

Feature Consistency in Compile-Time-Configurable System Software

Facing the Linux 10000 Feature Problem

Reinhard Tartler, Daniel Lohmann, Julio Sincero,
Wolfgang Schröder-Preikschat

System Software Group

Friedrich-Alexander University
Erlangen-Nuremberg

April 11, 2011



supported by **DFG**

System Software is Configurable

- System Software is incredibly configurable



System Software is Configurable

- System Software is incredibly configurable
- Complexity increases considerably

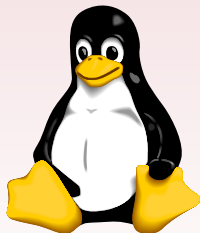


System Software is Configurable

- System Software is incredibly configurable
- Complexity increases considerably

→ Source of **bugs!**





Linux v2.6.35 contains:

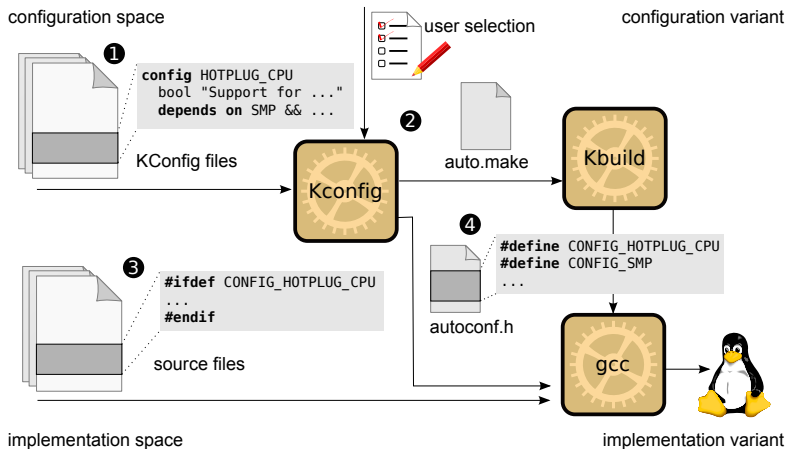
11.057 Features

27.166 Source files

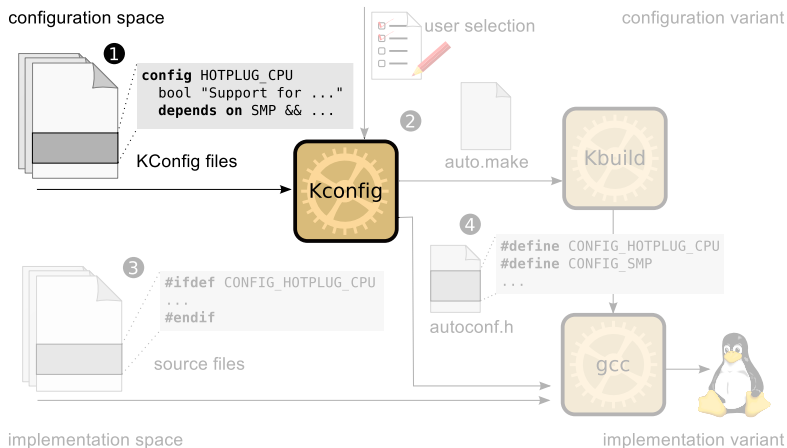
82.116 #ifdef blocks



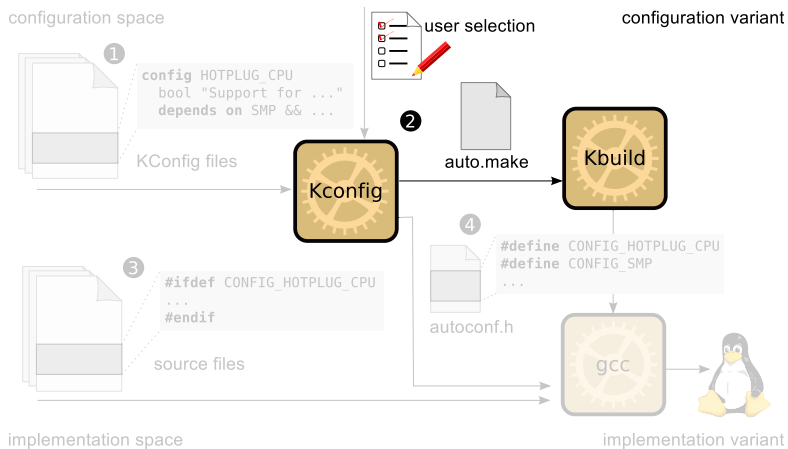
Variability Implementation in Linux



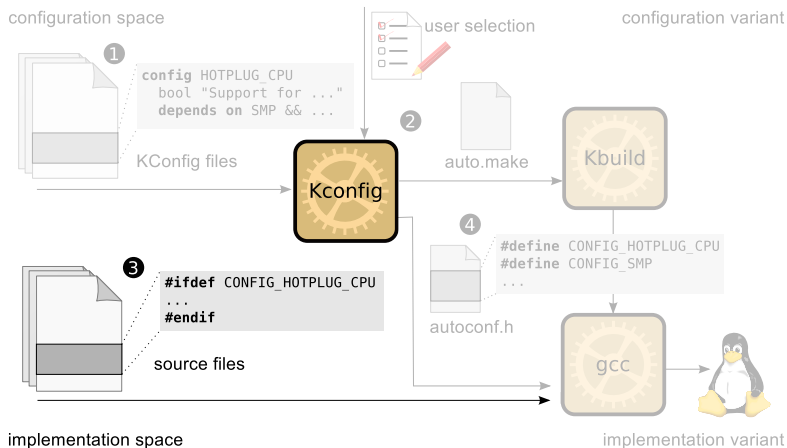
Variability Implementation in Linux



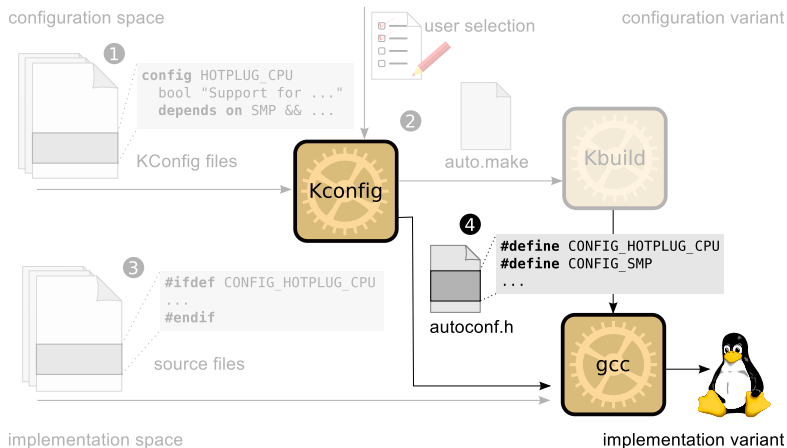
Variability Implementation in Linux



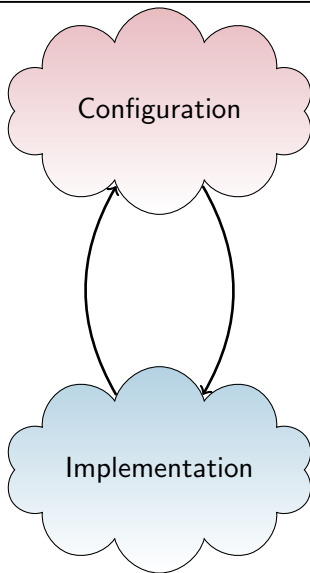
Variability Implementation in Linux



Variability Implementation in Linux



The Problem



Configuration

**Source of
Inconsistencies!**

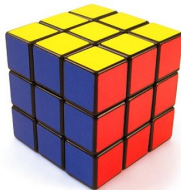
Implementation



- Bugs in **declaration** and **implementation**
- Excellent tool support for **static analysis**:
 - Dingo: Taming Device Drivers (EuroSys'09)
 - KLEE: Automatic generation of high-coverage tests (EuroSys'08)
 - RWset: Attacking path explosion (TACAS'08)
 - EXE: Automatically generating inputs of death (CCS'06)
 - ...



- Bugs in **declaration** and **implementation**
- Excellent tool support for **static analysis**:
 - Dingo: Taming Device Drivers (EuroSys'09)
 - KLEE: Automatic generation of high-coverage tests (EuroSys'08)
 - RWset: Attacking path explosion (TACAS'08)
 - EXE: Automatically generating inputs of death (CCS'06)
 - ...
- Each of them is configuration **agnostic**:

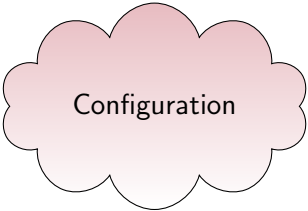


Outline

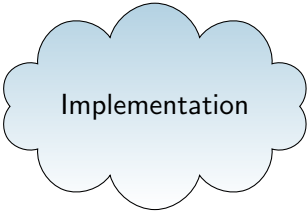
1. Introduction
2. Analysis
3. Approach and Implementation
4. Results
5. Future Work and Conclusions



Problem Analysis



Configuration



Implementation



Problem Analysis

Configuration

symbols

constraints



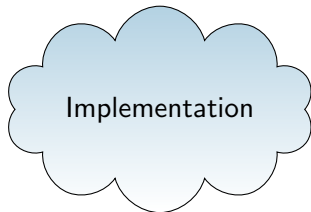
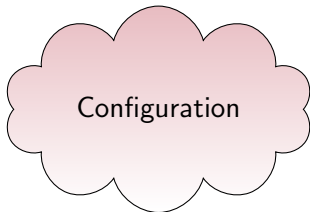
Implementation

symbols

constraints



Problem Analysis



`config` `HOTPLUG_CPU`

symbols



constraints

`depends on` `SMP` `&&` `HOTPLUG`

symbols



constraints



Problem Analysis

Configuration

Implementation

```
config HOTPLUG_CPU
```

symbols



constraints

```
depends on SMP && HOTPLUG
```

```
#ifdef CONFIG_CPU_HOTPLUG
```

symbols

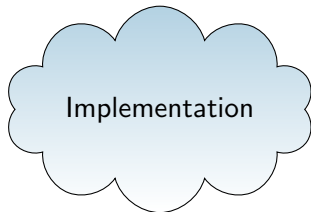
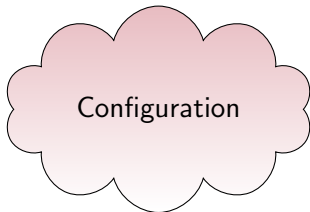


constraints

```
#ifdef CONFIG_CPU_HOTPLUG  
#else  
#endif
```



Problem Analysis



```
config HOTPLUG_CPU
```

symbols

```
#ifdef CONFIG_CPU_HOTPLUG
```

symbols



constraints



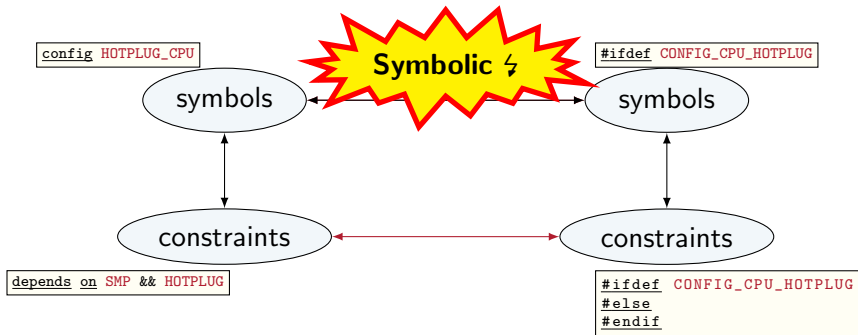
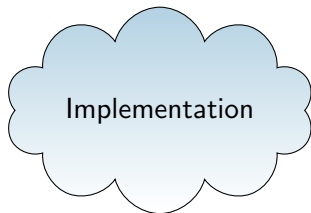
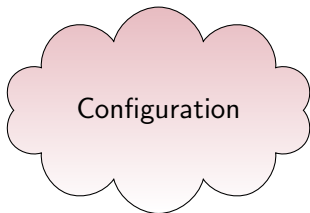
constraints

```
depends on SMP && HOTPLUG
```

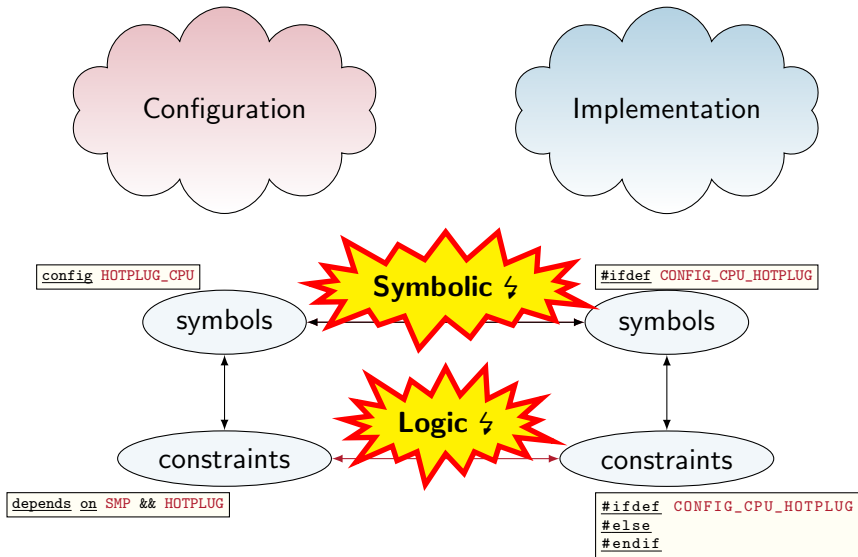
```
#ifdef CONFIG_CPU_HOTPLUG  
#else  
#endif
```



Problem Analysis



Problem Analysis



Symbolic Inconsistency

```
config HOTPLUG_CPU
    bool "Support for hot-pluggable CPUs"
    depends on SMP && HOTPLUG
    ---help---
```

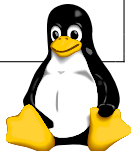


Symbolic Inconsistency

```
config HOTPLUG_CPU
    bool "Support for hot-pluggable CPUs"
    depends on SMP && HOTPLUG
    ---help---
```

```
static int
hotplug_cfd(struct notifier_block *nfb, unsigned long action, void *hcpu)
{
    // [...]
    switch (action) {
        case CPU_UP_PREPARE:
        case CPU_UP_PREPARE_FROZEN:
            // [...]
#ifdef CONFIG_CPU_HOTPLUG
        case CPU_UP_CANCELED:
        case CPU_UP_CANCELED_FROZEN:

        case CPU_DEAD:
        case CPU_DEAD_FROZEN:
            free_cpumask_var(cfd->cpumask);
            break;
#endif
    };
    return NOTIFY_OK;
```



Symbolic Inconsistency

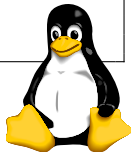
```
config HOTPLUG_CPU
    bool "Support for hot-pluggable CPUs"
    depends on SMP && HOTPLUG
    ---help---
```

```
static int
hotplug_cfd(struct notifier_block *nb, unsigned long action, void *hcpu)
{
    // [...]
    switch (action) {
        case CPU_UP_PREPARE:
        case CPU_UP_PREPARE_FROZEN:
            // [...]
#ifdef CONFIG_CPU_HOTPLUG
        case CPU_UP_CANCELED:
        case CPU_UP_CANCELED_FROZEN:

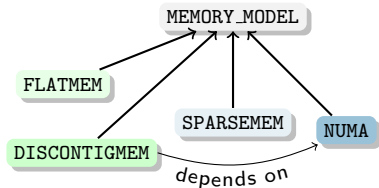
        case CPU_DEAD:
        case CPU_DEAD_FROZEN:
            free_cpumask_var(cfd->cpumask);
            break;
#endif
    };
    return NOTIFY_OK;
```

Symbolic ↴

■ Result: Fix for a **critical bug**



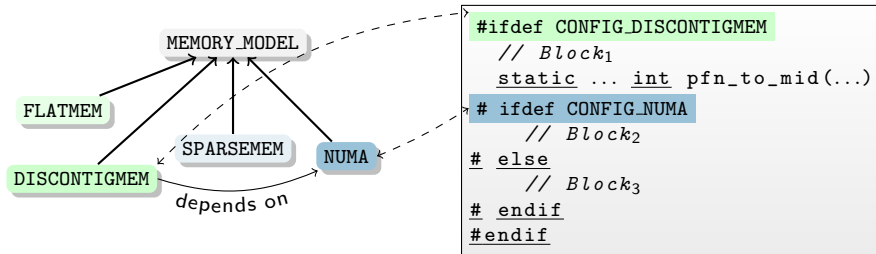
Logic Inconsistencies



```
#ifdef CONFIG_DISCONTIGMEM
    // Block1
    static ... int pfn_to_mid(...)
#endif
#ifdef CONFIG_NUMA
    // Block2
#else
    // Block3
#endif
#endif
```



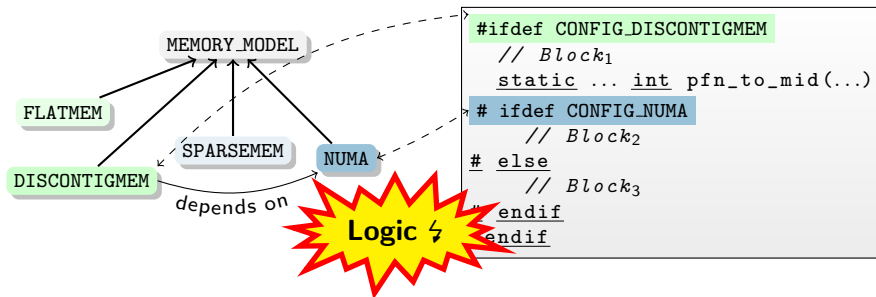
Logic Inconsistencies



- Feature DISCONTIGMEM requires NUMA
- Inner block is not configuration dependent anymore



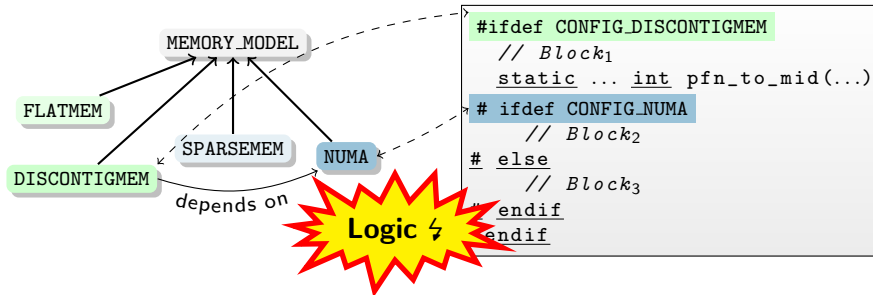
Logic Inconsistencies



- Feature DISCONTIGMEM requires NUMA
- Inner block is not configuration dependent anymore



Logic Inconsistencies



- Feature DISCONTIGMEM requires NUMA
- Inner block is not configuration dependent anymore
- Result: **code cleanup**

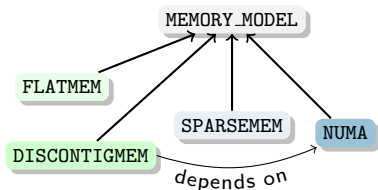


Outline

1. Introduction
2. Analysis
3. Approach and Implementation
4. Results
5. Future Work and Conclusions



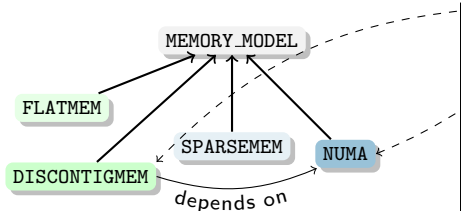
General Approach



```
#ifdef CONFIG_DISCONTIGMEM
    // Block1
    static ... int pfn_to_mid(...)
#endif
#ifdef CONFIG_NUMA
    // Block2
#else
    // Block3
#endif
#endif
```



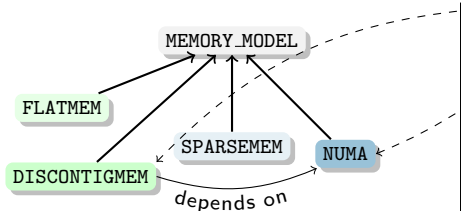
General Approach



```
#ifdef CONFIG_DISCONTIGMEM
    // Block1
    static ... int pfn_to_mid(...)
#endif
#ifdef CONFIG_NUMA
    // Block2
#else
    // Block3
#endif
#endif
```



General Approach

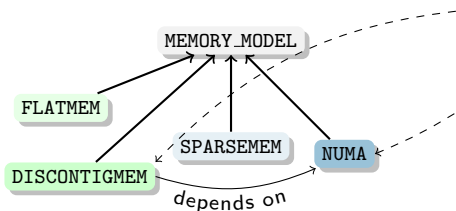


```
#ifdef CONFIG_DISCONTIGMEM
// Block1
static ... int pfn_to_mid(...)
#endif
#ifdef CONFIG_NUMA
// Block2
#else
// Block3
#endif
#endif
```

extract

$$\begin{aligned} C = & (\text{FLATMEM} \rightarrow \text{MEMORY_MODEL}) \\ & \wedge (\text{DISCONTIGMEM} \rightarrow \text{MEMORY_MODEL}) \\ & \wedge (\text{SPARSEMEM} \rightarrow \text{MEMORY_MODEL}) \\ & \wedge (\text{NUMA} \rightarrow \text{MEMORY_MODEL}) \\ & \wedge (\text{DISCONTIGMEM} \rightarrow \text{NUMA}) \end{aligned}$$


General Approach

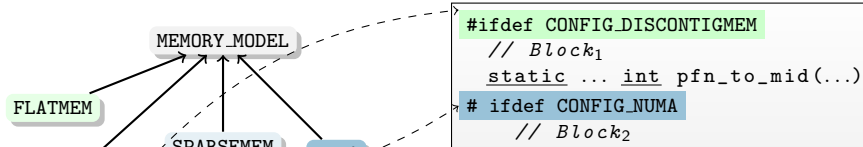


```
#ifdef CONFIG_DISCONTIGMEM
// Block1
static ... int pfn_to_mid(...)
#endif
#ifdef CONFIG_NUMA
// Block2
#else
// Block3
#endif
#endif
```


$$C = (\text{FLATMEM} \rightarrow \text{MEMORY_MODEL})$$
$$\wedge (\text{DISCONTIGMEM} \rightarrow \text{MEMORY_MODEL})$$
$$\wedge (\text{SPARSEMEM} \rightarrow \text{MEMORY_MODEL})$$
$$\wedge (\text{NUMA} \rightarrow \text{MEMORY_MODEL})$$
$$\wedge (\text{DISCONTIGMEM} \rightarrow \text{NUMA})$$

$$I = (\text{Block}_1 \leftrightarrow \text{DISCONTIGMEM})$$
$$\wedge (\text{Block}_2 \leftrightarrow \text{Block}_1 \wedge (\text{NUMA}))$$
$$\wedge (\text{Block}_3 \leftrightarrow \text{Block}_1 \wedge \neg \text{Block}_2)$$


General Approach



Crosscheck both formulas with a SAT solver:

$$\mathbf{dead?} = \text{sat}(C \wedge \mathcal{I} \wedge \text{Block}_N)$$

$$\mathbf{undead?} = \text{sat}(C \wedge \mathcal{I} \wedge \neg \text{Block}_N \wedge \text{parent}(\text{Block}_N))$$

$$\wedge (\text{DISCONTIGMEM} \rightarrow \text{MEMORY_MODEL})$$

$$\wedge (\text{SPARSEMEM} \rightarrow \text{MEMORY_MODEL})$$

$$\wedge (\text{NUMA} \rightarrow \text{MEMORY_MODEL})$$

$$\wedge (\text{DISCONTIGMEM} \rightarrow \text{NUMA})$$

$$\wedge (\text{Block}_2 \leftrightarrow \text{Block}_1 \wedge (\text{NUMA}))$$

$$\wedge (\text{Block}_3 \leftrightarrow \text{Block}_1 \wedge \neg \text{Block}_2)$$



- Accuracy
 - Conceptually **no false positives**
 - **Exact** identification of variation points



■ Accuracy

- Conceptually **no false positives**
- **Exact** identification of variation points

■ Coverage

- Extract configuration model for all **22 architectures**
- Defect \rightsquigarrow detected on **each** architecture



■ Accuracy

- Conceptually **no false positives**
- **Exact** identification of variation points

■ Coverage

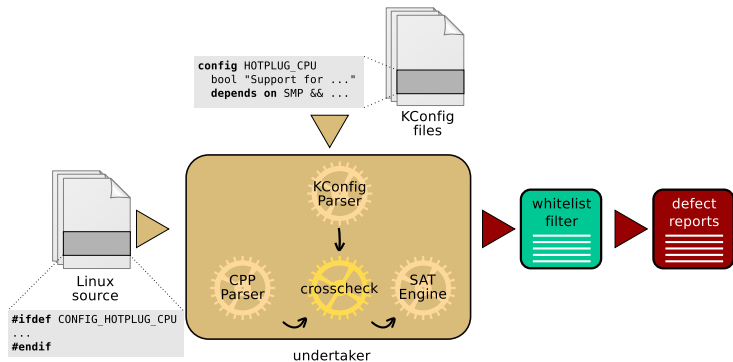
- Extract configuration model for all **22 architectures**
- Defect \rightsquigarrow detected on **each** architecture

■ Performance

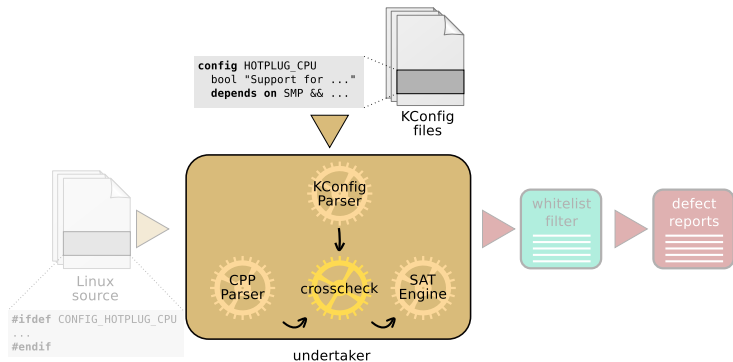
- Easy and fast to use during **incremental builds**
- Possible by **problem slicing**
- **Complete run** on Linux in **15** minutes



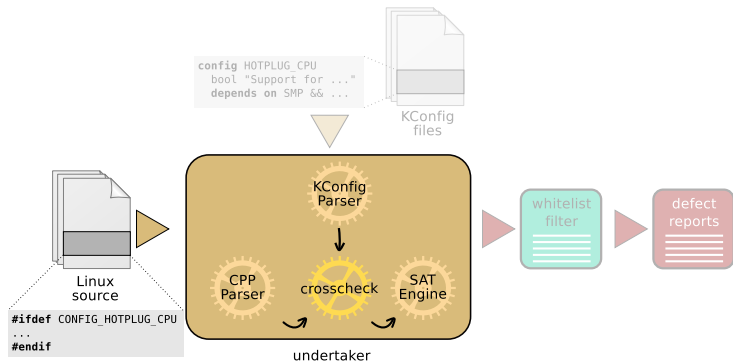
Implementation for Linux



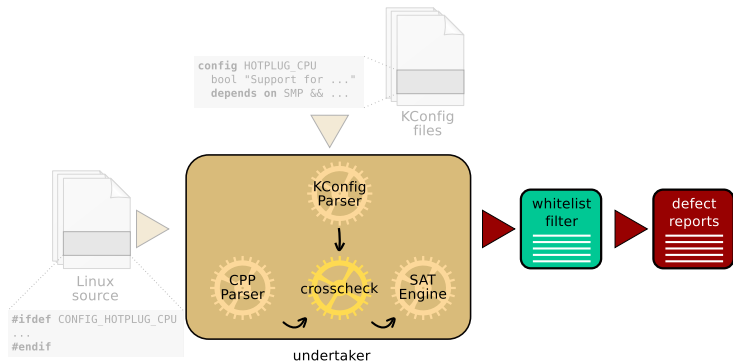
Implementation for Linux



Implementation for Linux



Implementation for Linux



Outline

1. Introduction
2. Analysis
3. Approach and Implementation
4. Results
5. Future Work and Conclusions



Results

subsystem	#ifdefs	logic	symbolic	total
arch/	33757	345	581	926
drivers/	32695	88	648	736
fs/	3000	4	13	17
include/	7241	6	11	17
kernel/	1412	7	2	9
mm/	555	0	1	1
net/	2731	1	49	50
sound/	3246	5	10	15
virt/	53	0	0	0
other subsystems	601	4	1	5
Σ	85291	460	1316	1776
fix proposed		150 (1)	214 (22)	364 (23)
confirmed defect		38 (1)	116 (20)	154 (21)
confirmed rule-violation		88 (0)	21 (2)	109 (2)
pending		24 (0)	77 (0)	101 (0)



Results

We have found **1776** configurability issues

Submitted **123** patches for **364** defects

20 are confirmed **new bugs** (affecting binary code)

Cleaned up **5129** lines of cruft code

pending

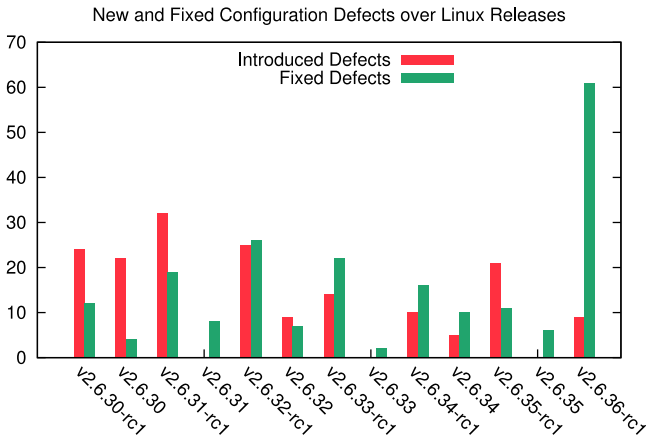
24 (0)

77 (0)

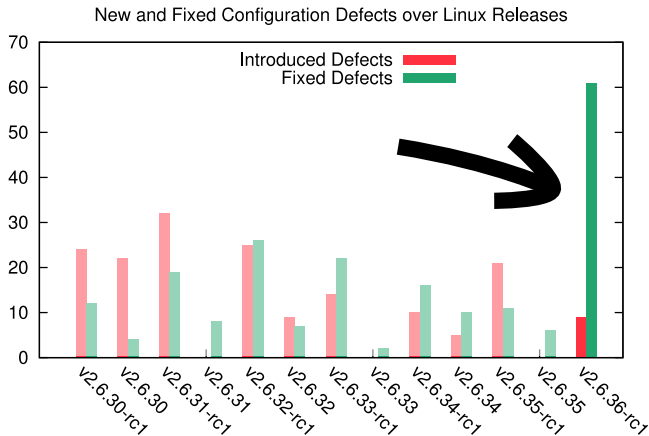
101 (0)



Patch Submission during the merge window of version 2.6.36:



Patch Submission during the merge window of version 2.6.36:



Outline

1. Introduction
2. Analysis
3. Approach and Implementation
4. Results
5. Future Work and Conclusions



- Data has to be seen as **lower bound**:
 - More precise configuration space extraction is possible
 - #define support
 - Improved implementation find > 4000 defects



- Data has to be seen as **lower bound**:
 - More precise configuration space extraction is possible
 - #define support
 - Improved implementation find > 4000 defects

- Focus on pure variability defects



- Data has to be seen as **lower bound**:
 - More precise configuration space extraction is possible
 - #define support
 - Improved implementation find > 4000 defects
- Focus on pure variability defects
- Integration of configuration agnostic tools for static analysis



Conclusions

- Configurability has to be seen as a significant cause of software defects in its own respect
- **Configuration** and **implementation** need to be kept consistent
- Our approach finds and fixes real problems!
 - Over **100 patches** submitted and about **50** accepted!
 - Excellent feedback from kernel developers



Conclusions

- Configurability has to be seen as a significant cause of software defects in its own respect
- **Configuration** and **implementation** need to be kept consistent
- Our approach finds and fixes real problems!
 - Over **100 patches** submitted and about **50** accepted!
 - Excellent feedback from kernel developers

<http://vamos.informatik.uni-erlangen.de/trac/undertaker>

