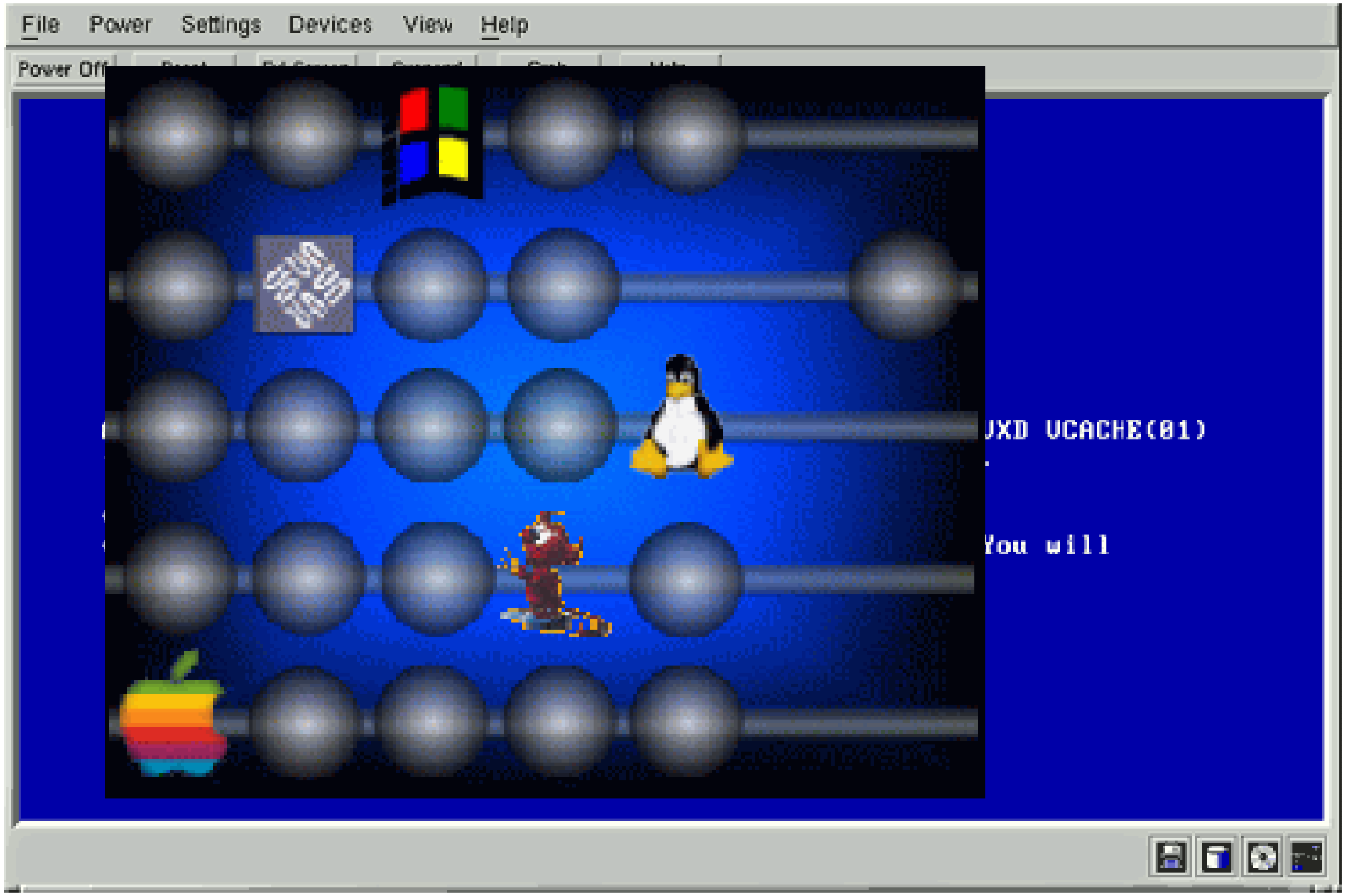
On Evaluating OS Dependability

The Fun (& Science ...) of Experimental Approaches ...

Neeraj Suri
Dept. of Computer Science
TU Darmstadt, Germany

suri@informatik.tu-darmstadt.de

Failure is not an option; it comes with the software!

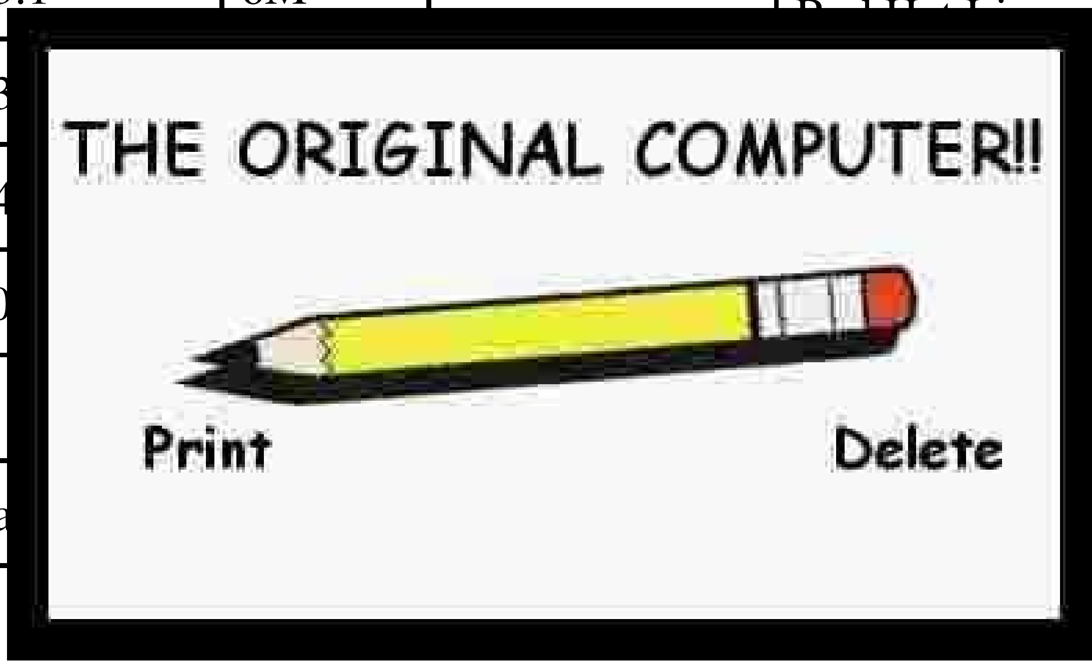


Talk Outline

- Why are experimental techniques useful for OS evaluation?
- Where to focus in the OS's?
- How to meaningfully use Fault Injection (FI) based experimentation to detect as many OS kernel robustness vulnerabilities as possible!
  - Where to inject
  - What to inject
  - When to inject

Why Experiment? What makes analyzing OS's hard?

Operating System	~ SLOC
Windows NT 3.1	6M
Windows NT 3	
Windows NT 4	
Windows 2000	
Windows XP	
Windows Vista	



Operating System	~ SLOC
6.2	17M
7.1	30M
ux 4.0	283M
	9.7M
	86M
6.32	12.6M

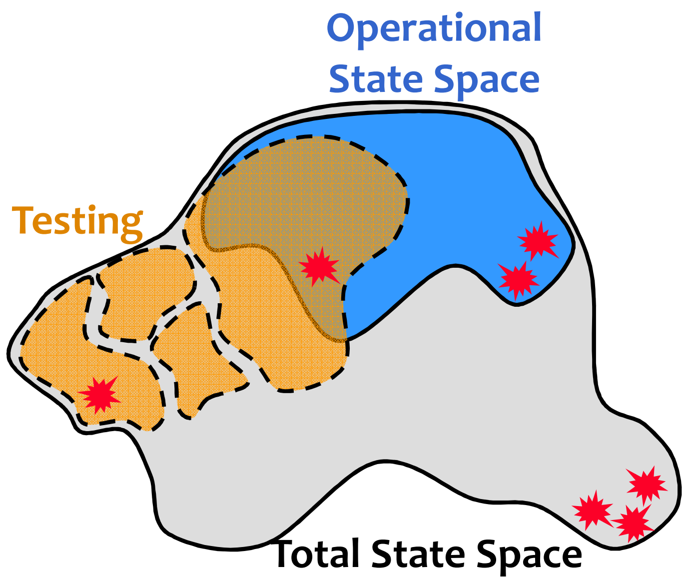
There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies. (C.A.R. Hoare)

The amount of damage one human being can do doubles every 18 months. (1st Corollary of Moore's Law)

OS Issues: Evaluation

What limits analytical approaches?

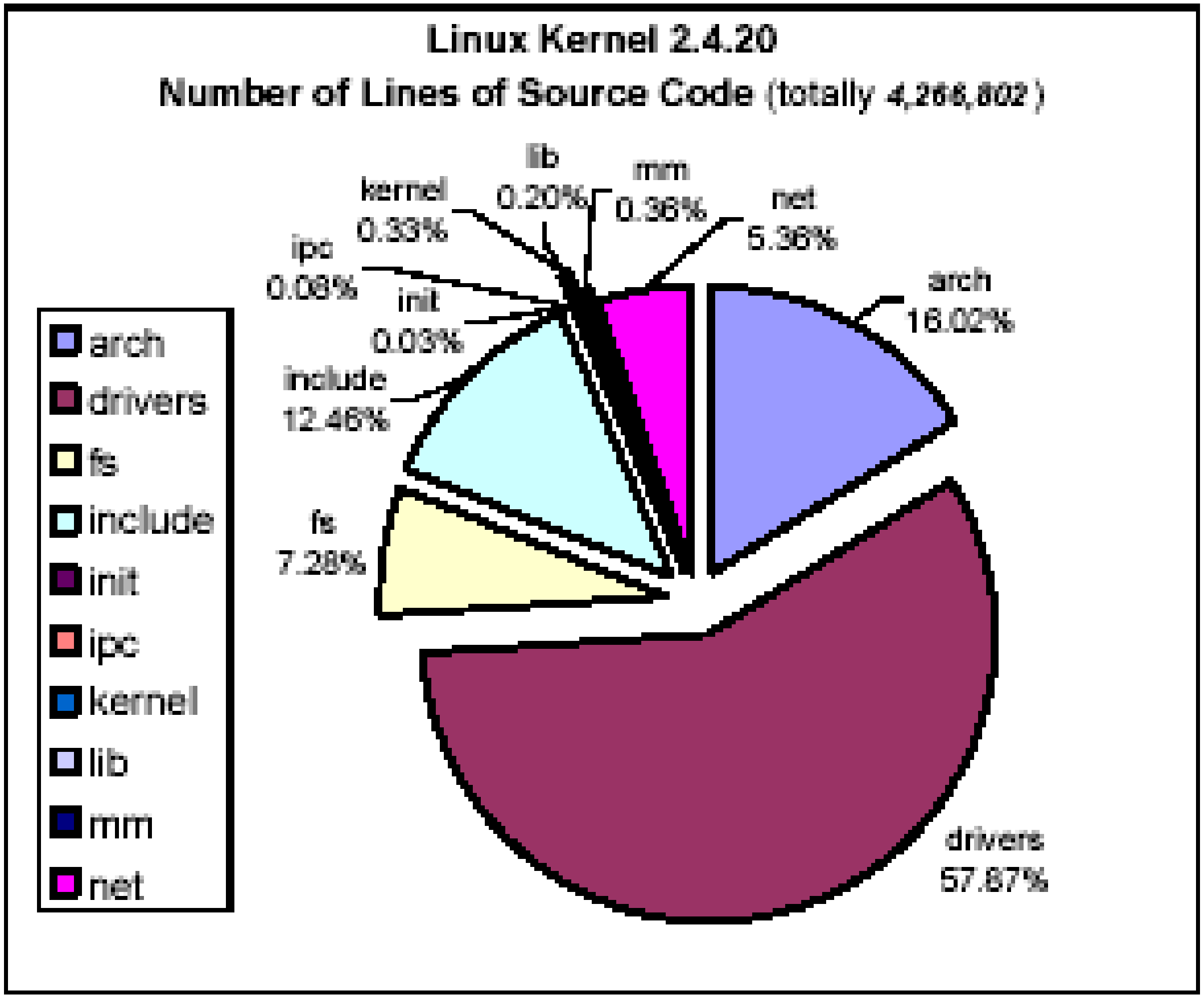
- Size & Complexity
  - every line of code? all program paths?
  - all transitions and states?
- Leaky SW (code, module, interface)
- Services/Applications variety
- Dynamic nature of interactions
- Load/Environment



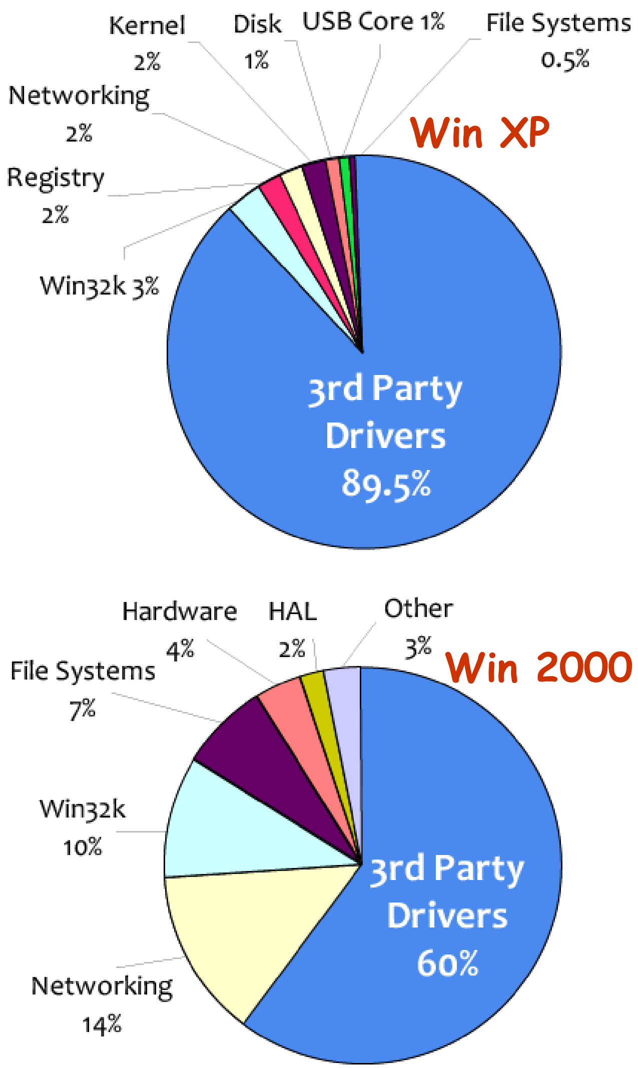
❖ Lets just focus on data errors to start…!

❖ No/Limited source code availability!

...what OS failures dominate?



...the kernel is often not the (big) problem



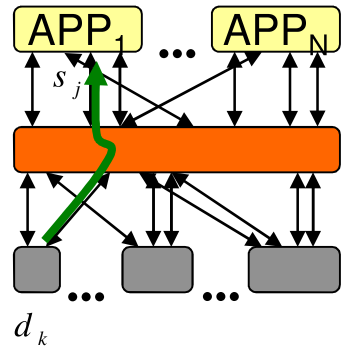
- Numerous: ~26K Ecosystem; 250 installed (100 active) in XP/Vista
- Immature: 25 new/100 "daily" revisions on Vista drivers
- Large & complex: 70% of Linux code base, Video drivers up to 2M LoC

- Access Rights: drivers often use kernel mode operations...

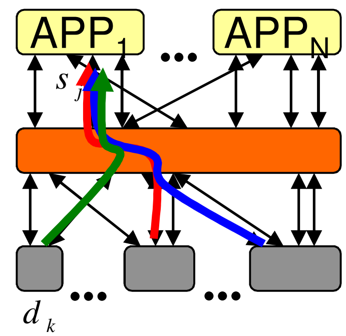
- WDM/WDK interface compliance but limited source code details known...

Driver Effects on OS Services (Dynamic Apps)

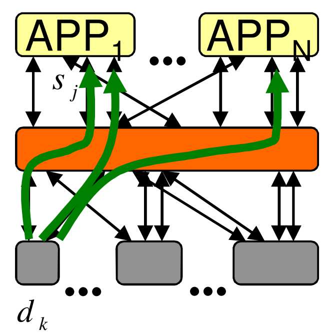
- Which triggers affect which service? Permeability

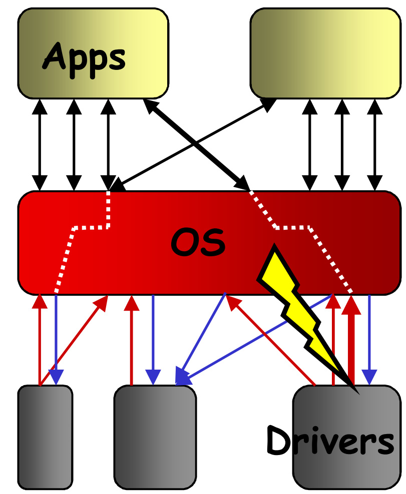


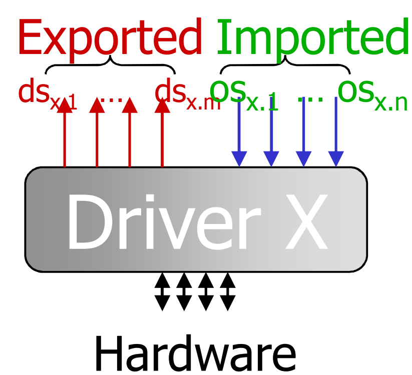
- Which service is most exposed? Exposure



- Which driver spreads the most errors? Diffusion







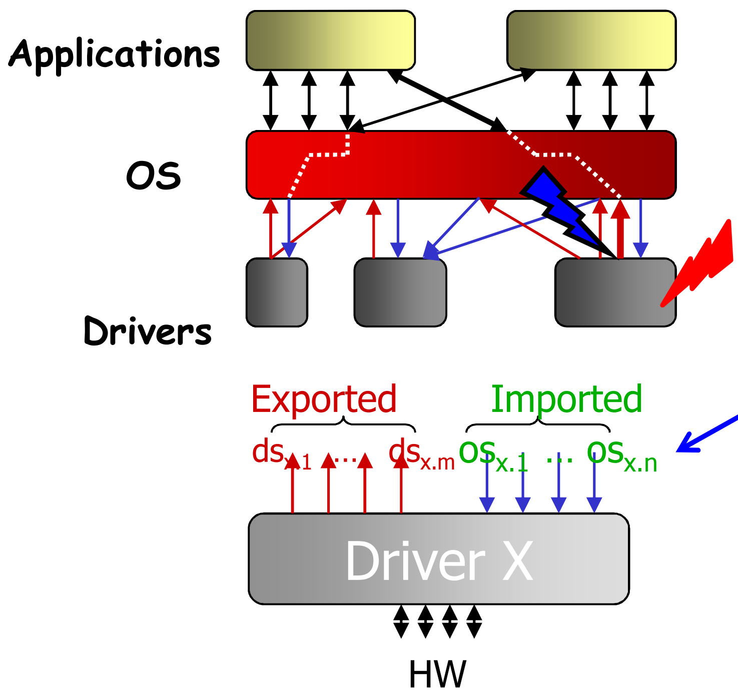
- BUT
  - Are we injecting at the right place?
    Where to inject [DSN 05/07]
  - Did we choose the right injection model? GIGO!
    What to inject [DSN 05/07; TOC 04]
  - Are we injecting at the right time?
    When to inject [ISSRE 07]

Framework/ Authors	Fault Location	Fault Type	Fault Latency	Injection Trigger
MAFALDA [8]	CUE Server IUE	SEU MBU DT	Transient Permanent	1^{st} occ.
Albinet et al. [7]	IUE	DT	Transient	1^{st} occ.
Kalakech et al. [22]	IUE	SEU DT	Transient	1^{st} occ.
Xception [11]	CUE Server IUE	SEU MBU DT FZ	Transient Intermittent Permanent	1^{st} occ. n^{th} occ. Timer X-call
G-SWFIT [13]	CUE Server	Coding mistakes	Permanent	1^{st} occ.
Medonça & Neves [28]	CUE Server	Coding mistakes	Permanent	1^{st} occ.
Johansson [18]	IUE	SEU DT FZ	Transient Intermittent Permanent	1^{st} occ. n^{th} occ. Timer X-call Call Block

Objective 1: "Where/What to Inject?"

- FI's effectiveness based on the chosen fault model being:
  (a) representative of actual perturbations, and (b) effective triggers
- Comparative evaluation of "effectiveness" of different injection models.

Fault-Injection: Fault Models, Failure Classes



- Injection Models
  - Data Type (DT)
  - Bit Flip (BF)
  - Fuzzing (FZ)
  - SEU (bit flips – code mutations)

Failure Class	Description
No Failure	No observable effect
Class 1	Error propagated, but still satisfied the OS service specification
Class 2	Error propagated and violated the service specification
Class 3	The OS hung or crashed

Models: Data-Type (DT), Bit Flip (BF), Fuzzing (FZ)

DT

int foo(int a, int b) {…}

```
foo(0x45a209f1,…
    ↓
foo(0x80000000,…
```

- #boundary cases depending on data type (int, char, boolean, …)
- C-types: int (long, short…)
- Requires tracking of the types for correct injection
- Complex implementation but scales

BF

int foo(int a, int b) {…}

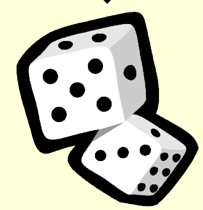
```
foo(0x45a209f1,…
          ↓
   ..0010000001001

   ..0010100001001

foo(0x45a289f1,…
```

- Typically 32 cases per parameter
- Tedious … but can be mechanized

FZ

int foo(int a, int b) {…}

```
foo(0x45a209f1,…
     ↓
```



```
foo(0x17af34c2,…
```

- Selective # of cases – uniform dist. across parameter range
- Simple implementation

Target Drivers

Driver	Description
cerfio_serial	Serial port
91C111	Ethernet
atadisk	CompactFlash

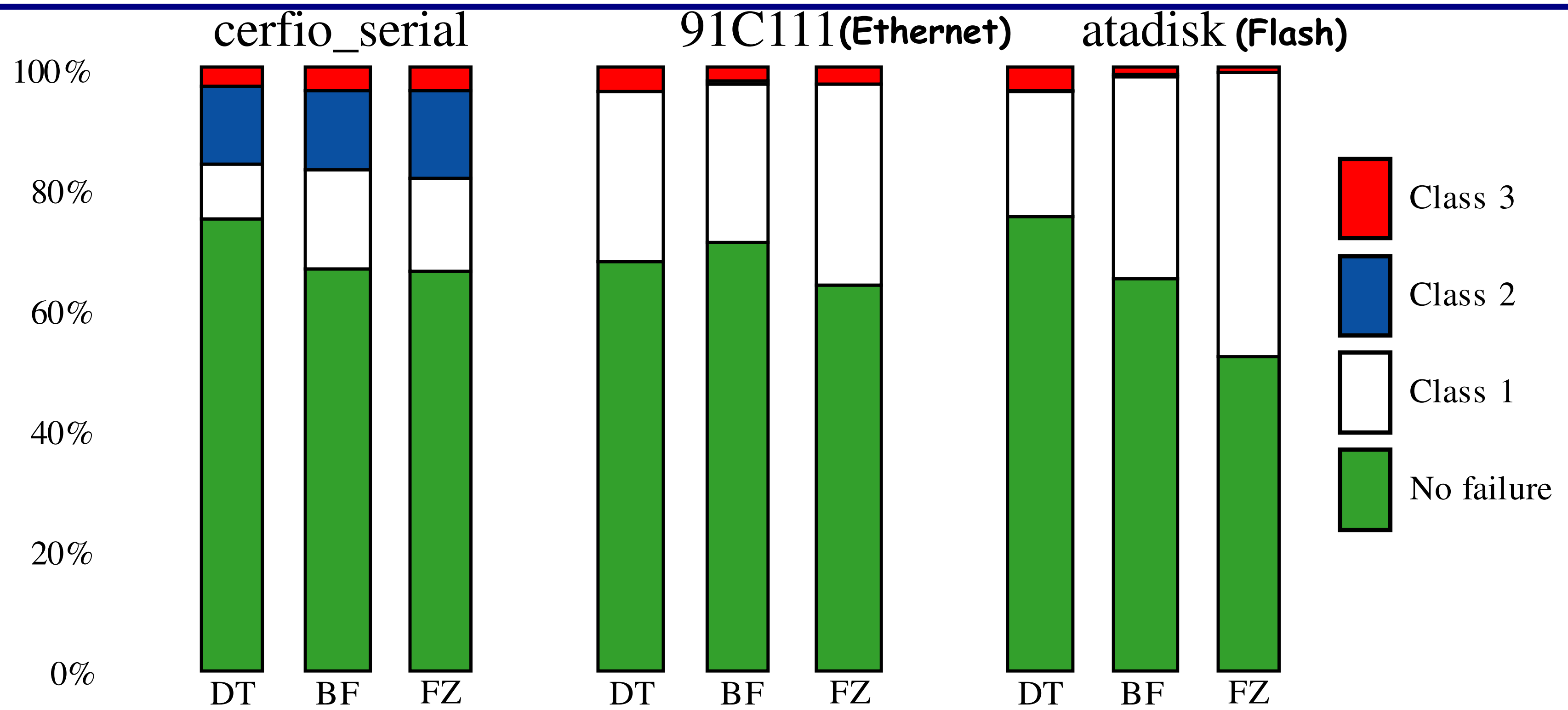
Compare Injection Models on:

- Number of failures
- Effectiveness
- Experimentation Time
- Identifying services
- Error propagation

Comparative Effort

Driver	Description	#Injection cases		
		DT	BF	FZ
cerfio_serial	Serial port	397	2362	1410
91C111	Ethernet	255	1722	1050
atadisk	CompactFlash	294	1658	1035

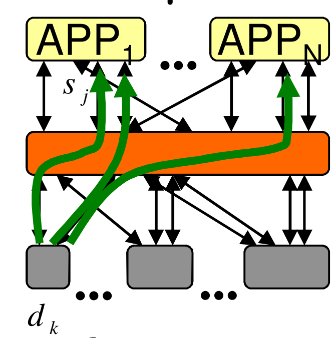
Failure Classes Triggering (Win CE.NET)



Driver Diffusion (Class 3)

Drivers	DT	BF	FZ
cerfio_serial	1.50	1.05	1.56
91C111	0.73	0.98	0.69
atadisk	0.63	1.86	0.29

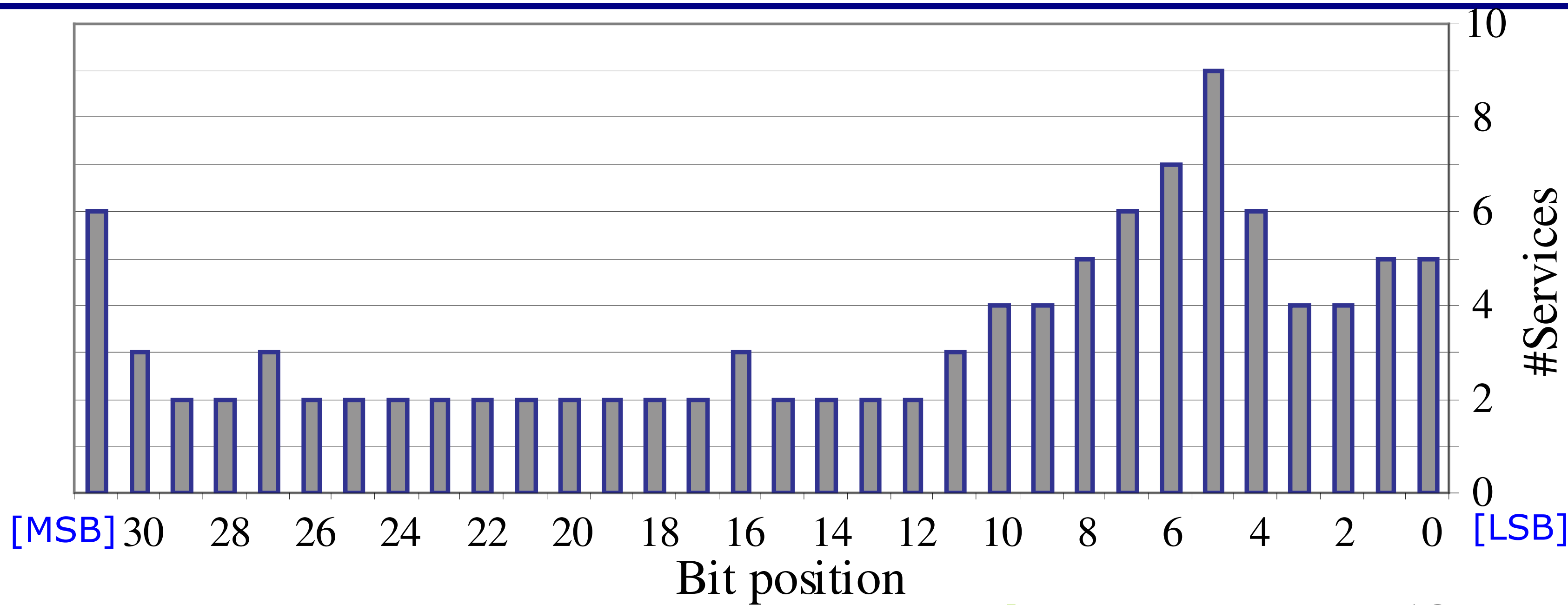
Which Driver Spreads Errors

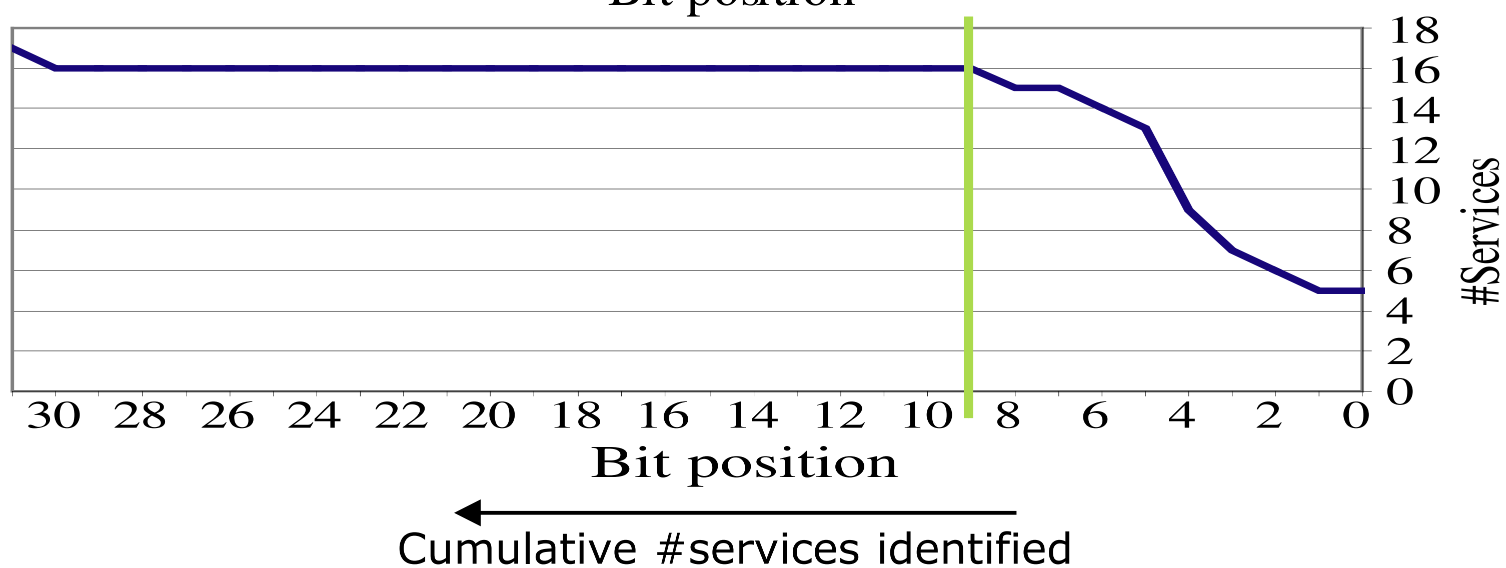


Experimentation Time

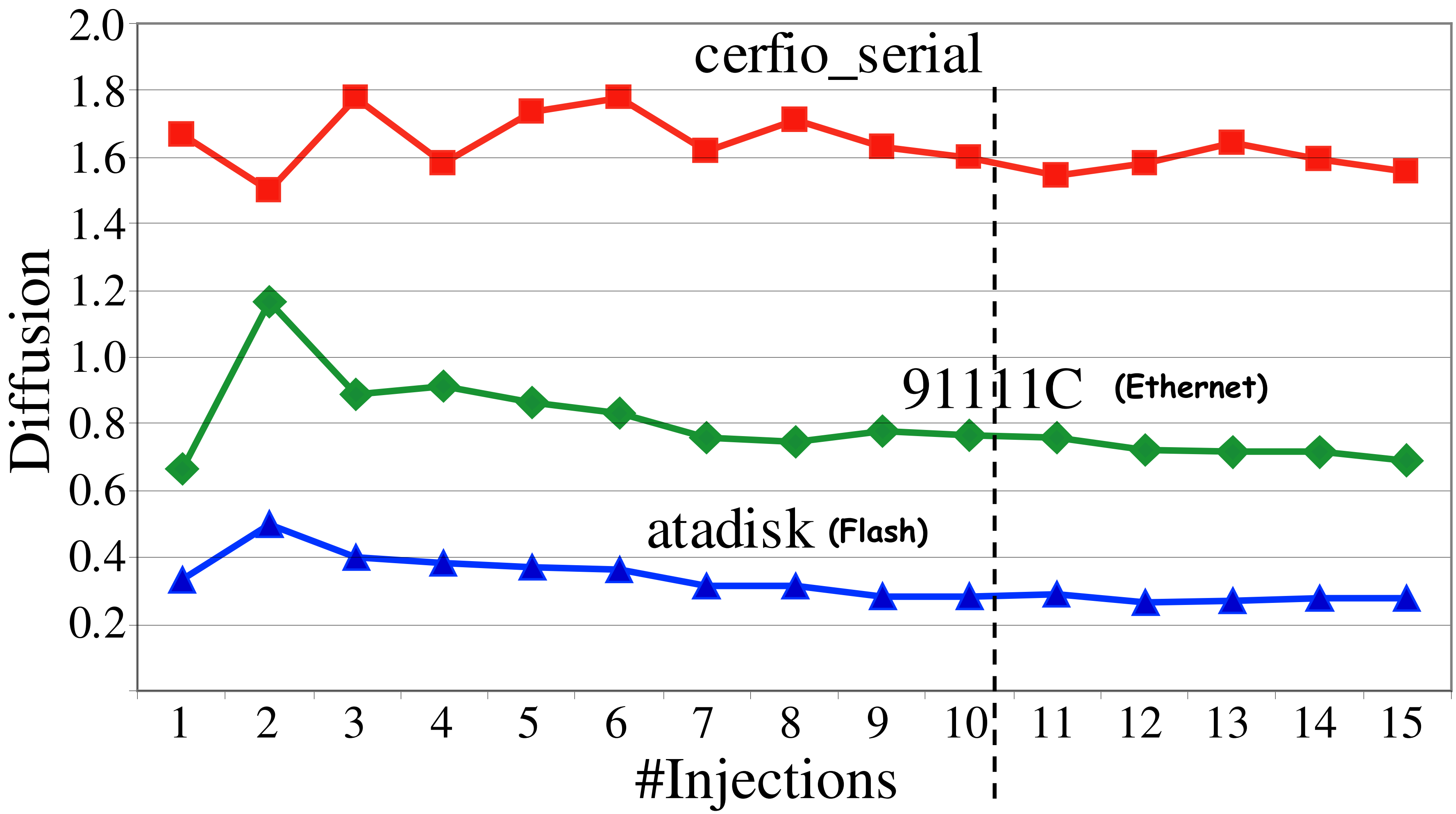
Driver	Injection Model	Exec. time	
		h	min
cerfio_serial	DT	5	15
	BF	**38**	**14**
	FZ	20	44
91C111 Ethernet	DT	1	56
	BF	**17**	**20**
	FZ	7	48
Atadisk Flash	DT	2	56
	BF	**20**	**51**
	FZ	11	55

1) BF Profile: Sensitivity (& Effort) w.r.t Bit Position?





2) Fuzzing Diffusion - Senstivity w.r.t # Injections?



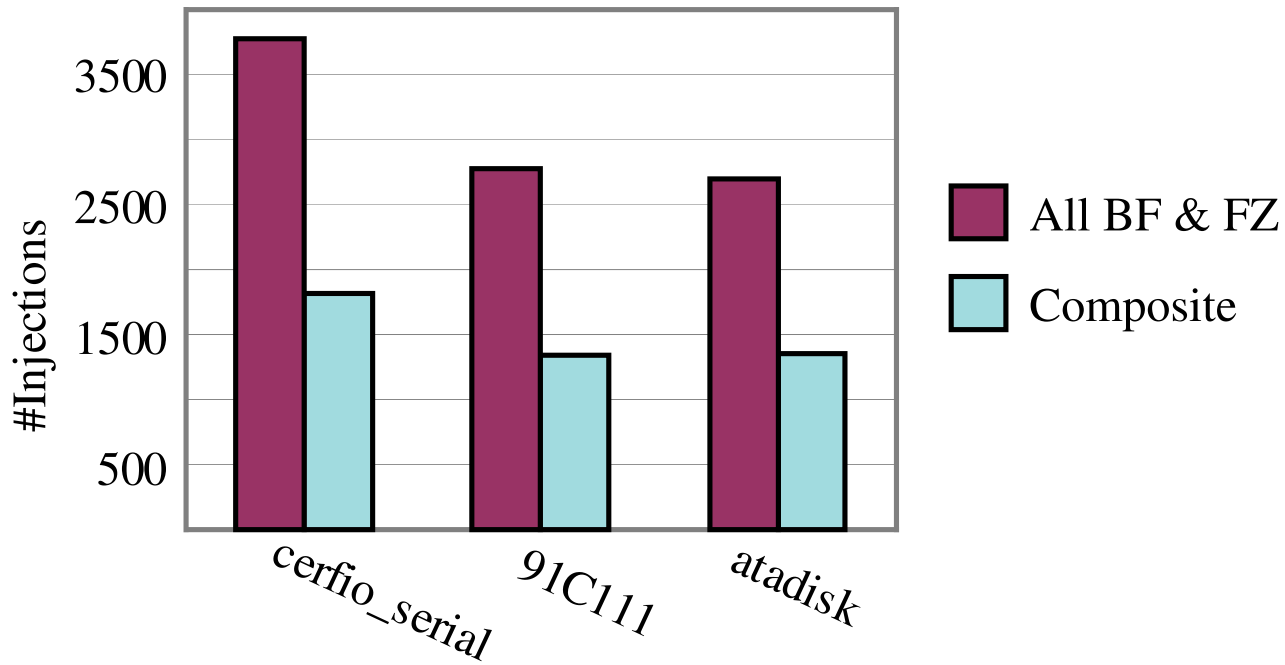
3) Sensitivity w.r.t Identifying Services (Class 3 + 2)

- Which OS services can cause Class 3+2 failures?
- Which fault model identifies most services (coverage)?
- Is some model consistently better/worse?
- Can we combine models?

Service	DT	BF	FZ
1	O	X	O
2	X	X	O
3		X	O
4		X	X
5			X
6	X	X	
7	X	X	O
8	X	X	
9	X	X	X
10	X	X	X
11	X	X	X
12	O	X	
13		X	
14	X	X	X
15		X	
16	X	X	X
17		X	
18		X	

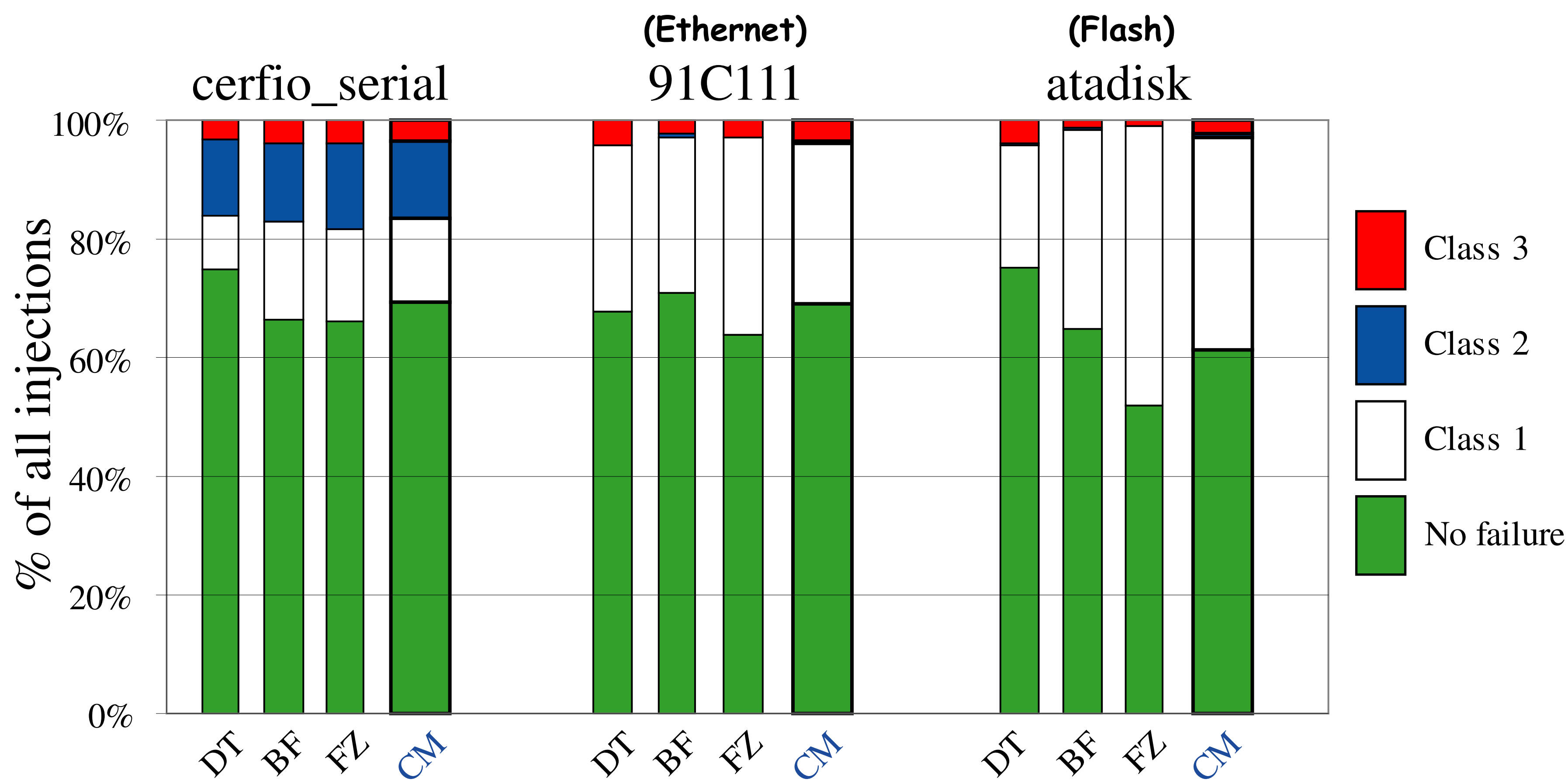
Composite Fault Model (CM)

- Let's take the best of BF and FZ models
  - Selective BF: Bits 0-9 and 31
  - Limited FZ: 10 injection cases

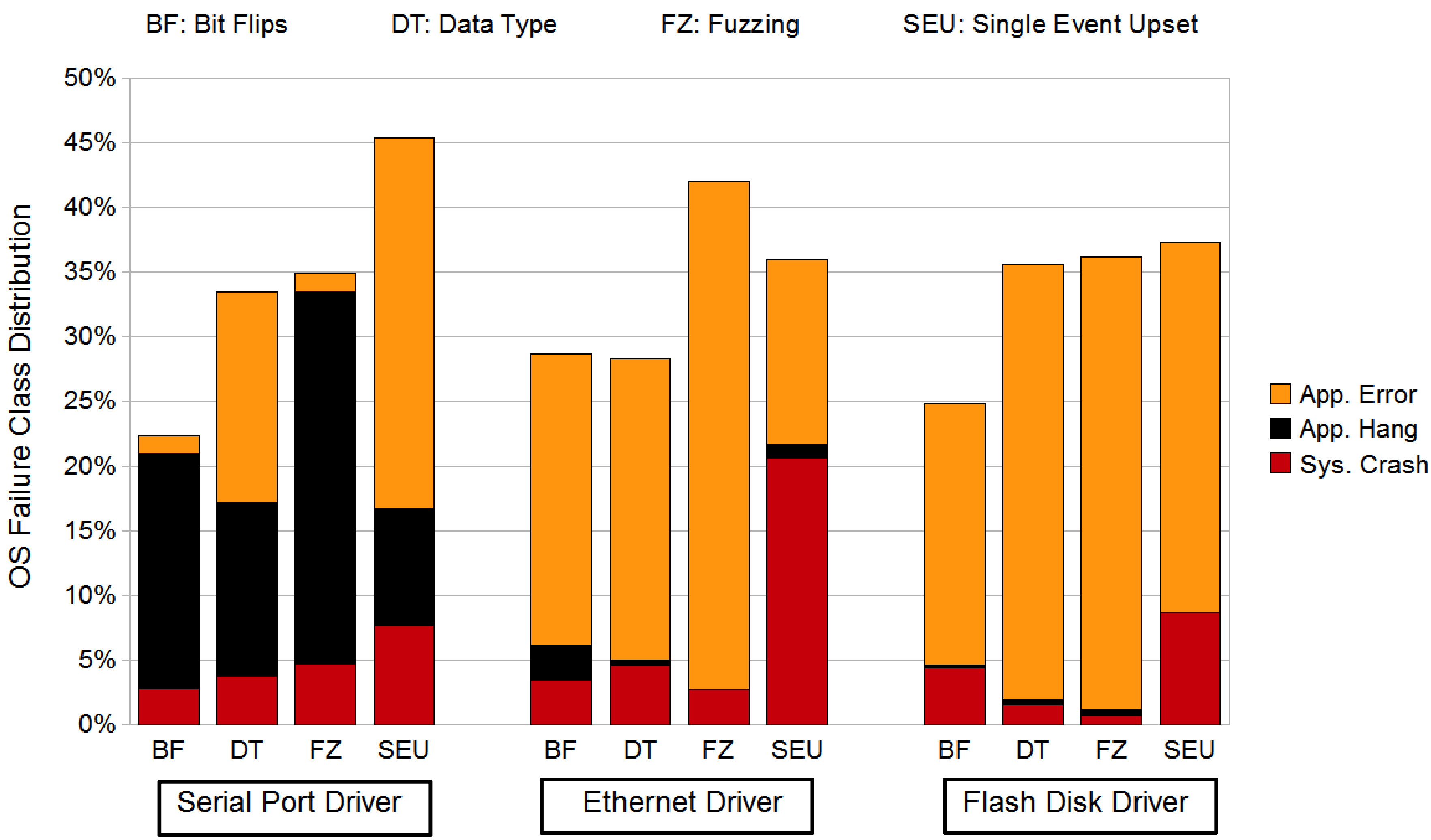


- ~50% fewer injections
- Identifies the same service set

Composite Fault Model – Results (Win CE.NET)



Injecting SEU's "into" Drivers



SEU: Control often not returned to calling kernel component – error prop. by direct kernel space mem. corruption with driver running in kernel mode – no interface errors

Comparing Across Established Models and CM

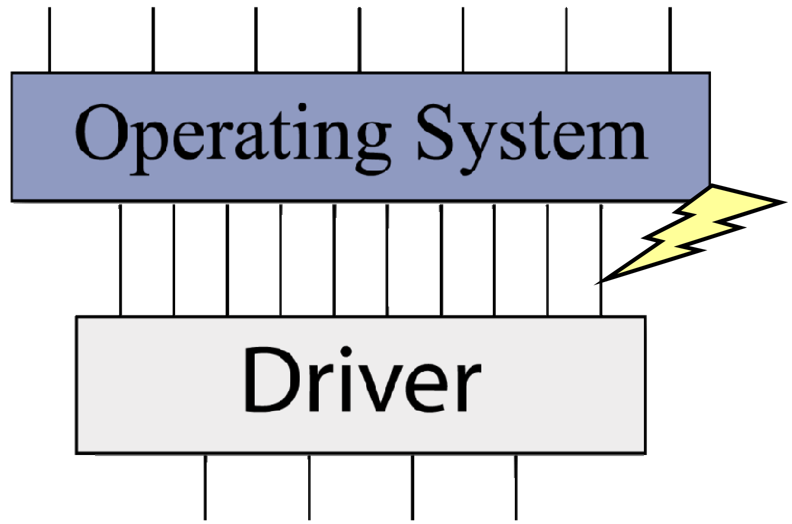
- Comparison metrics
  - Coverage: how many vulnerable services can a model identify?
  - Implementation complexity: input cases and output analysis
  - Injection efficiency: how good are models at provoking failures?
  - Execution time

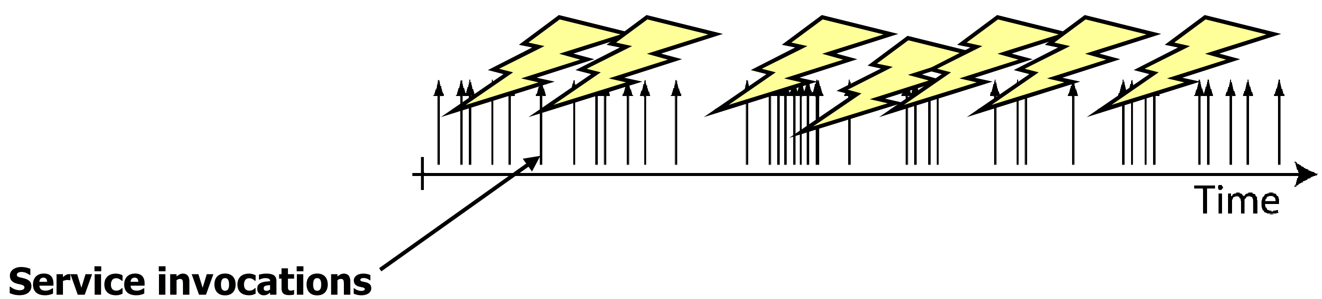
Model	Coverage	Implementation Complexity	Injection Efficiency	Execution Time
BF	★★★⋆	★★★★	★⋆⋆⋆	★★★⋆
DT	★★⋆⋆	★⋆⋆⋆	★★⋆⋆	★★★★
FZ	★★★★	★★⋆⋆	★★★⋆	★★⋆⋆
SEU	★⋆⋆⋆	★★⋆⋆	★★★⋆	★⋆⋆⋆
CM	****	***	***	**

Framework/ Authors	Fault Location	Fault Type	Fault Latency	Injection Trigger
MAFALDA [8]	CUE Server IUE	SEU MBU DT	Transient Permanent	1^{st} occ.
Albinet et al. [7]	IUE	DT	Transient	1^{st} occ.
Kalakech et al. [22]	IUE	SEU DT	Transient	1^{st} occ.
Xception [11]	CUE Server IUE	SEU MBU DT FZ	Transient Intermittent Permanent	1^{st} occ. n^{th} occ. Timer X-call
G-SWFIT [13]	CUE Server	Coding mistakes	Permanent	1^{st} occ.
Medonça & Neves [28]	CUE Server	Coding mistakes	Permanent	1^{st} occ.
Johansson [18]	IUE	SEU DT FZ	Transient Intermittent Permanent	1^{st} occ. n^{th} occ. Timer X-call Call Block

Where, What & When to Inject: The Timing Basis

- Target: interface OS-Driver
- Application → service(s) request
- Each service ='s many driver calls
- Each call is a potential injection
- Problem: too many calls
  - First-occurrence + timeouts
  - Sample (uniform?)



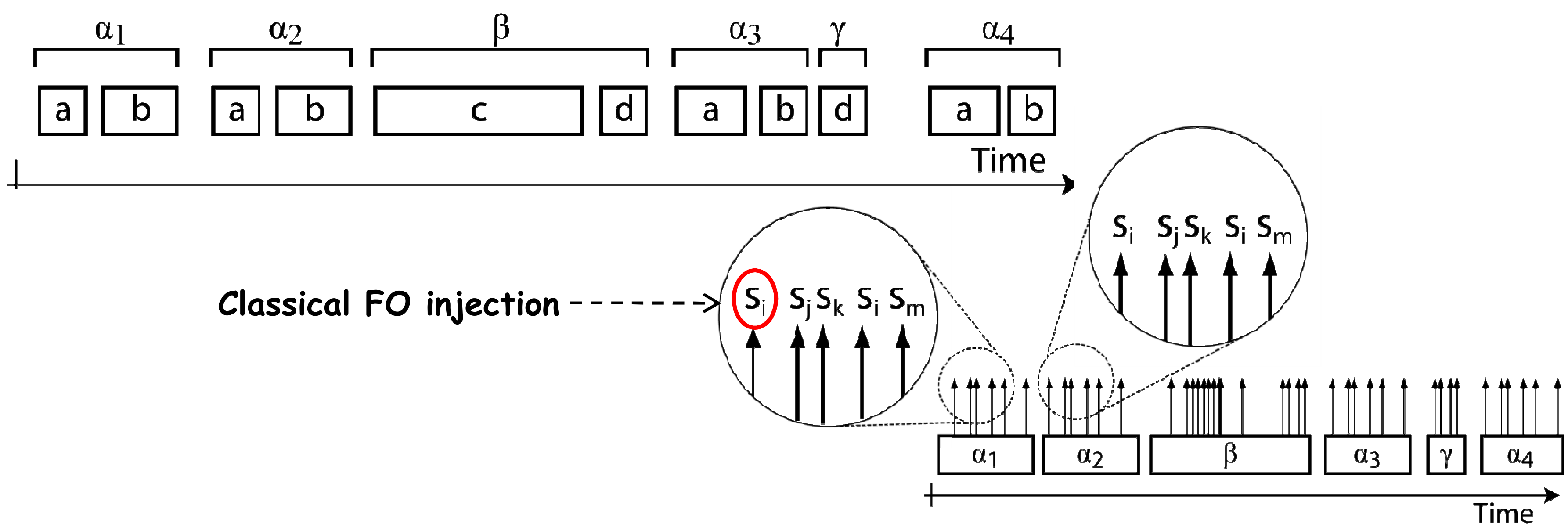


Calls String/Calls Blocks Basis

- Execute workload [Selected: Serial and Ethernet Drivers]
  - Record calls string specific to each driver "service req." a,b,c…

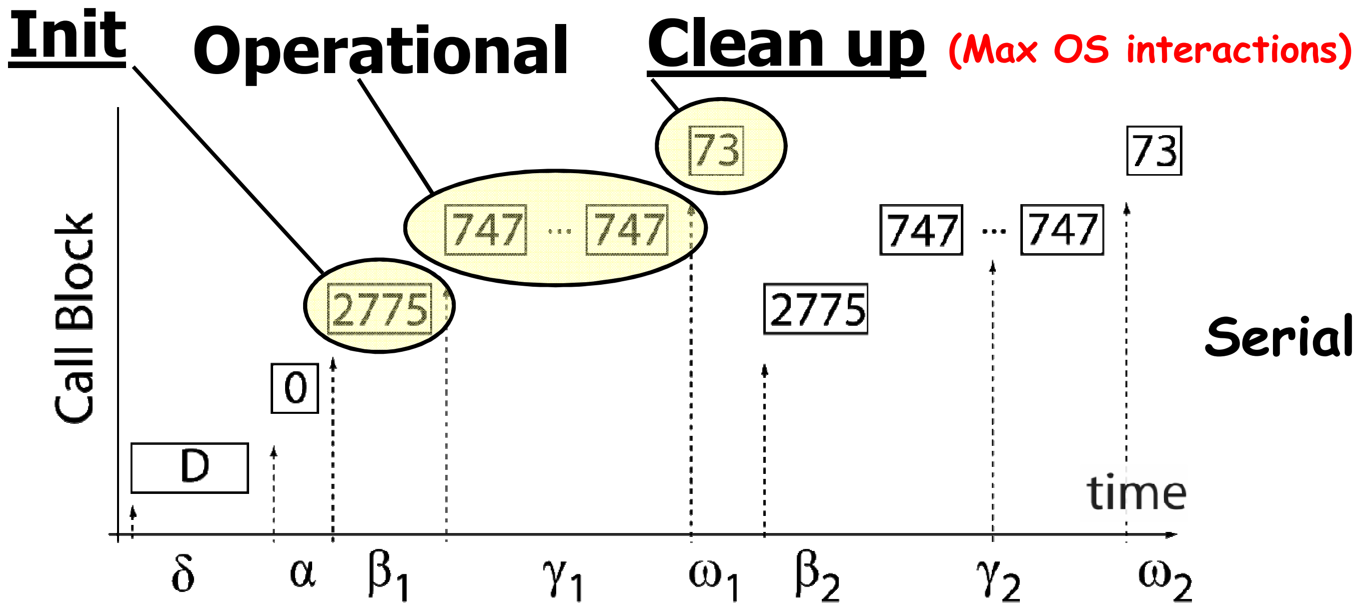
Services Call String: ababcdabdab

- Track repeating call blocks (subsequence of call strings)
  - Select service targets (1 per call block) ab ab c d ab d ab
- Identify call block triggers (ab) {2} c d (ab) d (ab); do injection

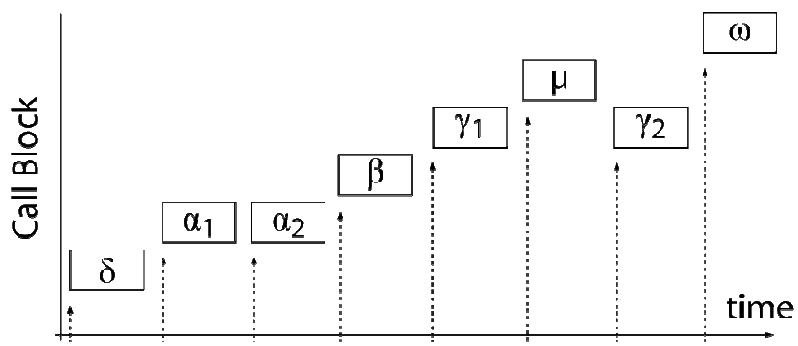


Call Blocks and Driver Phases (BF, Win CE.NET)

- Call string: `D02775(747){23}732775(747){23}23`



Serial Driver

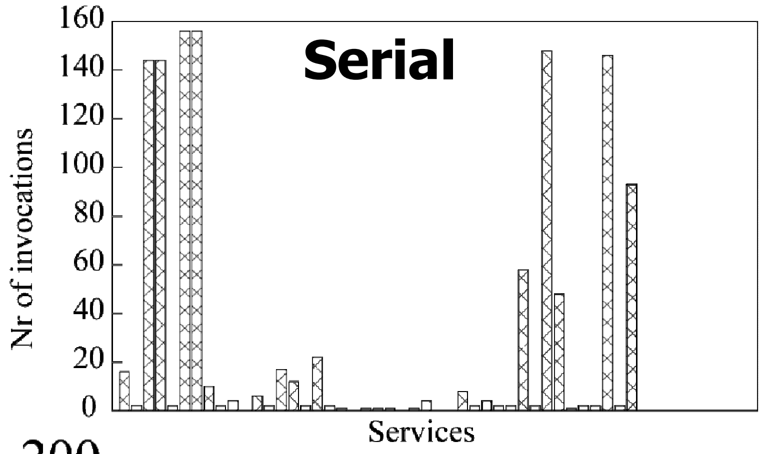


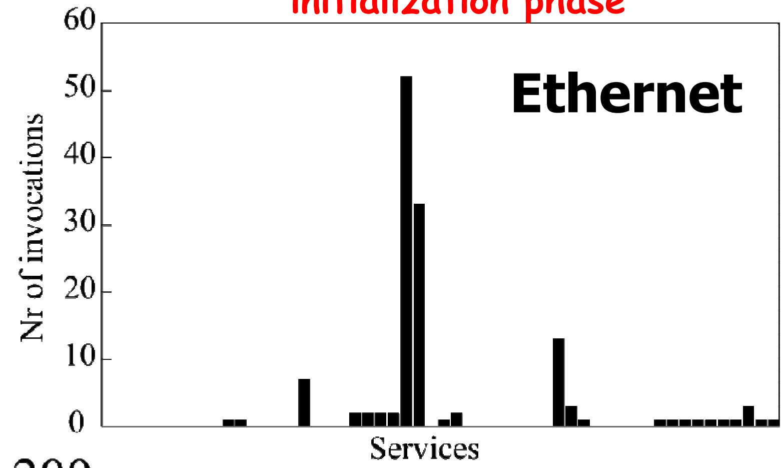
Ethernet

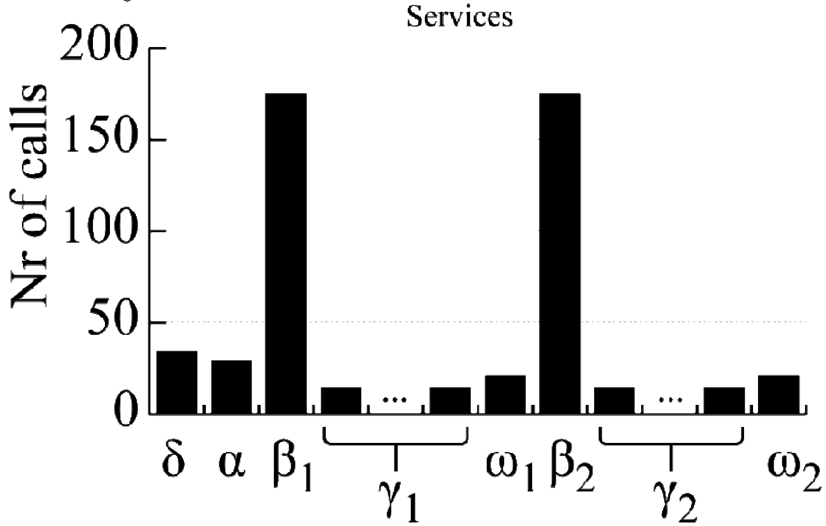
Driver Profiles

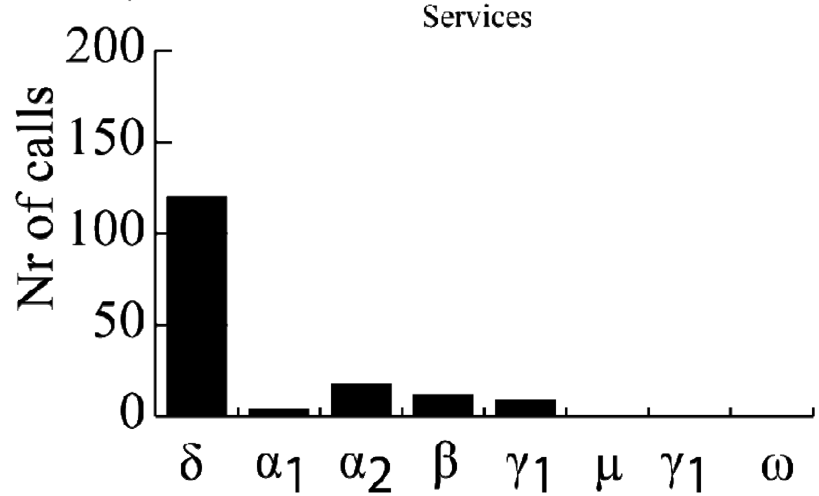
- Driver invocation patterns differ
- Impact of call block injection efficiency

Most OS interactions in initialization phase







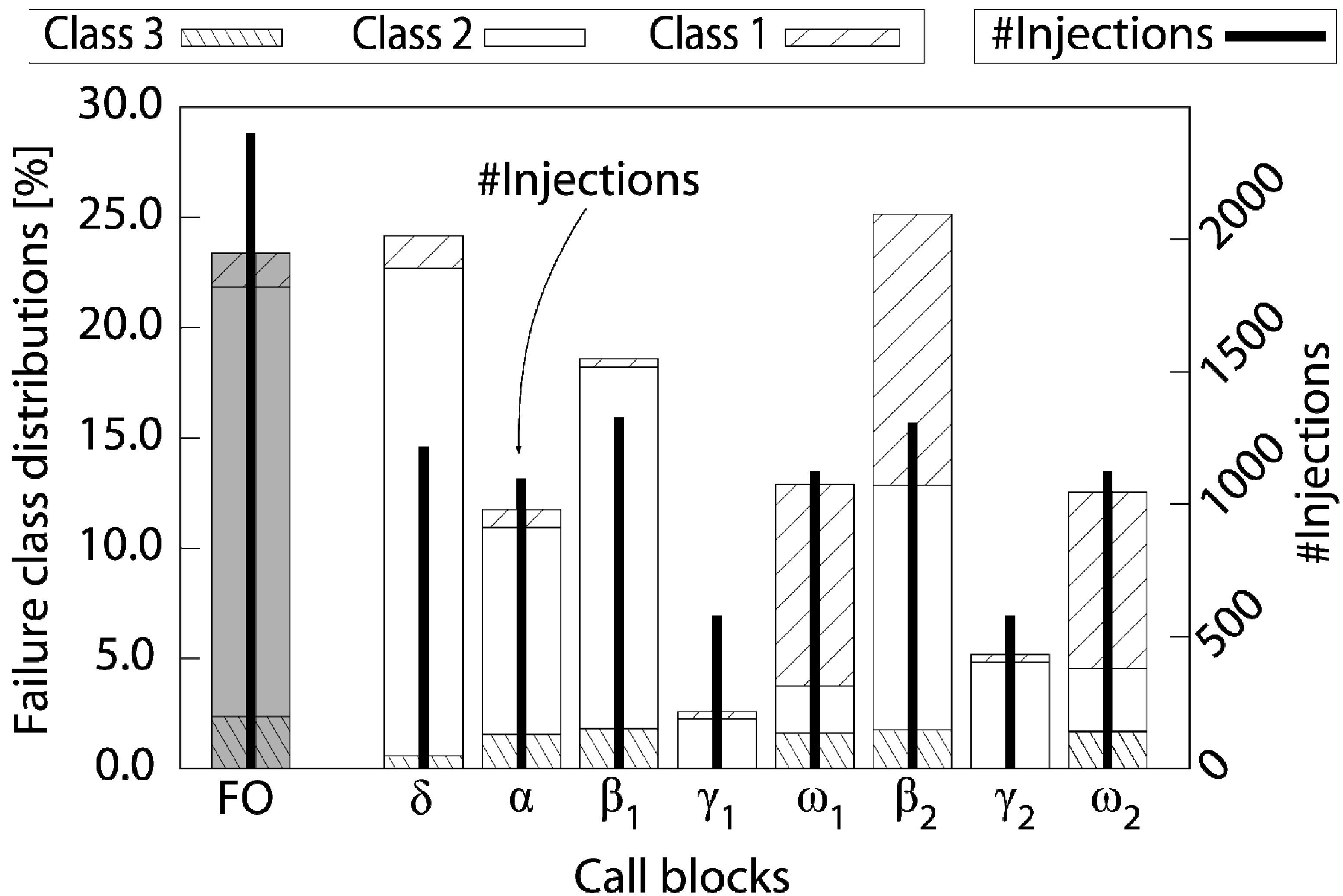


Serial Driver Service Identification

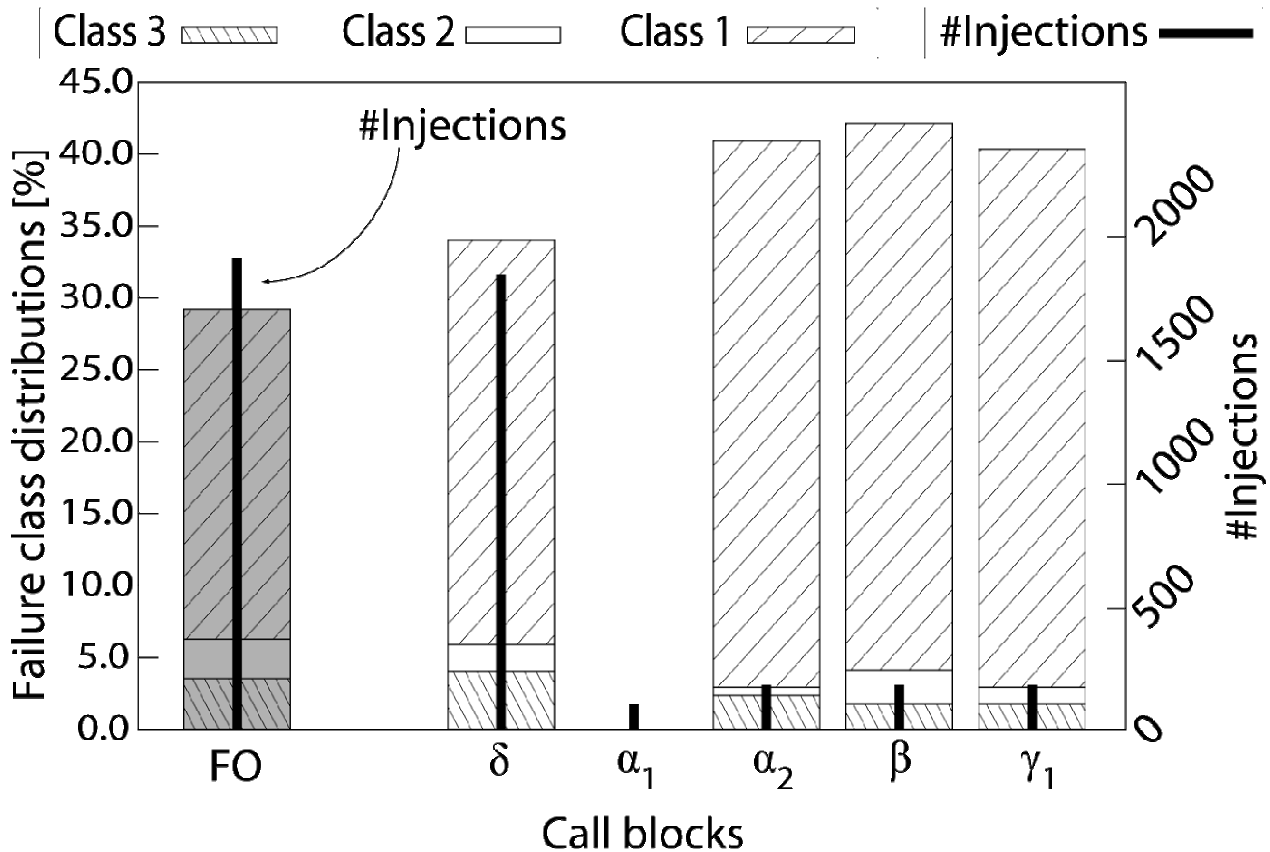
w. timeouts

	FO	δ	α	β_1	γ_1	ω_1	β_2	γ_2	ω_2
CreateThread	**x**			x			x		
DisableThreadLibraryCalls	**x**	x							
EventModify						x			x
FreeLibrary	**x**	x							
HalTranslateBusAddress			x						
InitializeCriticalSection		x							
InterlockedDecrement									x
LoadLibrary	**x**	x							
LocalAlloc	**x**	x							
memcpy	**x**			x			x		
memset	**x**			x			x		
SetProcPermissions	**x**			x			x		
TransBusAddrToStatic			x						

Serial Driver Results



Ethernet Driver Results



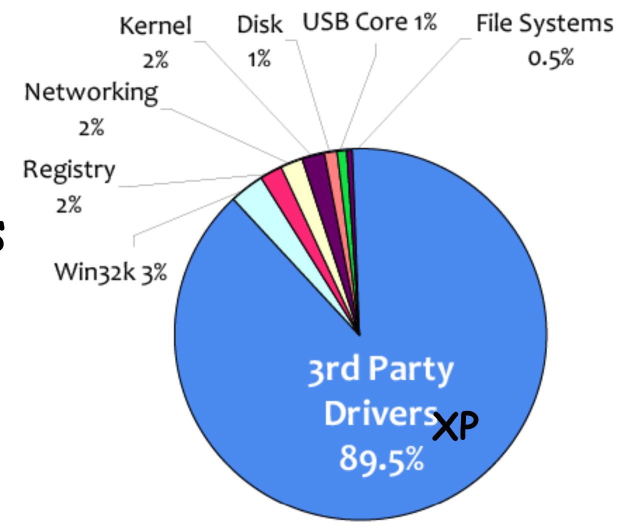
Trigger	Serial		Ethernet	
	#Injections	#C3	#Injections	#C3
First Occ.	2436	8	1820	12
Call Blocks	8408	13	2356	12

Timing Approach Summary

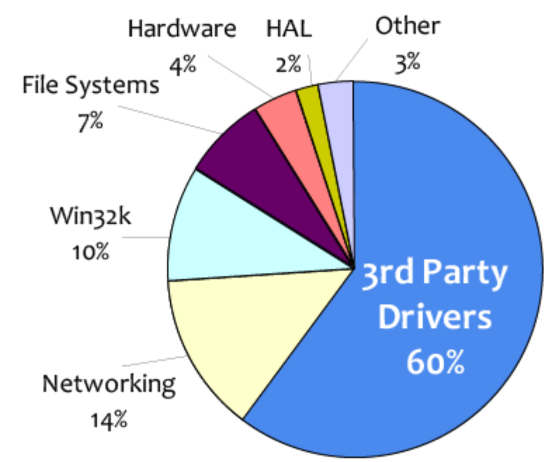
- Where, What & When?
- New call string/calls blocks timing model for interface FI
  - Often significant difference to FO
    - More injections (FO: 2436 vs. 8408)
    - BUT injections for specific/full coverage of services
  - Initialization and Clean up phases are most effective triggers based on higher OS interactions
  - Driver dependent with driver pre-profiling
  - Concurrent access (by svcs) to call strings: open issue

So what did the experimental approach buy us?

- Selective fault models
- Workload handling, dynamic app interactions
- Profiling for bits/data flows; hotspots & calls
- Better quantification basis
- Better granularity service identification as basis for design improvements



- Guidance to analysis!!!
  - Experimentation provides useful trends with caution not to over-generalize



Ongoing Issues

- What, When, Where to inject?
  - **Where**: to apply change (location, abstraction/system level)
  - **What**: to inject (what should be injected/corrupted?)
  - **Which**: trigger to use (event, instruction, timeout, exception?)
  - **When**: to inject (corresponding to type of fault)
  - **How**: often to inject (corresponding to type of fault)
  - …
- Correlations? Sequences? Timings?

- Reproducibility
- Generalization across driver classes
- Automation
- Does having source code actually help?