Persistency semantics of the ext4 filesystem

Ilya Kaysin

Research supervisor:
PhD Anton Podkopaev,
HSE, JetBrains Research

Reviewer:
PhD Christopher Pulte,
University of Cambridge

HIGHER · SCHOOL · OF · ECONOMICS ·

NATIONAL RESEARCH
UNIVERSITY

Memory

```
// a = b = 0
a = 1
b = 1
// a = b = 1
```

Memory

```
// a = b = 0
a = 1
b = 1
// a = b = 1
```

crash

```
// a = b = 0
```

Memory

```
// a = b = 0
a = 1
b = 1
// a = b = 1
```

crash

```
// a = b = 0
```

HDD

```
// a.txt = b.txt = “0”
a.txt <- “1”
b.txt <- “1”
// a.txt = b.txt = “1”
```

```
// a.txt = b.txt = “1”
```

Memory

```
// a = b = 0
a = 1
b = 1
// a = b = 1
```

crash

```
// a = b = 0
```

HDD

```
// a.txt = b.txt = “0”
a.txt <- “1”
b.txt <- “1”
// a.txt = b.txt = “1”
```

```
// a.txt = b.txt = “1”
//   “0”           “0”
//   “1”           “0”
```

Memory

```
// a = b = 0
a = 1
b = 1
// a = b = 1
```

crash

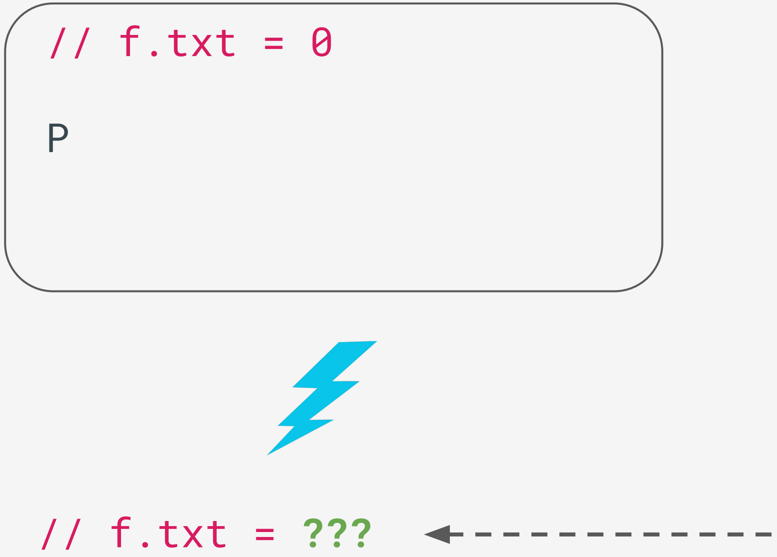
```
// a = b = 0
```

HDD

```
// a.txt = b.txt = “0”
a.txt <- “1”
b.txt <- “1”
// a.txt = b.txt = “1”
```

```
// a.txt = b.txt = “1”
//   “0”          “0”
//   “1”          “0”
//   “0”          “1”
```

Concurrent environment



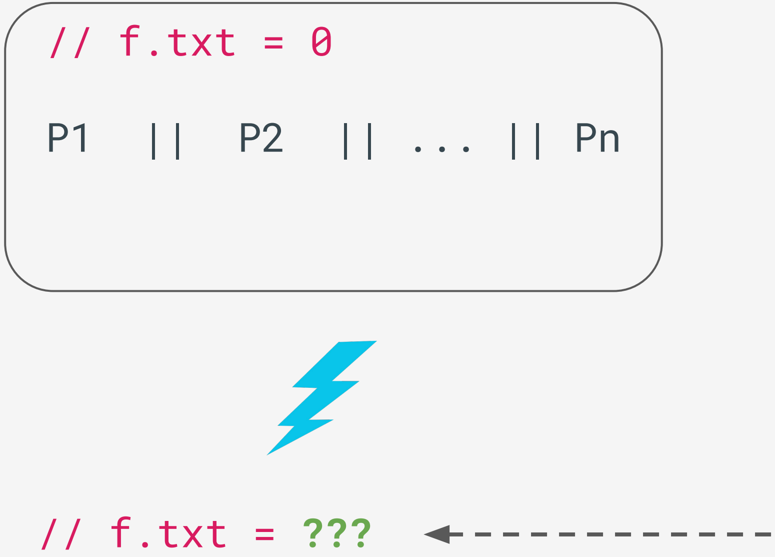
```
// f.txt = 0

P
```

```
// f.txt = ???
```

Persistency: what is observable after a system crash and restart

Concurrent environment



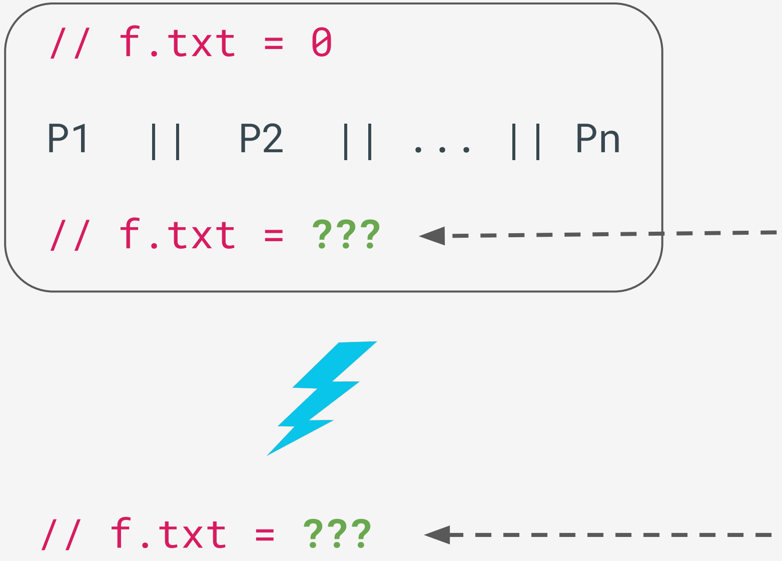
```
// f.txt = 0

P1  ||  P2  || ... || Pn
```

```
// f.txt = ???
```

Persistency: what is observable after a system crash and restart

Concurrent environment



```
// f.txt = 0

P1  ||  P2  || ... || Pn

// f.txt = ???
```

```
// f.txt = ???
```

Consistency: what is observable by concurrent threads

Persistency: what is observable after a system crash and restart

Motivation

- Filesystems are complicated
- Ext4 – the most popular modern filesystem in Linux. Optimisations => obscure behavior => bugs
- For formal verification, **formal semantics** is required

Motivation

- Filesystems are complicated
- Ext4 – the most popular modern filesystem in Linux.

  Optimisations => obscure behavior => bugs
- For formal verification, **formal semantics** is required

mathematical model defining what can be observed (1) by other threads, (2) upon a crash

Related work

Weak persistency models

- Persistency semantics of the Intel-x86 architecture [Raad et al. POPL 2019]
- Weak Persistency Semantics from the Ground Up [Raad et al. OOPSLA 2019]

Filesystem models

- Development of a Verified Flash File System [Schellhorn et al. ABZ 2014]
- Specifying and Checking File System Crash-Consistency Models [Bornholt et al. ASPLOS 2016]

Related work

Weak persistency models

- Persistency semantics of the Intel-x86 architecture [Raad et al. POPL 2019]
- Weak Persistency Semantics from the Ground Up [Raad et al. OOPSLA 2019]

Filesystem models

- Development of a Verified Flash File System [Schellhorn et al. ABZ 2014]
- Specifying and Checking File System Crash-Consistency Models [Bornholt et al. ASPLOS 2016]

But no model encompasses **both:**

filesystem consistency and persistency

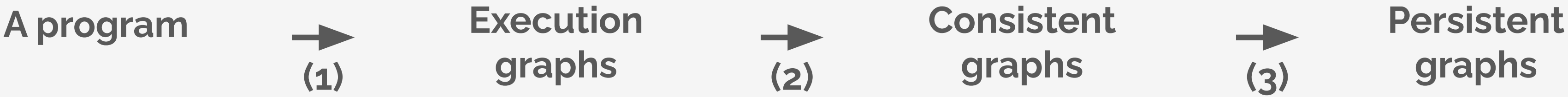
Goal: developing a formal model of ext4 compatible with formal verification

Objectives:

1. introduce a **general** filesystem **framework**
2. **specialize** the framework to build the **ext4 semantics**
3. adapt the ext4 semantics for **formal verification**

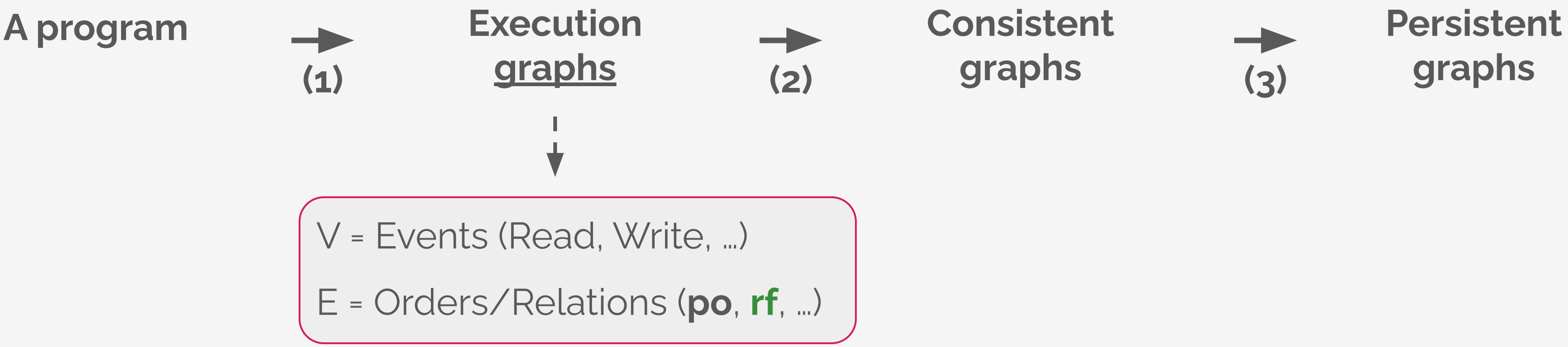
Objective 1

General framework scheme:



A program → (1) Execution graphs → (2) Consistent graphs → (3) Persistent graphs

General framework scheme:

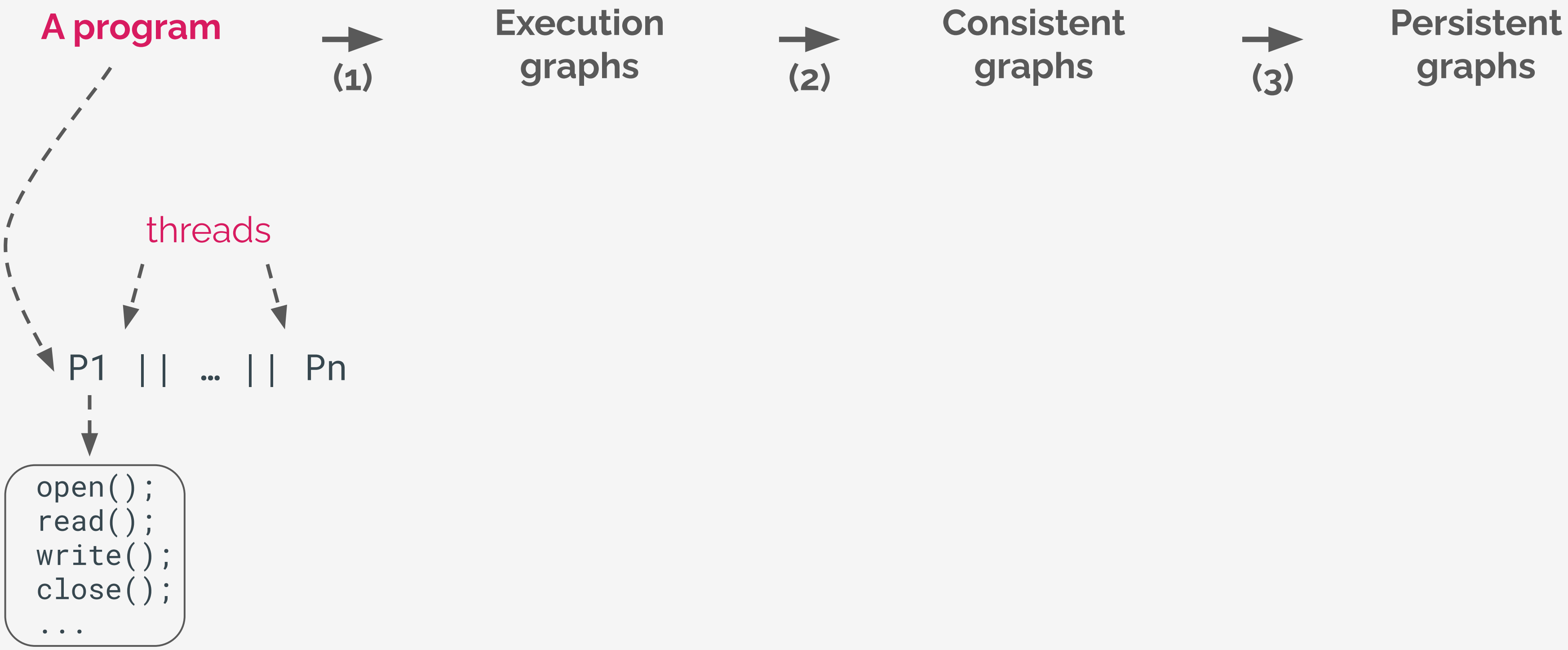


A program (1) Execution graphs (2) Consistent graphs (3) Persistent graphs

V = Events (Read, Write, ...)

E = Orders/Relations (**po**, **rf**, ...)

General framework scheme:



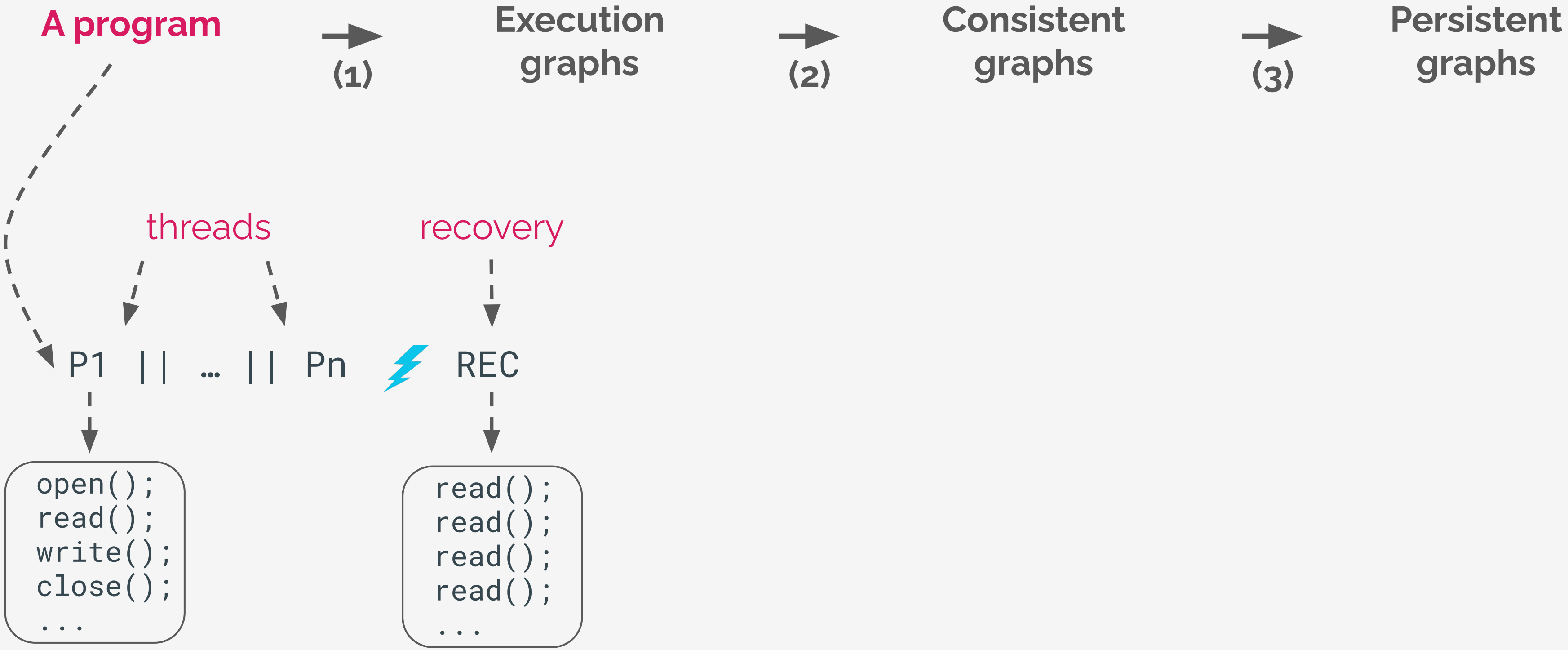
A program → (1) Execution graphs → (2) Consistent graphs → (3) Persistent graphs

threads

P1 || … || Pn

```
open();
read();
write();
close();
...
```

General framework scheme:



A program → (1) Execution graphs → (2) Consistent graphs → (3) Persistent graphs

threads

recovery

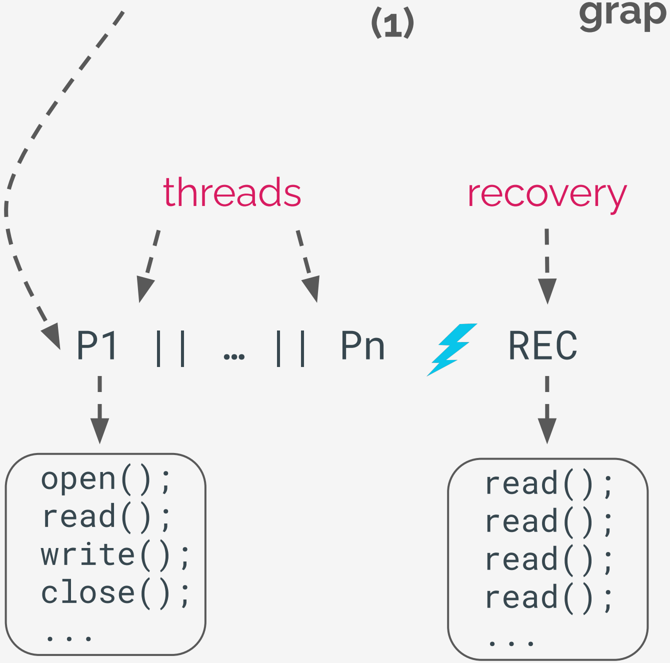
P1 || … || Pn ⚡ REC

```
open();
read();
write();
close();
...
```

```
read();
read();
read();
read();
...
```

General framework scheme:

A program → (1) Execution graphs → (2) Consistent graphs → (3) Persistent graphs



```
open();
read();
write();
close();
...
```

```
read();
read();
read();
read();
...
```

(2) Consistency:

Based on the C++ **memory** model

(3) Persistency:

- Defines **snapshots** - sets of writes surviving a crash
- Enforces the recovery to *read from a snapshot*
- Based on the *persists before* relation (**pb**) – the order in which writes become durable

Objective 2

Goal: specializing the framework for ext4

1. Define **execution generation** from programs
2. Define **pb**

Methodology:

- consulting the **sources** (VFS + ext4)
- running **litmus tests**

Objective 2

Goal: specializing the framework for ext4

1. Define **execution generation** from programs as a procedure generating *events* for each *system call*
2. Define **pb**

Methodology:

- consulting the **sources** (VFS + ext4)
- running **litmus tests**

Objective 2

Goal: specializing the framework for ext4

1. Define **execution generation** from programs as a procedure generating *events* for each *system call*
2. Define **pb**

Methodology:

- consulting the **sources** (VFS + ext4)
- running **litmus tests**:
  - data race tests for consistency
  - write tests for persistency

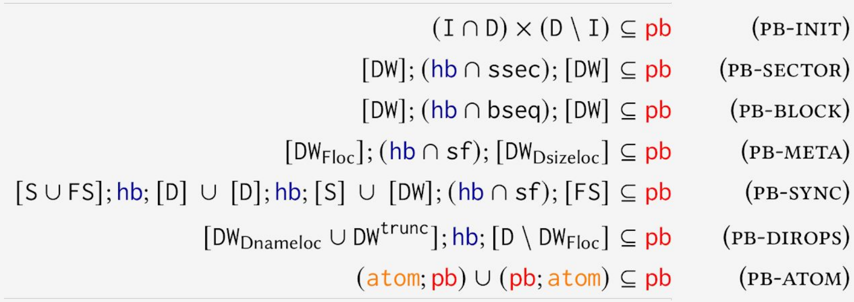
⇓

Correctness

Objective 2

Goal: specializing the framework for ext4

1. Define **execution generation** from programs as a procedure generating *events* for each *system call*
2. Define **pb** as the minimal relation satisfying the axioms:



$$
\begin{array}{rr}
(\mathsf{I} \cap \mathsf{D}) \times (\mathsf{D} \setminus \mathsf{I}) \subseteq \mathsf{pb} & (\textsc{pb-init}) \\
[\mathsf{DW}]; (\mathsf{hb} \cap \mathsf{ssec}); [\mathsf{DW}] \subseteq \mathsf{pb} & (\textsc{pb-sector}) \\
[\mathsf{DW}]; (\mathsf{hb} \cap \mathsf{bseq}); [\mathsf{DW}] \subseteq \mathsf{pb} & (\textsc{pb-block}) \\
[\mathsf{DW}_{\mathsf{Floc}}]; (\mathsf{hb} \cap \mathsf{sf}); [\mathsf{DW}_{\mathsf{Dsizeloc}}] \subseteq \mathsf{pb} & (\textsc{pb-meta}) \\
[\mathsf{S} \cup \mathsf{FS}]; \mathsf{hb}; [\mathsf{D}] \;\cup\; [\mathsf{D}]; \mathsf{hb}; [\mathsf{S}] \;\cup\; [\mathsf{DW}]; (\mathsf{hb} \cap \mathsf{sf}); [\mathsf{FS}] \subseteq \mathsf{pb} & (\textsc{pb-sync}) \\
[\mathsf{DW}_{\mathsf{Dnameloc}} \cup \mathsf{DW}^{\mathsf{trunc}}]; \mathsf{hb}; [\mathsf{D} \setminus \mathsf{DW}_{\mathsf{Floc}}] \subseteq \mathsf{pb} & (\textsc{pb-dirops}) \\
(\mathsf{atom}; \mathsf{pb}) \cup (\mathsf{pb}; \mathsf{atom}) \subseteq \mathsf{pb} & (\textsc{pb-atom})
\end{array}
$$

Methodology:

- consulting the **sources** (VFS + ext4)
- running **litmus tests**:
  - data race tests for consistency
  - write tests for persistency

⇓

Correctness

Objective 3

Goal: adapt ext4 semantics for effective formal verification

(3) Persistency:

recovery reads from a snapshot

Exponential

Objective 3

Goal: adapt ext4 semantics for effective formal verification

~~**(3) Persistency:**~~

~~**recovery reads from a snapshot**~~

Exponential

Objective 3

Goal: adapt ext4 semantics for effective formal verification

~~**(3) Persistency:**~~

~~**recovery reads from a snapshot**~~

Exponential

(3') Persistency:

executions do not contain certain paths

$$[\mathrm{REC}]; \mathtt{rb}; \mathtt{atom}^?; \mathtt{pb}^?; \mathtt{rf}; [\mathrm{REC}] = \emptyset$$

Objective 3

Goal: adapt ext4 semantics for effective formal verification

~~**(3) Persistency:**~~

~~**recovery reads from a snapshot**~~

Exponential

(3') Persistency:

executions do not contain certain paths

$$[\mathrm{REC}] ; \mathtt{rb} ; \mathtt{atom}^{?} ; \mathtt{pb}^{?} ; \mathtt{rf} ; [\mathrm{REC}] = \emptyset$$

Equivalent to (3) (proved)

Axiomatic

Can be used by model checkers

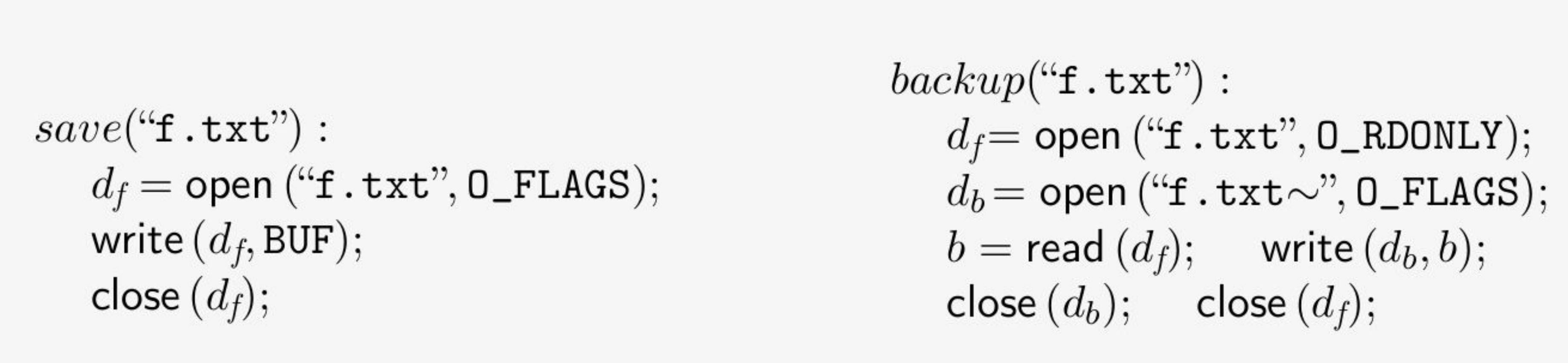
Results

- Built the general filesystem persistency+consistency framework (employs order **pb**; applicable to other filesystems and NVM)
- Constructed the ext4 semantics (by consulting the sources + running litmus tests)
- Adapted the semantics for formal verification (it helped to found anomalies in nano, vim, emacs)
- The work was published on POPL'21 (CORE A*)

M. Kokologiannakis, I. Kaysin, A. Raad, and V. Vafeiadis. **PerSeVerE: Persistency semantics for verification under ext4**. Proc. ACM Program. Lang., (POPL), 2021

Backup slides

Data loss anomalies in text editors



```
$save(“f.txt”)$ :
    $d_f$ = open (“f.txt”, O_FLAGS);
    write ($d_f$, BUF);
    close ($d_f$);
```

```
$backup(“f.txt”)$ :
    $d_f$ = open (“f.txt”, O_RDONLY);
    $d_b$ = open (“f.txt∼”, O_FLAGS);
    $b$ = read ($d_f$);      write ($d_b$, $b$);
    close ($d_b$);      close ($d_f$);
```

Persists Before (Ext4 specific)

$$(\mathtt{I} \cap \mathtt{D}) \times (\mathtt{D} \setminus \mathtt{I}) \subseteq \mathsf{pb} \tag{PB-INIT}$$

$$[\mathtt{DW}]; (\mathsf{hb} \cap \mathtt{ssec}); [\mathtt{DW}] \subseteq \mathsf{pb} \tag{PB-SECTOR}$$

$$[\mathtt{DW}]; (\mathsf{hb} \cap \mathtt{bseq}); [\mathtt{DW}] \subseteq \mathsf{pb} \tag{PB-BLOCK}$$

$$[\mathtt{DW}_{\mathrm{Floc}}]; (\mathsf{hb} \cap \mathtt{sf}); [\mathtt{DW}_{\mathrm{Dsizeloc}}] \subseteq \mathsf{pb} \tag{PB-META}$$

$$[\mathtt{S} \cup \mathtt{FS}]; \mathsf{hb}; [\mathtt{D}] \;\cup\; [\mathtt{D}]; \mathsf{hb}; [\mathtt{S}] \;\cup\; [\mathtt{DW}]; (\mathsf{hb} \cap \mathtt{sf}); [\mathtt{FS}] \subseteq \mathsf{pb} \tag{PB-SYNC}$$

$$[\mathtt{DW}_{\mathrm{Dnameloc}} \cup \mathtt{DW}^{\mathtt{trunc}}]; \mathsf{hb}; [\mathtt{D} \setminus \mathtt{DW}_{\mathrm{Floc}}] \subseteq \mathsf{pb} \tag{PB-DIROPS}$$

$$(\mathtt{atom}; \mathsf{pb}) \cup (\mathsf{pb}; \mathtt{atom}) \subseteq \mathsf{pb} \tag{PB-ATOM}$$

where $\mathtt{atom} \triangleq ([\mathtt{DW} \setminus \mathtt{DW}^{\mathtt{zero}}]; (\mathtt{ssec} \cap \mathtt{sid}); [\mathtt{DW} \setminus \mathtt{DW}^{\mathtt{zero}}]) \cup ([\mathtt{DW}^{\mathtt{rename}}]; \mathtt{sid}; [\mathtt{DW}^{\mathtt{rename}}])$.

System Call -> Events

```
write(a.txt, “hi”)
```

→

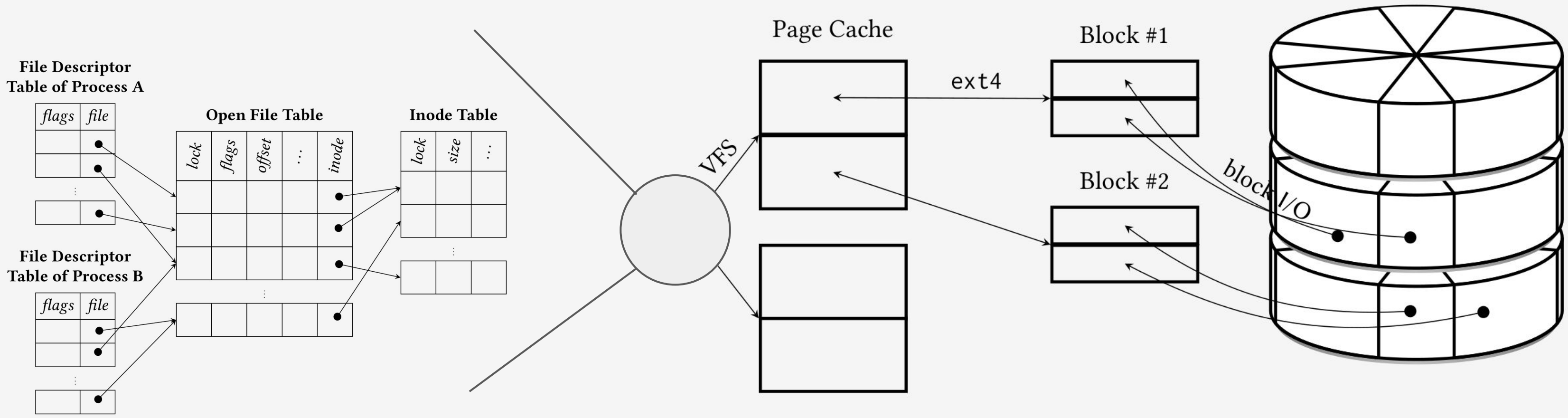
R size[a.txt]

W a.txt[0] "h"

W a.txt[1] "i"

W size[a.txt]

I/O Stack


File Descriptor Table of Process A
flags
file
Open File Table
lock
flags
offset
...
inode
Inode Table
lock
size
...
File Descriptor Table of Process B
flags
file
VFS
Page Cache
ext4
Block #1
Block #2
block I/O

Consistent executions

1. $\mathrm{cons}_{\mathcal{M}}(G)$

$$\mathrm{cons}_{\mathcal{M}}(.) : \mathrm{Exec} \longrightarrow \{\mathrm{true}, \mathrm{false}\}$$

$$\mathsf{hb} \triangleq (\mathsf{po} \cup \mathtt{sw})^{+} \text{ is irreflexive}$$

$$(\mathsf{po} \cup \mathtt{rf})^{+} \text{ is irreflexive}$$

2. $G.\mathsf{pb}$ is irreflexive

```
procedure BufferRead($f$, $buf$, $count$, $o$)
  $\rightsquigarrow$ $\mathtt{MR}(ms_f)$ in $size$
  $m \leftarrow \mathsf{min}(count, size - o)$
  for $i = 0$ to $m - 1$ do
    $\rightsquigarrow$ $\mathtt{DR}((f, o + i))$ in $buf[i]$
```

```
procedure BufferWrite($f$, $buf$, $count$, $o$)
  $\rightsquigarrow$ $\mathtt{L}(f)$
  $\rightsquigarrow$ $\mathtt{MR}(ms_f)$ in $size$
  if $\mathtt{isPreallocBlock}(o, count, size)$ then
    $end \leftarrow \mathsf{min}(count + o, \mathtt{getLastBlockEnd}(f))$
    for $i = size$ to $end - 1$ do $\rightsquigarrow$ $\mathtt{DW}^{\mathtt{zero}}((f, i), 0)$
    $\rightsquigarrow$ $\mathtt{DW}^{\mathtt{zero}}(ds_f, end)$
    $size \leftarrow end$
  if $o > size$ then
    for $i = size$ to $o - 1$ do $\rightsquigarrow$ $\mathtt{DW}^{\mathtt{norm}}((f, i), 0)$
  for $i = 0$ to $count - 1$ do
    $\rightsquigarrow$ $\mathtt{DW}^{\mathtt{norm}}((f, o + i), buf[i])$
    if $(\mathtt{isFstB}(f\ o{+}i{+}1) \vee i{=}count{-}1)) \wedge o{+}i > size$ then
      $\rightsquigarrow$ $\mathtt{DW}^{\mathtt{norm}}(ds_f, o + i)\,;\mathtt{MW}(ms_f, o + i)$
  $\rightsquigarrow$ $\mathtt{U}(f)$
```

```
procedure pread($d_f$, $buf$, $count$, $o$)
  BufferRead($f$, $buf$, $count$, $o$)
```

```
procedure read($d_f$, $buf$, $count$)
  $\rightsquigarrow$ $\mathtt{L}(d_f)$
  $\rightsquigarrow$ $\mathtt{MR}(ol_{d_f})$ in $o$
  BufferRead($f$, $buf$, $count$, $o$)
  $\rightsquigarrow$ $\mathtt{MW}(ol_{d_f}, o + count)$
  $\rightsquigarrow$ $\mathtt{U}(d_f)$
```

```
procedure pwrite($d_f$, $buf$, $count$, $o$)
  BufferWrite($f$, $buf$, $count$, $o$)
```

```
procedure write($d_f$, $buf$, $count$)
  $\rightsquigarrow$ $\mathtt{L}(d_f)$
  $\rightsquigarrow$ $\mathtt{MR}(ol_{d_f})$ in $o$
  BufferWrite($f$, $buf$, $count$, $o$)
  $\rightsquigarrow$ $\mathtt{MW}(ol_{d_f}, o + count)$
  $\rightsquigarrow$ $\mathtt{U}(d_f)$
```

System Calls

Open:

```
d = open("foo.txt", O_APPEND)
```

Reading:

```
r = read(d,3)
r = pread(d,3,42)
```

Writing:

```
write(d,"foo")
pwrite(d,"bar",0)
```

Close:

```
close(d)
```

Synchronization:

```
sync, fsync
```

Directory operations:

```
create, link, unlink, rename
```

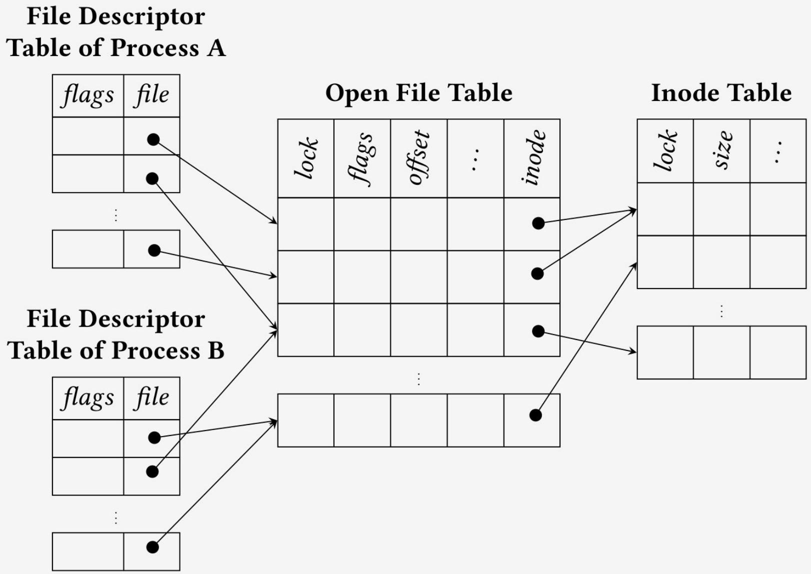


Table 1. Sequential benchmarks used to evaluate PerSeVerE

	P-Total		P-Partial		PerSeVerE	
	Execs	Time	Execs	Time	Execs	Time
pjnl-app-mfl	🕒	🕒	15	57.39	9	0.02
pjnl-crowr-ord	1323	7.84	18	0.04	6	0.02
pjnl-owr-at	36	0.03	5	0.02	5	0.02
pord-owr-N	🕒	🕒	243	308.26	2	0.02
pord-app-mbl	12	0.03	5	0.03	3	0.02
pord-app-sfl	900	1.90	13	0.03	3	0.02
pord-MP+fsync	8724	496.63	3	0.03	3	0.02
pord-trapp-dfl-ord	4044	47.22	5	0.03	3	0.02
pord-crapp-ooo	810	1.38	21	0.03	3	0.02

	P-Total		P-Partial		PerSeVerE	
	Execs	Time	Execs	Time	Execs	Time
pord-MP+osync	8724	596.20	3	0.03	3	0.02
pord-owrapp-ord	450	0.39	10	0.03	4	0.02
pord-owrapp-ord2	3780	26.69	21	0.03	5	0.02
pord-owr-sbl	592	0.26	25	0.02	6	0.02
pord-rnm-at2	696	0.14	9	0.06	5	0.02
pord-rnmtrapp-ord	3240	5.87	18	0.05	6	0.02
pord-app-N	🕒	🕒	🕒	🕒	5	0.03
nano-backup-old	🕒	🕒	🕒	🕒	46	0.08
nano-backup-fix	🕒	🕒	🕒	🕒	10	0.04

Table 2. Concurrent benchmarks used to evaluate PerSeVerE

	P-Total		P-Partial		PerSeVerE	
	Execs	Time	Execs	Time	Execs	Time
pord-wr+rdwr+fsync	🕒	🕒	206	0.30	6	0.03
pord-wr+wr-N	🕒	🕒	🕒	🕒	6240	2.13
pord-wr+wr-N-RR	🕒	🕒	🕒	🕒	17	0.72
pord-wr+wr-N-unord	🕒	🕒	🕒	🕒	22 680	27.74

	P-Total		P-Partial		PerSeVerE	
	Execs	Time	Execs	Time	Execs	Time
pord-wr+wr-N-join-main	🕒	🕒	🕒	🕒	216	1.19
pord-wr+wr-N-join-thr	🕒	🕒	🕒	🕒	2052	37.00
pord-rd-wr+wr-N-cont	🕒	🕒	🕒	🕒	12 888	61.99
pord-rd-wr+wr-N-join	🕒	🕒	🕒	🕒	216	3.76

Пример: перезапись

```
// a.txt = “000...0”
a.txt <- “111...1”
```

```
      a.txt
“000000000000”
“110011110000”
“111100001100”
“111111111111”
       ...
```

- Сектора записываются атомарно
- Сектора внутри одного блока упорядочены