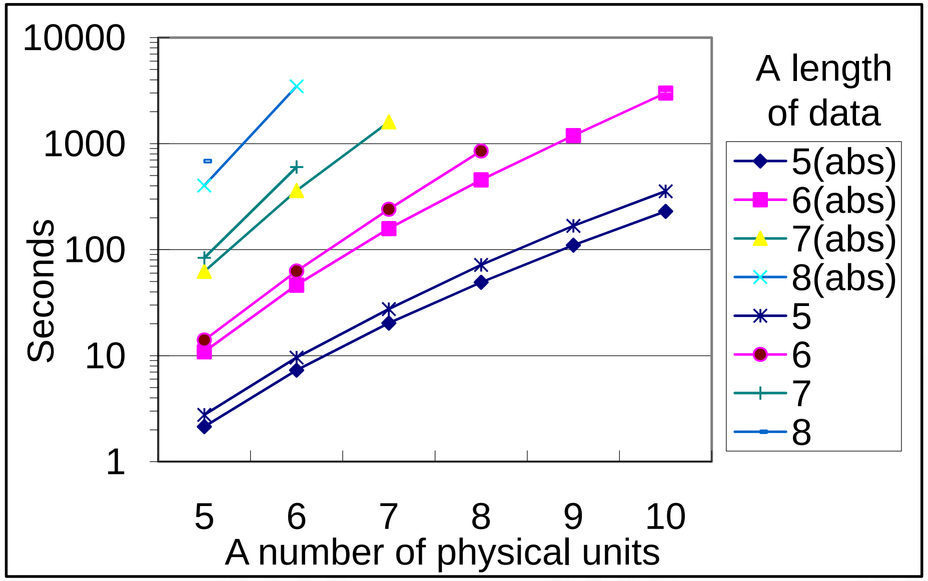
(a) Time consumption (b) Memory consumption

- **Verification was performed on the machine equipped with Xeon5160 (3Ghz, 32Gbyte Memory), 64 bit Fedora Linux 7, NuSMV 2.4.3**
- **The requirement property was proved correct for all the experiments (i.e., MSR is correct in this small model)**
  - For 7 sectors long data that are distributed over 7 PUs consumes more than 11 hours while consuming only 550 mb memory

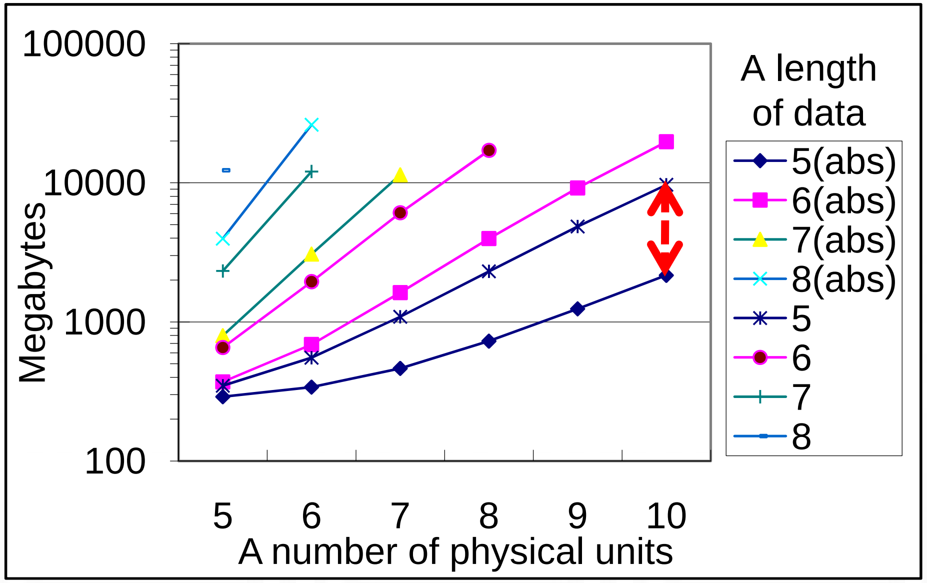
Excerpts of the Spin Model

```
active proctype SM_ReadSectors() {

  byte buf[NUM_LS_USED];
  byte nScts;
  byte nFirstOffset;
  byte nNumOfScts=NUM_LS_USED;
  byte nReadScts=nNumOfScts;
  byte nSamIdx;

  do  /* 1047: while (nNumOfScts >0) { */
  ::  nNumOfScts > 0 ->
     PU_id = lui[nLun];
     if        /* nReadScts = ... */
     :: (SECT_PER_U-nSamIdx)> nNumOfScts ->
        nReadScts = nNumOfScts;
     :: else->nReadScts =SECT_PER_U- nSamIdx;
     fi;
     nNumOfScts = nNumOfScts - nReadScts;

     do       /* line 1068: while (nReadScts > 0) */
     ::  (nReadScts > 0) ->  PU_id = lui[nLun];
         nFirstOffset=255;
         nScts=1; nReadScts--;
do  /* line 1075:  do {...  */
::  true;
   if /* line 1077: if(pstCurrent->pSam[nSamIdx]...*/
   :: SAM[PU_id].valid[nSamIdx]->  nFirstOffset =
     SAM[PU_id].offset[nSamIdx];nSamIdx++;
     do  /* line 1084:while (nReadScts > 0) { ...} */
     ::  (nReadScts > 0) ->
        if
        ::FirstOffset+nScts==
          SAM[PU_id].offset[nSamIdx] ->
          nScts++;nReadScts--;nSamIdx++;
        :: else-> break;
        fi;
     :: else->break;
     od;

     BML_MRead(PU_id,nFirstOffset,nScts,pBuf);
     break;
   :: else;
   fi;
   if /*line 1112: } while ( PU[PU_id].nil != true) */
   :: PU[PU_id].nil -> break;
   :: else;
   fi;
   PU_id++;
od;
}
```

Verification Performance of Spin



(a) Time consumption



(b) Memory consumption

- **The requirement property was satisfied**
- **The data abstraction technique shows significant performance improvement upto 78% of memory reduction and 35% time reduction (for 5 logical sectors data)**

# of physical units	5	6	7	8	9	10
Memory reduction	17%	38%	57%	68%	74%	78%
Time reduction	23%	24%	26%	32%	34%	35%

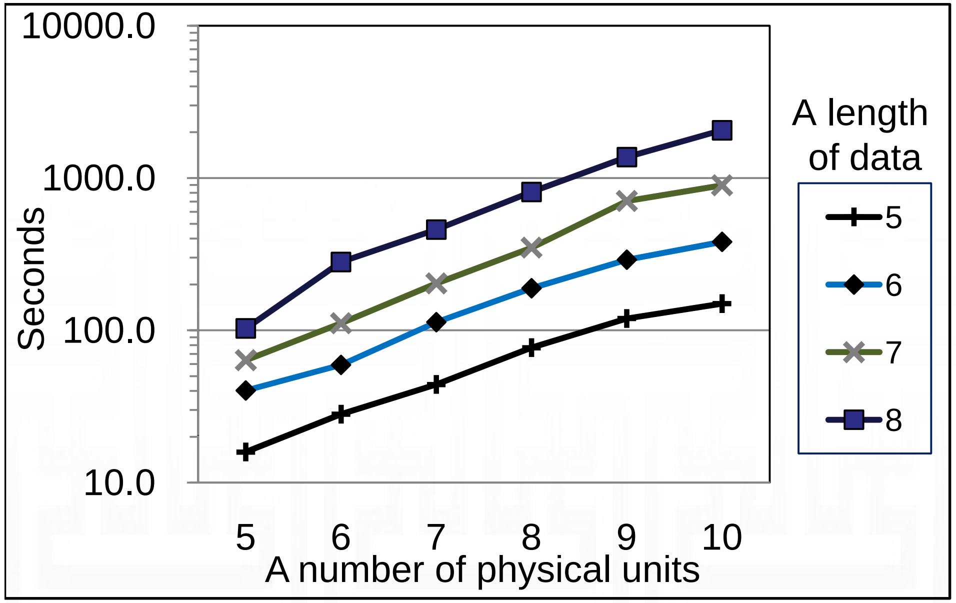
Modeling by CBMC

- **CBMC does not require an explicit target model creation**
- **An environment for MSR was specified using assume statements and the environment model was similar to the environment model in NuSMV**
- **For the loop bounds, we can get valid upper bounds from the loop structure and the environment setting**
  - The outermost loop: L times (L is a # of LUs)
  - The 2nd outermost loop: 4 times (one LU contains 4 LS's)
  - The 3rd outermost loop: M times (M is a # of PUs)
  - The innermost loop: 4 times (one PU contains 4 PS's)

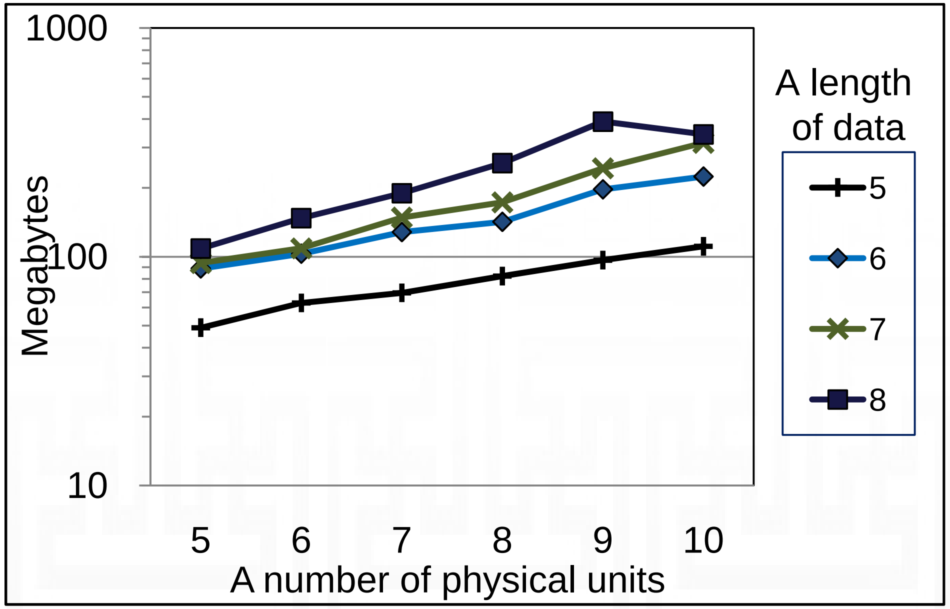
L=2, M=5

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

Verification Performance of CBMC



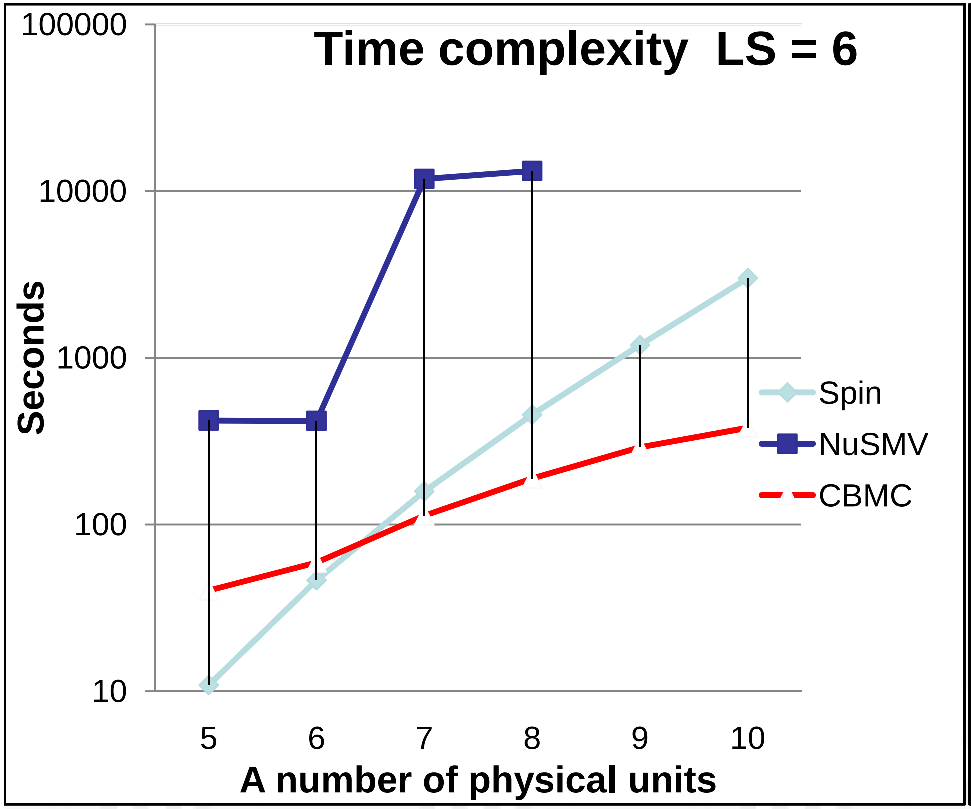
(a) Time consumption

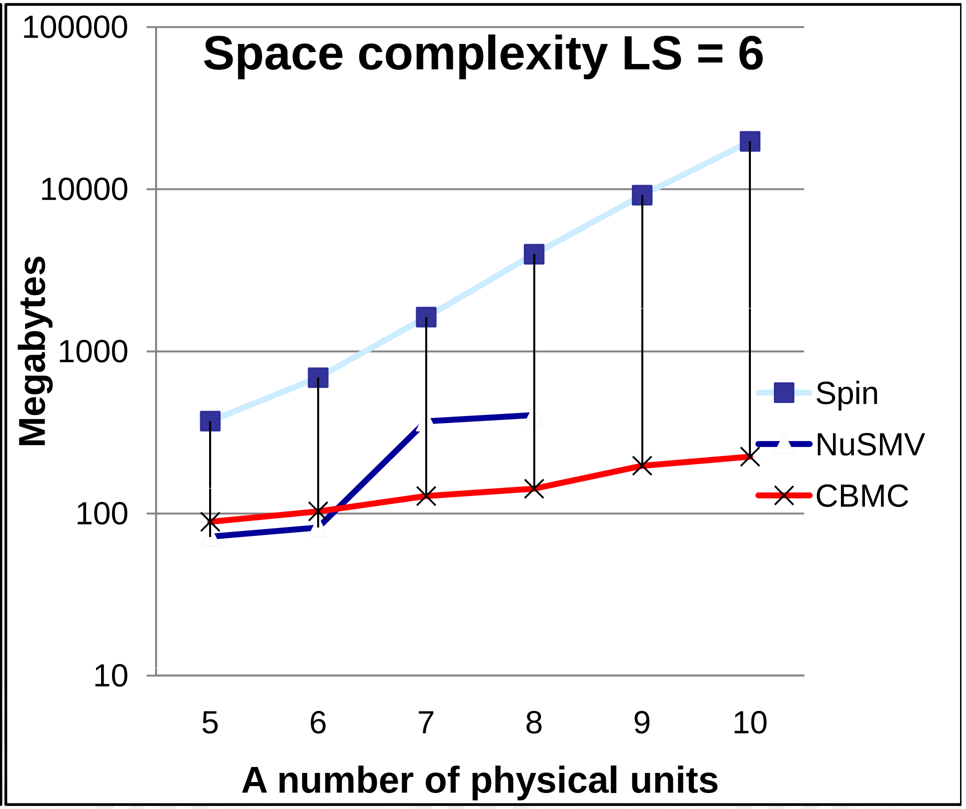


(b) Memory consumption

- **Exponential increase in both time and memory. However, the slope is much lower than those of NuSMV and Spin, which makes CBMC perform better for large problems**
- **A problem of 10 PUs and 8 LS's has 8.6×10^5 variables and 2.9×10^6 clauses.**

Performance Comparison





Conclusion

- **Application of Model Checking to Industrial SW Project**
  - Current off-the-shelf model checkers showed their effectiveness to debug a part of industrial software, if a target portion is carefully selected
  - Although model checker worked on a small scale problem, it still contributes due to its exhaustive exploration which is complementary to the testing result
- **Comparison among the Three Model Checkers**

	Modeling Difficulty	Memory Usage	Verification Speed
NuSMV	Most difficult	Good	Slow
Spin	Medium difficult	Poor	Fast
CBMC	Easiest	Best	Fastest