Failboxes: Provably safe exception handling

Bart Jacobs and Frank Piessens

Katholieke Universiteit Leuven

Overview of Presentation

- Purpose of exceptions: Dependency safety
- Conflicts with:
  - Non-exception-safe objects and:
  - Try-catch
  - Threads and locks
  - Thread.stop
  - Try-finally
- Solution: Failboxes!

Overview of Presentation

- **Purpose of exceptions: Dependency safety**
- Conflicts with:
  - Non-exception-safe objects and:
  - Try-catch
  - Threads and locks
  - Thread.stop
  - Try-finally
- Solution: Failboxes!

Problem

Purpose of exceptions

=

Skipping operations

that depend on success

of failed operations

(*dependency safety*)

Dependency safety: Example 1

```
int* x = malloc(1024 * sizeof(int));

x[1023] = 42;
```

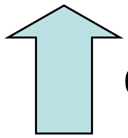
Dependency safety: Example 1

```
int* x = malloc(1024 * sizeof(int));
      ↑ depends on
x[1023] = 42;
```

Dependency safety: Example 1

```
int* x = malloc(1024 * sizeof(int));
```

x == 0

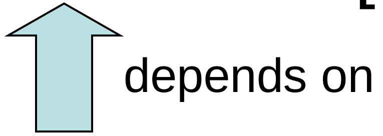


```
x[1023] = 42;
```

Dependency violation!

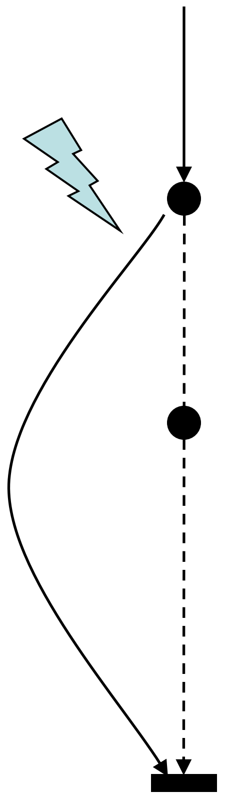
Dependency Safety: Example 2

```
int[] x = new int[1024];
```



```
x[1023] = 42;
```

Dependency Safety: Example 2



```
int[] x = new int[1024];
x[1023] = 42;
```

Dependency-safe!

Problem

Dependency safety

=

Hard to achieve

Non-exception-safe objects

Try-catch

Threads and locks

Thread.stop

Try-finally

Overview of Presentation

- Purpose of exceptions: Dependency safety
- Conflicts with:
  - Non-exception-safe objects and:
  - Try-catch
  - Threads and locks
  - Thread.stop
  - Try-finally
- Solution: Failboxes!

Exception Safety

Object is exception-safe

🡘

A checked or unchecked exception

Leaves the object

In a **consistent** state

Example: ArrayList

```
class ArrayList {
  int count;
  int[] elems;
  void add(int x) {
    count++;
    if (count > elems.length) {
      int[] elems2 = new int[count * 2];
      System.arraycopy(elems, elems2, 0, 0, elems.length);
      elems = elems2;
    }
    elems[count – 1] = x;
  }
}
```

Not exception-safe!

Exception Safety

Hard to achieve

Goal:

Achieve

dependency safety

in the **absence** of

exception safety

Overview of Presentation

- Purpose of exceptions: Dependency safety
- Conflicts with:
  - Non-exception-safe objects and:
  - **Try-catch**
  - Threads and locks
  - Thread.stop
  - Try-finally
- Solution: Failboxes!

Example: Interpreter 1

```
while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
    } catch (Throwable t) {t.printStackTrace();}
}
```

Example: Interpreter 1

```
while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
    } catch (Throwable t) {t.printStackTrace();}
}
```

OK!

Example: Interpreter 2

```
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
        … list.add(…); …
    } catch (Throwable t) {t.printStackTrace();}
}
```

Example: Interpreter 2

```
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
        … list.add(…); …
    } catch (Throwable t) {t.printStackTrace();}
}
```

Bad!

Proposed Solution: Failboxes

= Language extension

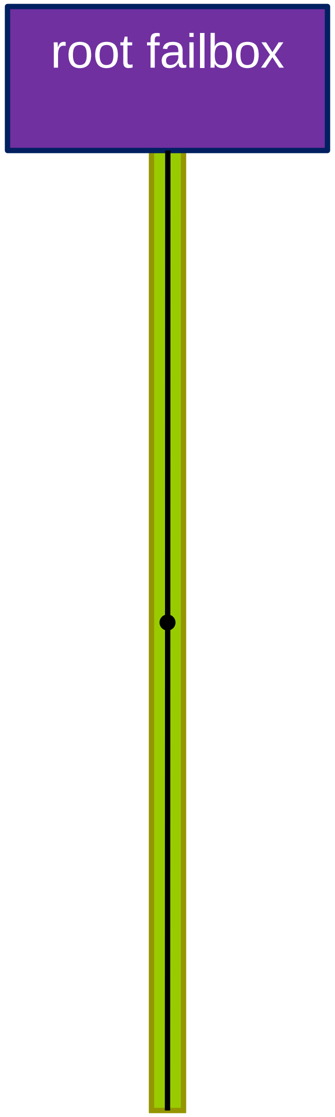
- API class Failbox:

```
class Failbox {
    public Failbox(Failbox parent) { … }
    public static Failbox getCurrent() { … }
    public boolean hasFailed() { … }
    …
}
```

- new statement:

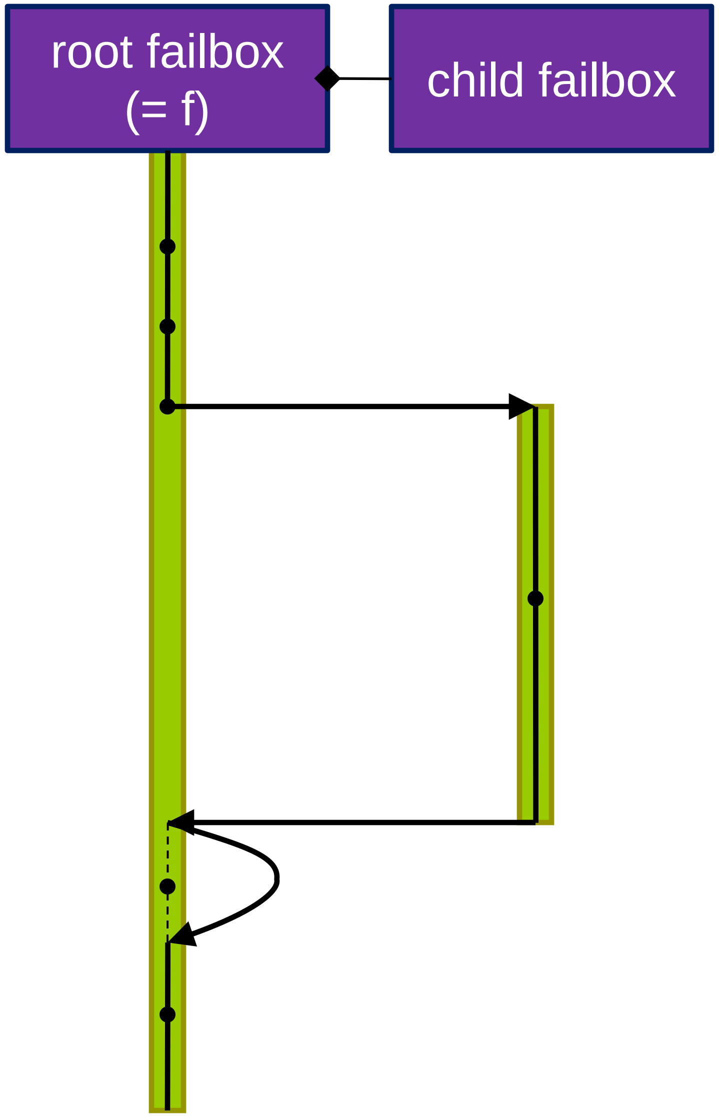
```
enter (f) { ... } catch (Throwable t) { … }
```

Failboxes



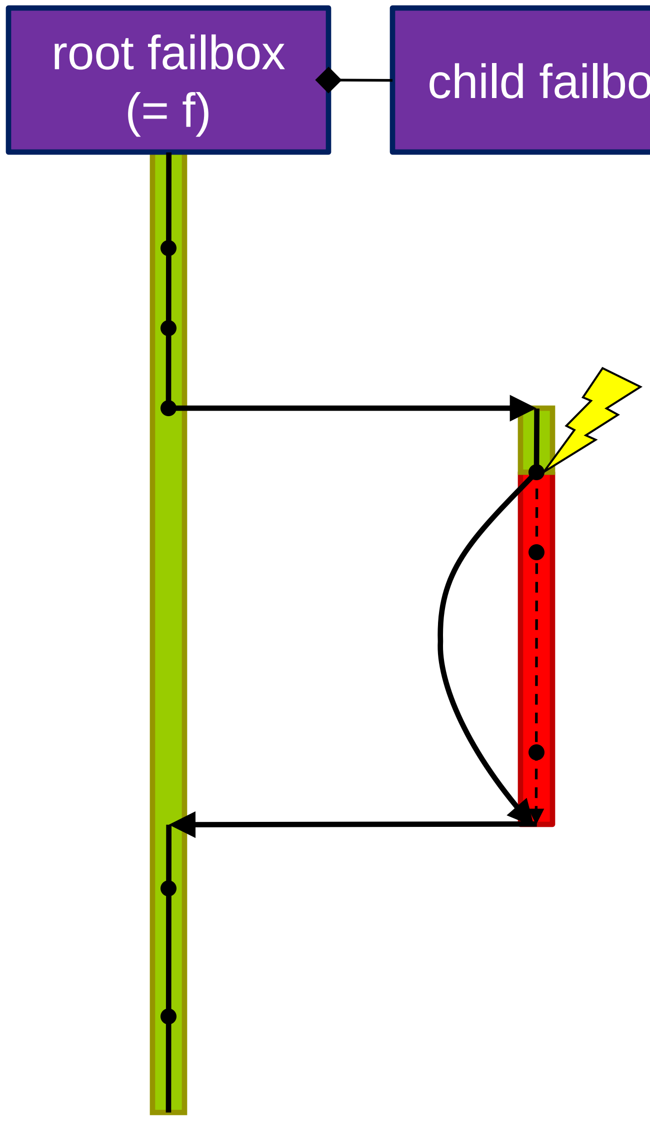
```
public static void main(String[] args) {

    …

}
```

Failboxes



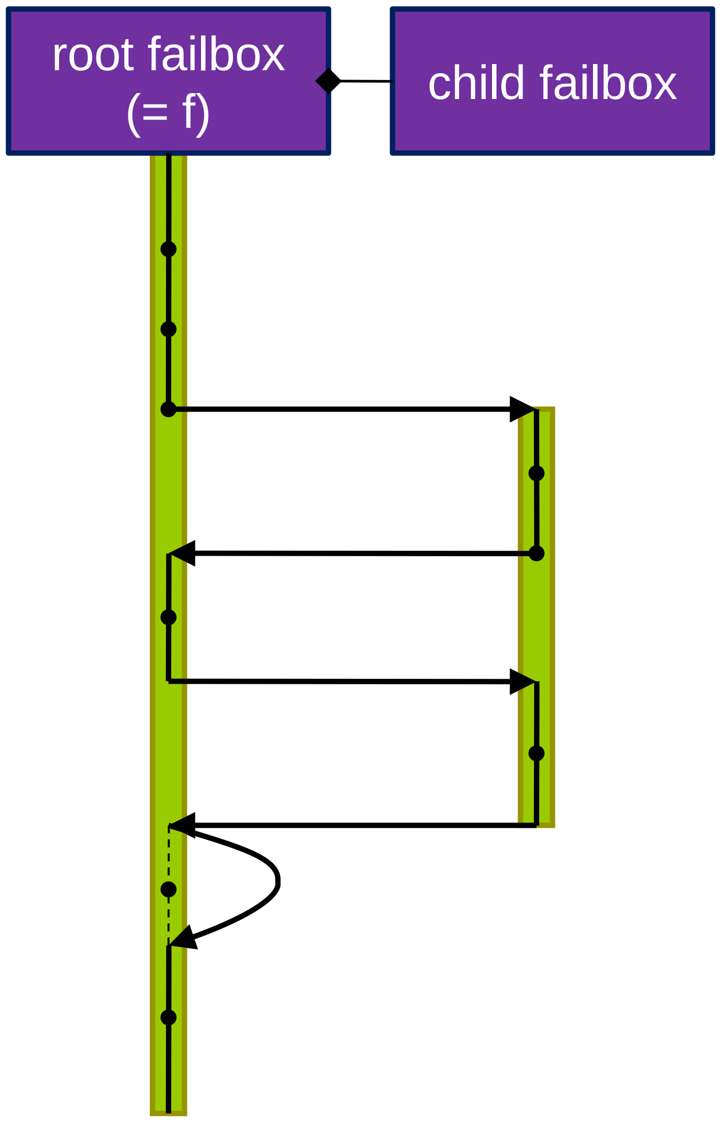
```
public static void main(String[] args) {
    Failbox f = Failbox.getCurrent();
    …
    enter (new Failbox(f)) {



        …



    } catch (Throwable t) {
        …
    }
    …
}
```

Failboxes



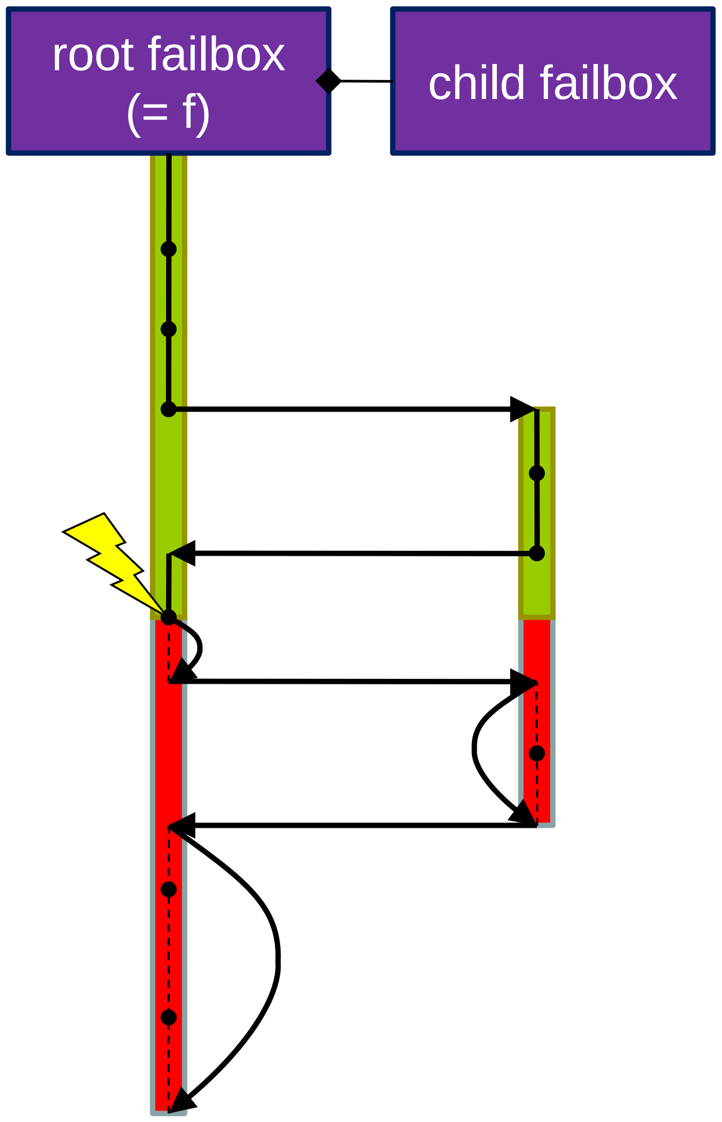
```
public static void main(String[] args) {
    Failbox f = Failbox.getCurrent();
    …
    enter (new Failbox(f)) {
        throw new RuntimeException();
        …


        …
    } catch (Throwable t) {
        …
    }
    …
}
```

Failboxes



```
public static void main(String[] args) {
    Failbox f = Failbox.getCurrent();
    …
    enter (new Failbox(f)) {
        …
        enter (f) {
            …
         } catch (Throwable t) { throw t; }
         …
    } catch (Throwable t) {
        …
    }
    …
}
```

Failboxes



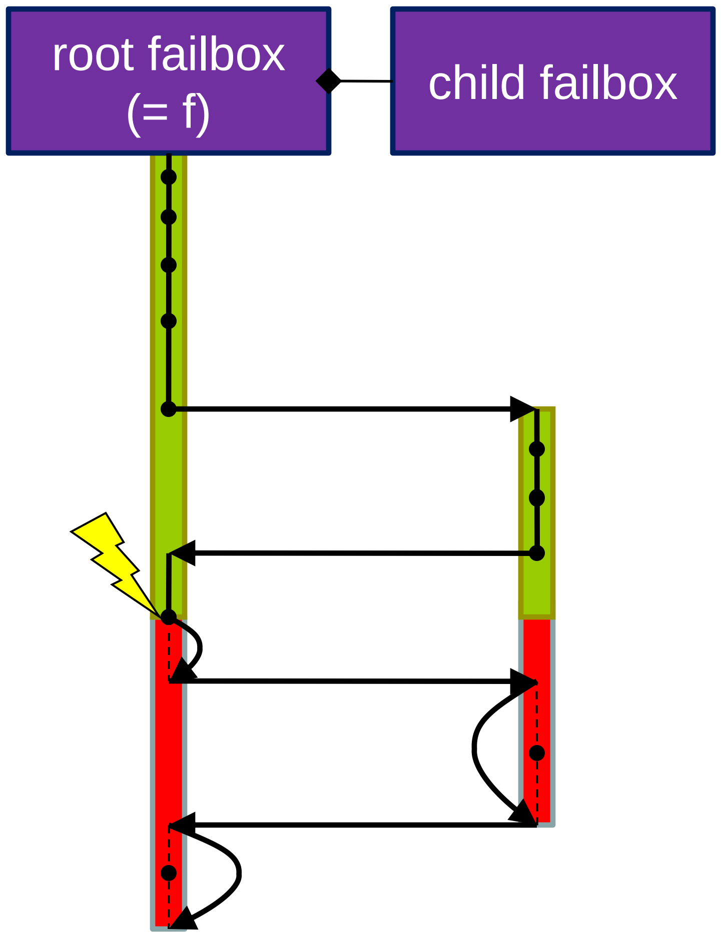
```
public static void main(String[] args) {
    Failbox f = Failbox.getCurrent();
    …
    enter (new Failbox(f)) {
        …
        enter (f) {
            throw new RuntimeException();
        } catch (Throwable t) { throw t; }
        …
    } catch (Throwable t) {
        …
    }
    …
}
```

Example: Interpreter 2

```
ArrayList list = new ArrayList();

while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
        …

        list.add(…);

        …
    } catch (Throwable t)
    { t.printStackTrace(); }
}
```

```
ArrayList list = new ArrayList();
Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand();
    enter (new Failbox(f)) {
        … compute(cmd); …
        …
        enter (f) {
            list.add(…);
        } catch (Throwable t) { throw t; }
        …
    } catch (Throwable t)
    { t.printStackTrace(); }
}
```

Example: Interpreter 2



```
ArrayList list = new ArrayList();
Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand();
    enter (new Failbox(f)) {
        … compute(cmd); …
        …
        enter (f) {
            list.add(…);
        } catch (Throwable t) { throw t; }
        …
    } catch (Throwable t)
    { t.printStackTrace(); }
}
```

Syntactic Sugar

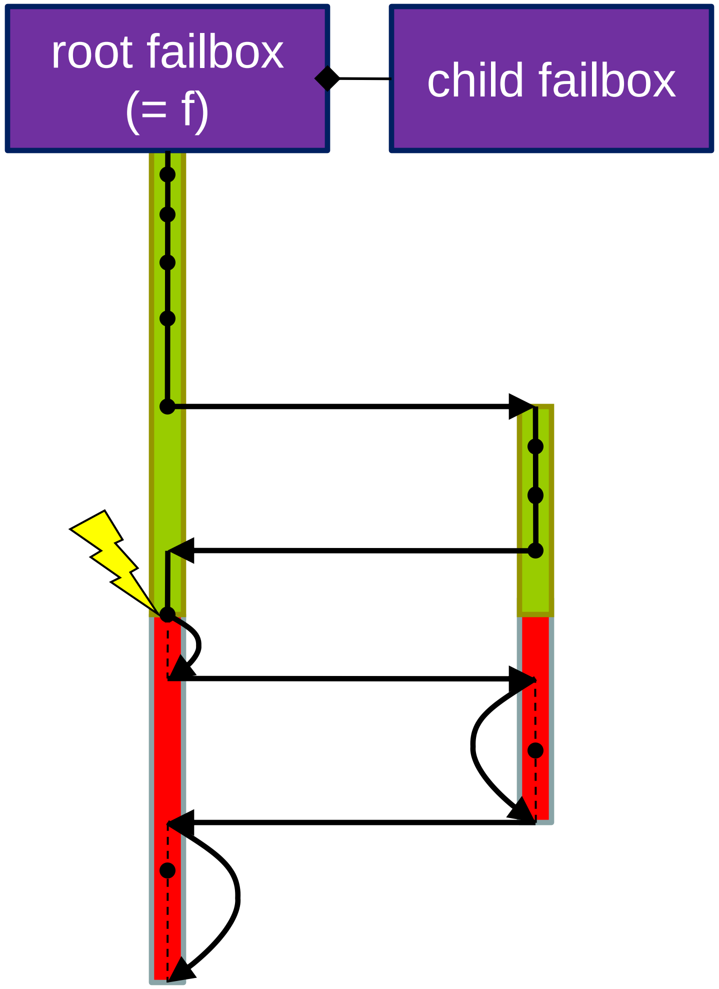
```
try {
    block1
} catch (Throwable t) {
    block2
}
```

```
enter (new Failbox(Failbox.getCurrent())) {
    block1
} catch (Throwable t) {
    block2
}
```

```
enter (f) {
    block
}
```

```
enter (f) {
    block
} catch (Throwable t) { throw t; }
```

Example: Interpreter 2



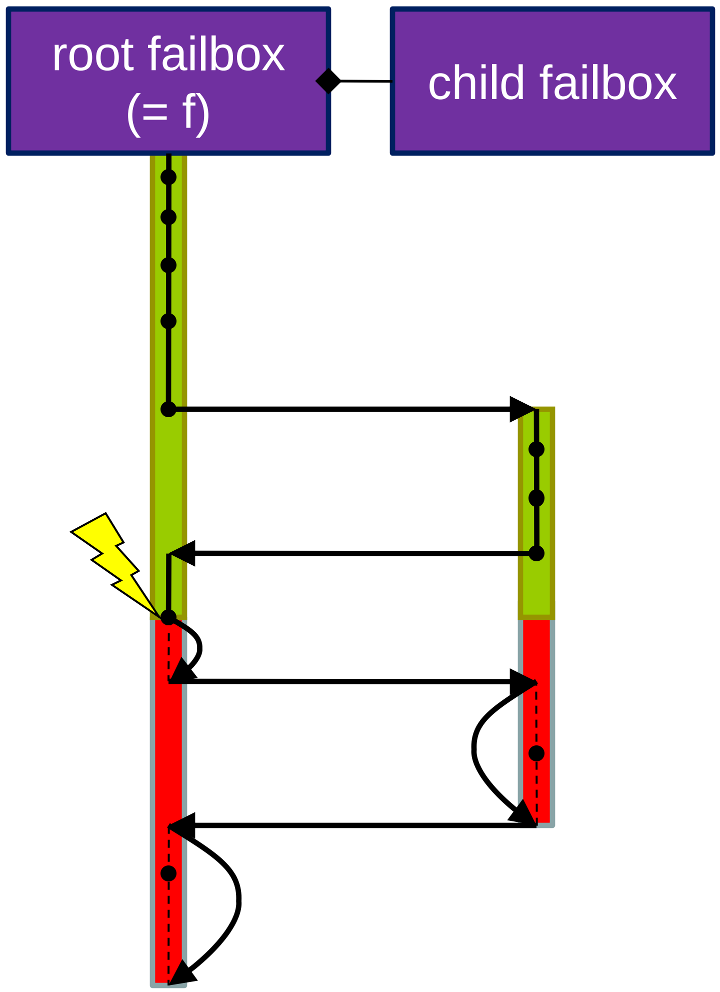
```
ArrayList list = new ArrayList();
Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
        …
        enter (f) {
            list.add(…);
        }
        …
    } catch (Throwable t)
    { t.printStackTrace(); }
}
```

Syntactic Sugar (2)

```
failboxed class C {

    void foo() {

        …

    }
    void bar() {

        …

    }
}
```

```
class C {
    Failbox f = Failbox.getCurrent();
    void foo() {
        enter (f) {
            …
        }
    }
    void bar() {
        enter (f) {
            …
        }
    }
}
```

Example: Interpreter 2



```
ArrayList list = new ArrayList();

while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
        …

        list.add(…);


        …
    } catch (Throwable t)
    { t.printStackTrace(); }
}
```

Overview of Presentation

- Purpose of exceptions: Dependency safety
- Conflicts with:
  - Non-exception-safe objects and:
  - Try-catch
  - **Threads and locks**
  - Thread.stop
  - Try-finally
- Solution: Failboxes!

Example: Concurrent Interpreter

```
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
        }
    }.start();
}
```

Example: Concurrent Interpreter

```
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
        }
    }.start();
}
```

OK!

Example: Concurrent Interpreter 2

```
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { list.add(…); } …
        }
    }.start();
}
```

Example: Concurrent Interpreter 2

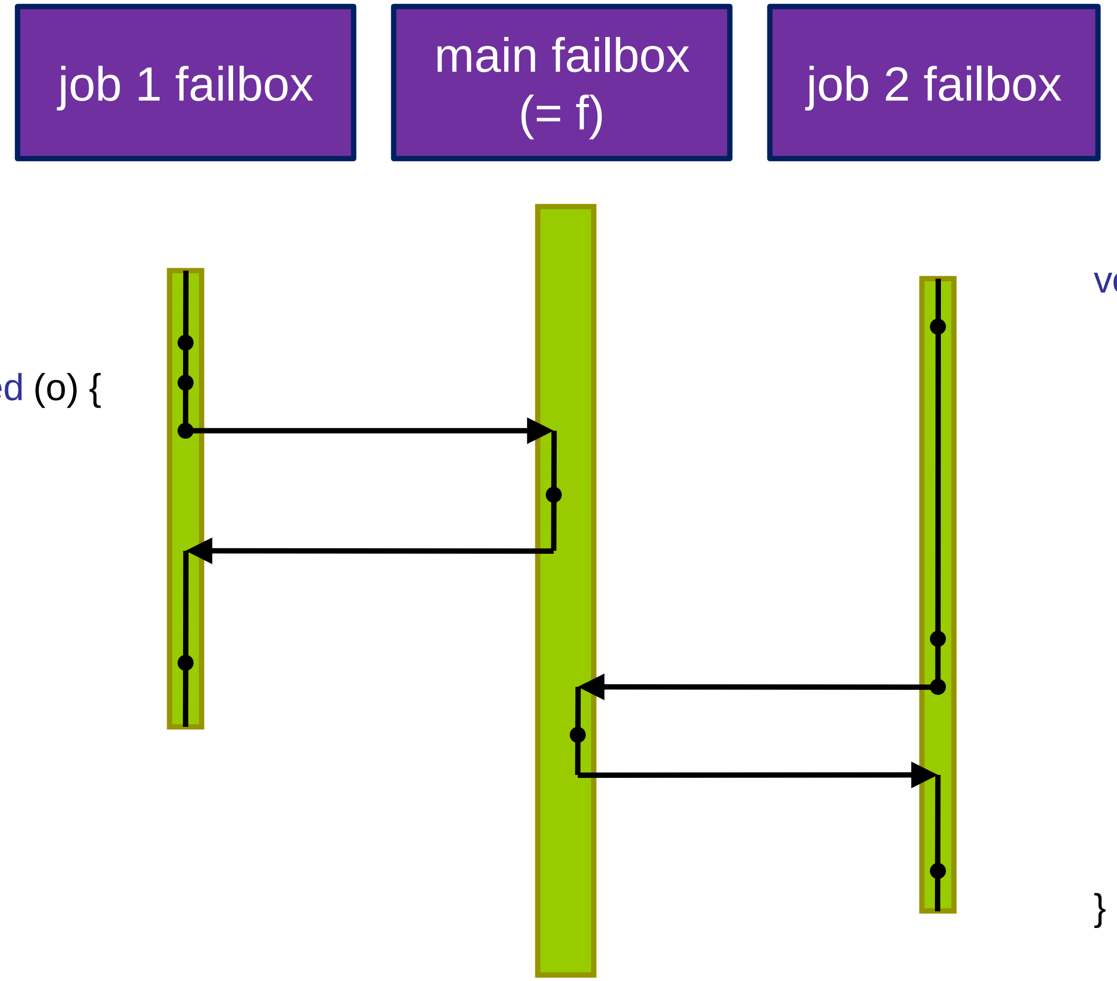
```
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { list.add(…); } …
        }
    }.start();
}
```

Bad!

Solution: Failboxes

```
Failbox f = Failbox.getCurrent();
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { enter (f) { list.add(…); } } …
        }
    }.start();
}
```

Failboxes and Threads

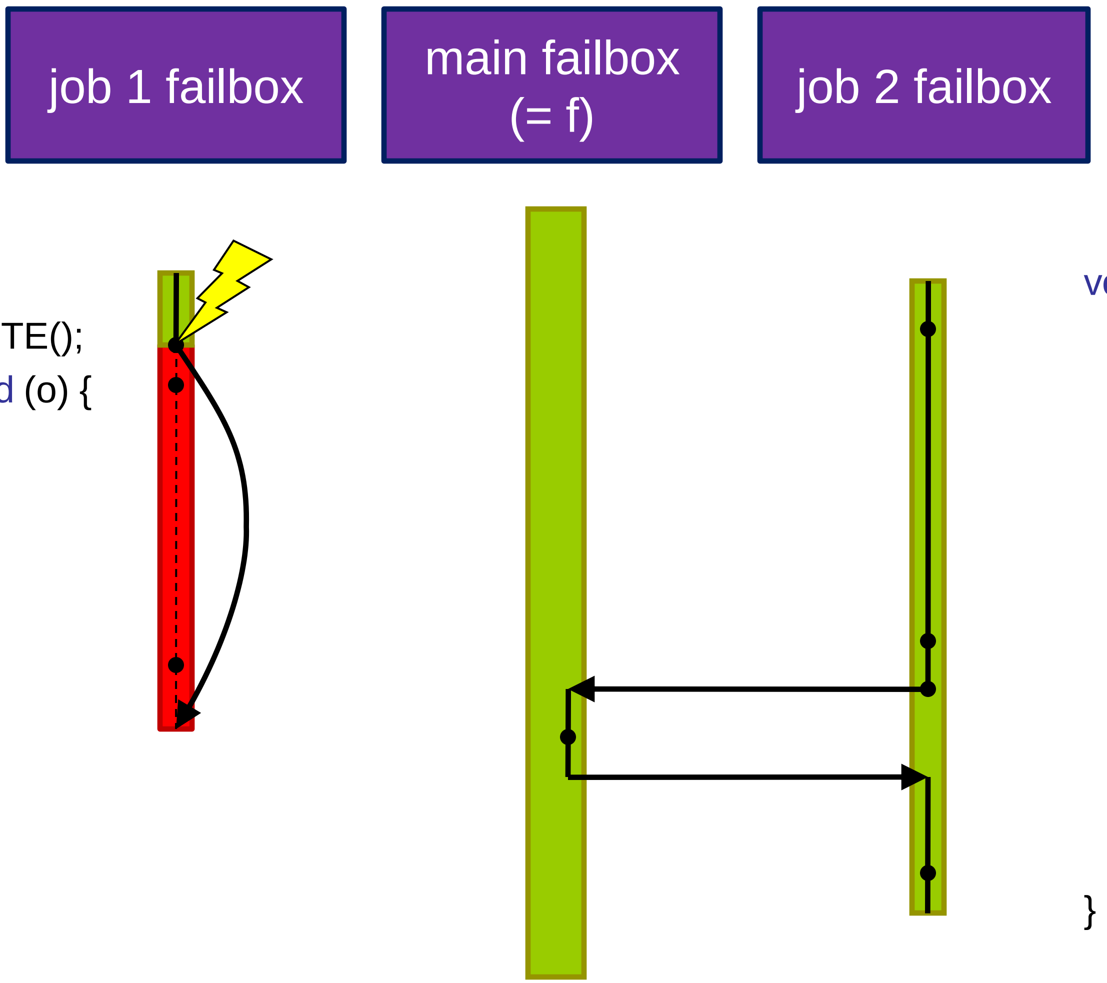


```
void run() {
    …
    synchronized (o) {
        enter (f) {
            ...
        }
    }
    ...
}
```

```
void run() {
    …




    synchronized (o) {
        enter (f) {
            ...
        }
    }
    ...
}
```

Failboxes and Threads

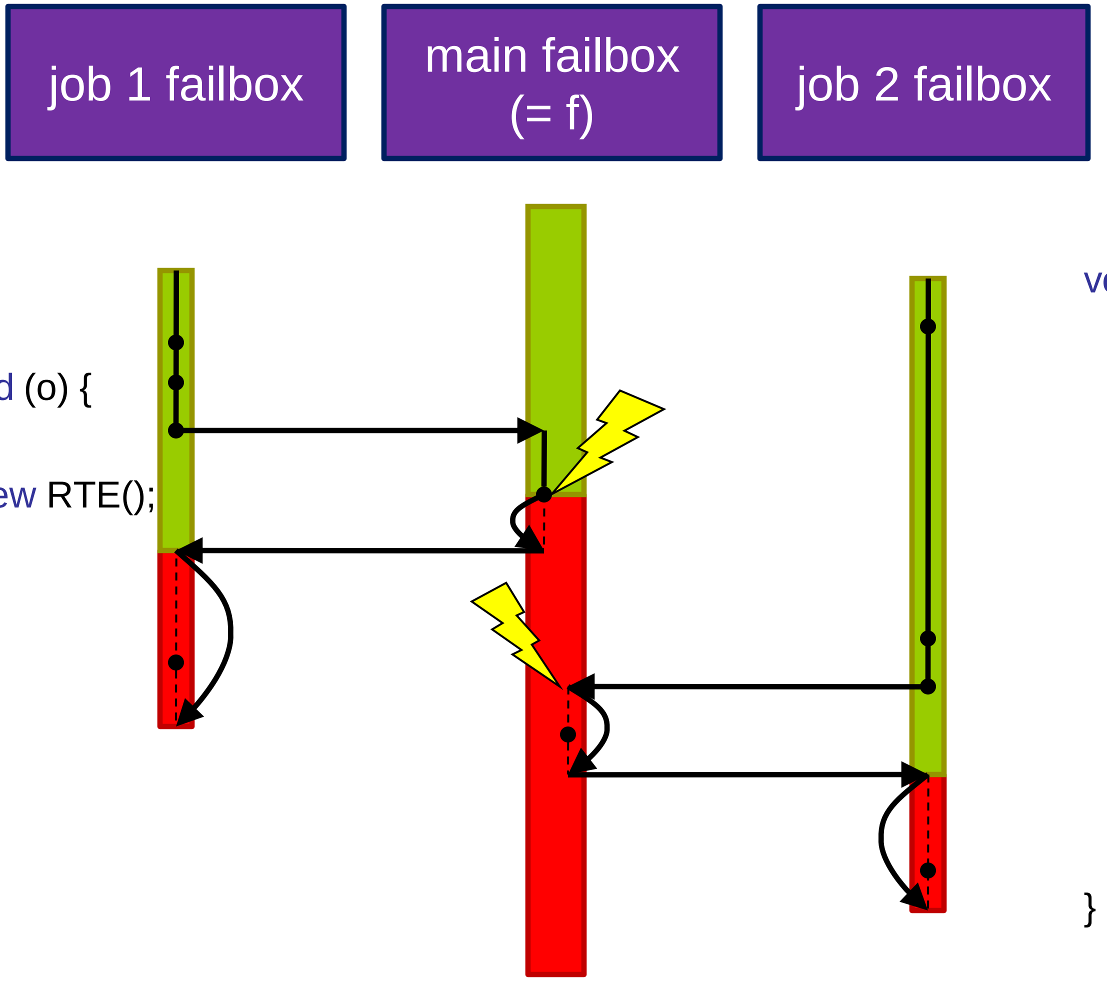
```
void run() {
    throw new RTE();
    synchronized (o) {
        enter (f) {
            ...
        }
    }
    ...
}
```



```
void run() {
    …







    synchronized (o) {
        enter (f) {
            ...
        }
    }
    ...
}
```

Failboxes and Threads

```
void run() {
    …
    synchronized (o) {
        enter (f) {
            throw new RTE();
        }
    }
    …
}
```

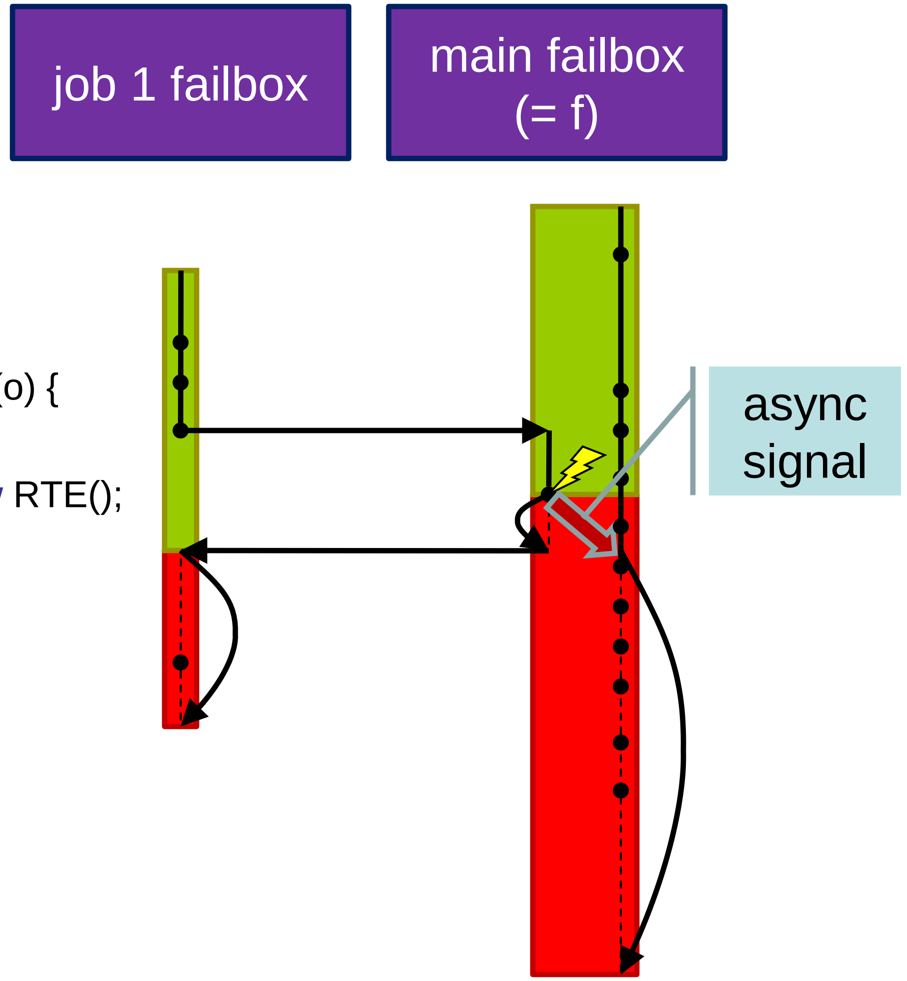


```
void run() {
    …



    synchronized (o) {
        enter (f) {
            …
        }
    }
    …
}
```

The need for Fail Fast

```
Failbox f = Failbox.getCurrent();
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { enter (f) { list.add(…); } } …
        }
    }.start();
}
```

Asynchronous failure: Fail Fast



```
void run() {
   …
   synchronized (o) {
      enter (f) {
         throw new RTE();
      }
   }
   ...
}
```

```
public static void main(String[] args) {
   …


   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();
   serverSocket.accept();



}
```

Solution: Failboxes

```
Failbox f = Failbox.getCurrent();
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { enter (f) { list.add(…); } } …
        }
    }.start();
}
```

Overview of Presentation

- Purpose of exceptions: Dependency safety
- Conflicts with:
  - Non-exception-safe objects and:
  - Try-catch
  - Threads and locks
  - Thread.stop
  - Try-finally
- Solution: Failboxes!

Example: Concurrent Interpreter

```
while (true) {
    String cmd = readCommand();
    new Thread() {
        public void run() {
            … compute(cmd); …
        }
    }.start();
}
```

Example: Concurrent Interpreter

```
ArrayList<Thread> jobs = new ArrayList<Thread>();
while (true) {
    String cmd = readCommand();
    if (cmd.equals(“cancelAll”))
    { for (Thread t : jobs) t.stop(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
        }
    }; jobs.add(t); t.start();
}
```

Example: Concurrent Interpreter

```
ArrayList<Thread> jobs = new ArrayList<Thread>();
while (true) {
    String cmd = readCommand();
    if (cmd.equals("cancelAll"))
    { for (Thread t : jobs) t.stop(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
        }
    }; jobs.add(t); t.start();
}
```

OK!

Example: Concurrent Interpreter

```
ArrayList<Thread> jobs = new ArrayList<Thread>();
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    if (cmd.equals(“cancelAll”))
    { for (Thread t : jobs) t.stop(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { list.add(…); } …
        }
    }; jobs.add(t); t.start();
}
```

Example: Concurrent Interpreter

```
ArrayList<Thread> jobs = new ArrayList<Thread>();
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    if (cmd.equals(“cancelAll”))
    { for (Thread t : jobs) t.stop(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { list.add(…); } …
        }
    }; jobs.add(t); t.start();
}
```

Bad!

Example: Concurrent Interpreter

```
ArrayList<Thread> jobs = new ArrayList<Thread>();
ArrayList list = new ArrayList(); Failbox f =
  Failbox.getCurrent();
while (true) {
  String cmd = readCommand();
  if (cmd.equals("cancelAll"))
  { for (Thread t : jobs) t.stop(); continue; }
  Thread t = new Thread() {
    public void run() {
      … compute(cmd); …
      … synchronized (list) { enter (f) { list.add(…); } } …
    }
  }; jobs.add(t); t.start();
}
```

Example: Concurrent Interpreter

```
ArrayList<Thread> jobs = new ArrayList<Thread>();
ArrayList list = new ArrayList(); Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand();
    if (cmd.equals(“cancelAll”))
    { for (Thread t : jobs) t.stop(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { enter (f) { list.add(…); } } …
        }
    }; jobs.add(t); t.start();
}
```

Safe, but not OK

Example: Concurrent Interpreter

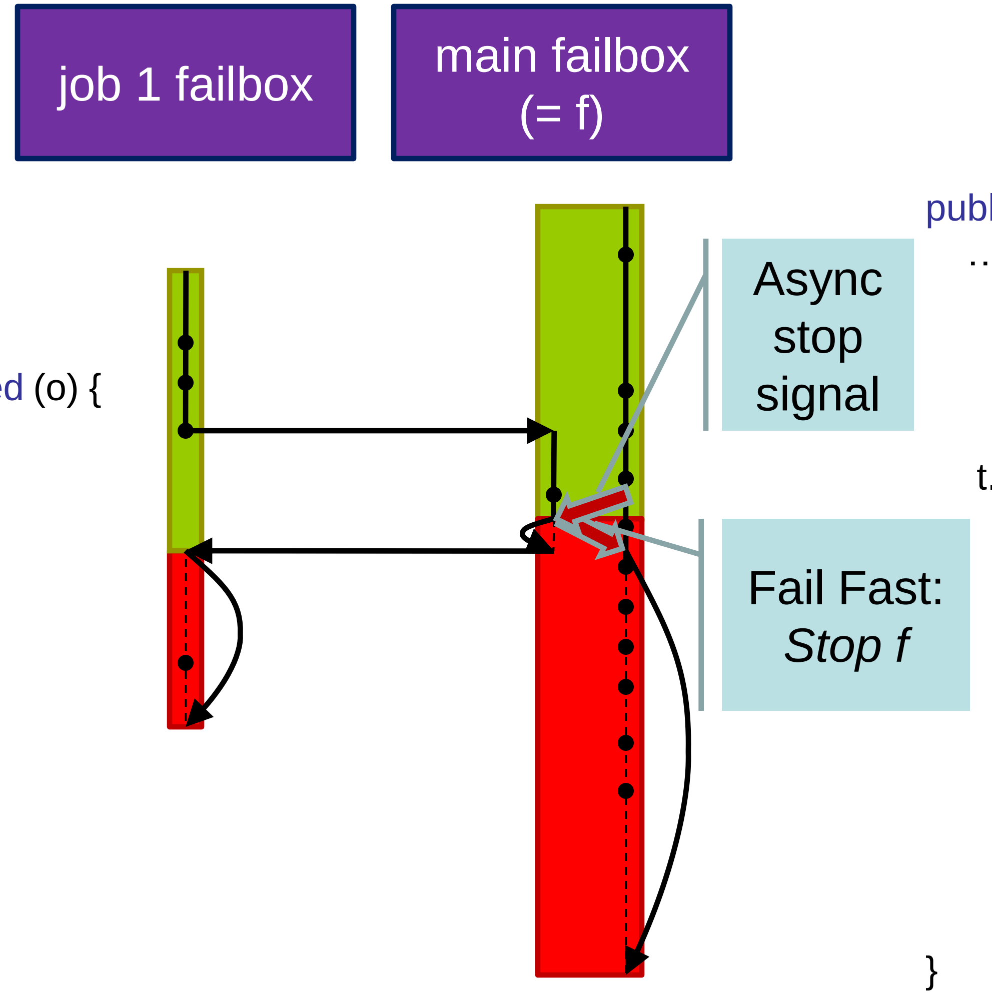
```
ArrayList<Failbox> jobs = new ArrayList<Failbox>();
ArrayList list = new ArrayList(); Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand(); Failbox j = new Failbox(null);
    if (cmd.equals(“cancelAll”))
    { for (Failbox j : jobs) j.cancel(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { enter (f) { list.add(…); } } …
        }
    }; jobs.add(j); t.startInFailbox(j);
}
```

Example: Concurrent Interpreter

```
ArrayList<Failbox> jobs = new ArrayList<Failbox>();
ArrayList list = new ArrayList(); Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand(); Failbox j = new Failbox(null);
    if (cmd.equals(“cancelAll”))
    { for (Failbox j : jobs) j.cancel(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { enter (f) { list.add(…); } } …
        }
    }; jobs.add(j); t.startInFailbox(j);
}
```

OK!

The problem with Thread.stop



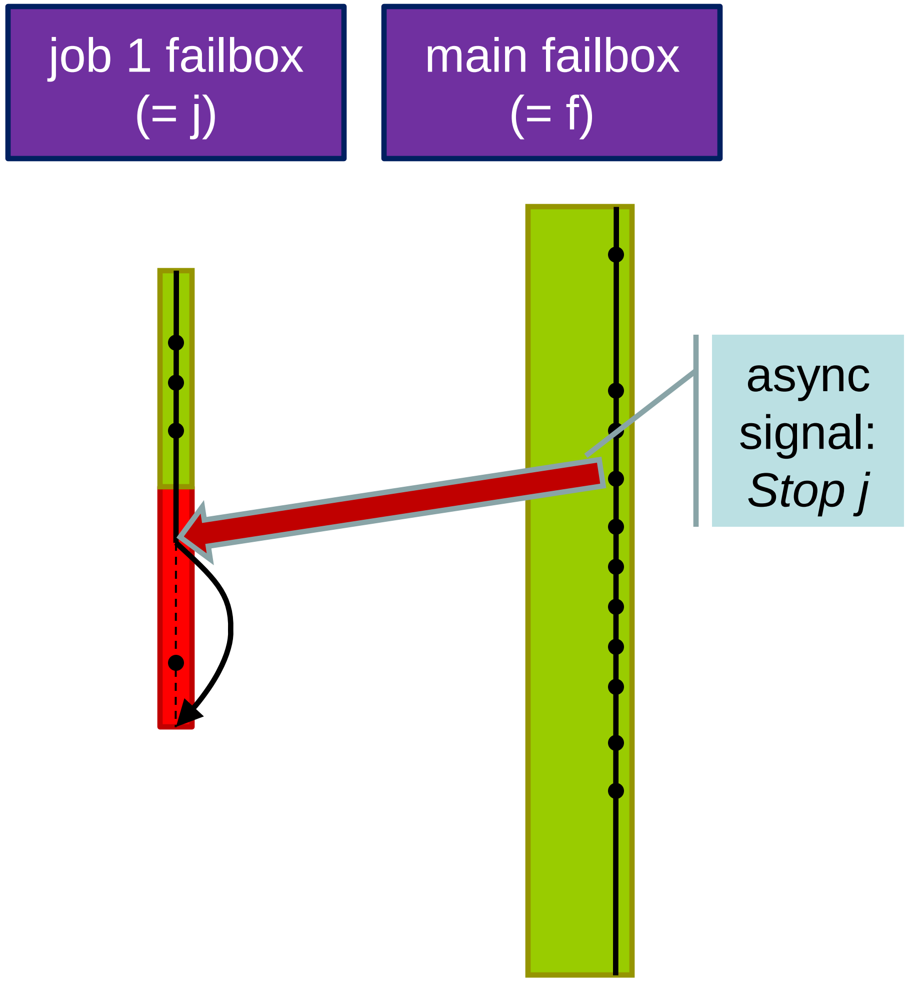
```
void run() {
    …
    synchronized (o) {
        enter (f) {
            ...
        }
    }
    ...
}
```

```
public static void main(String[] args) {
    …

    t.stop();

}
```

Fail Fast for cancellation

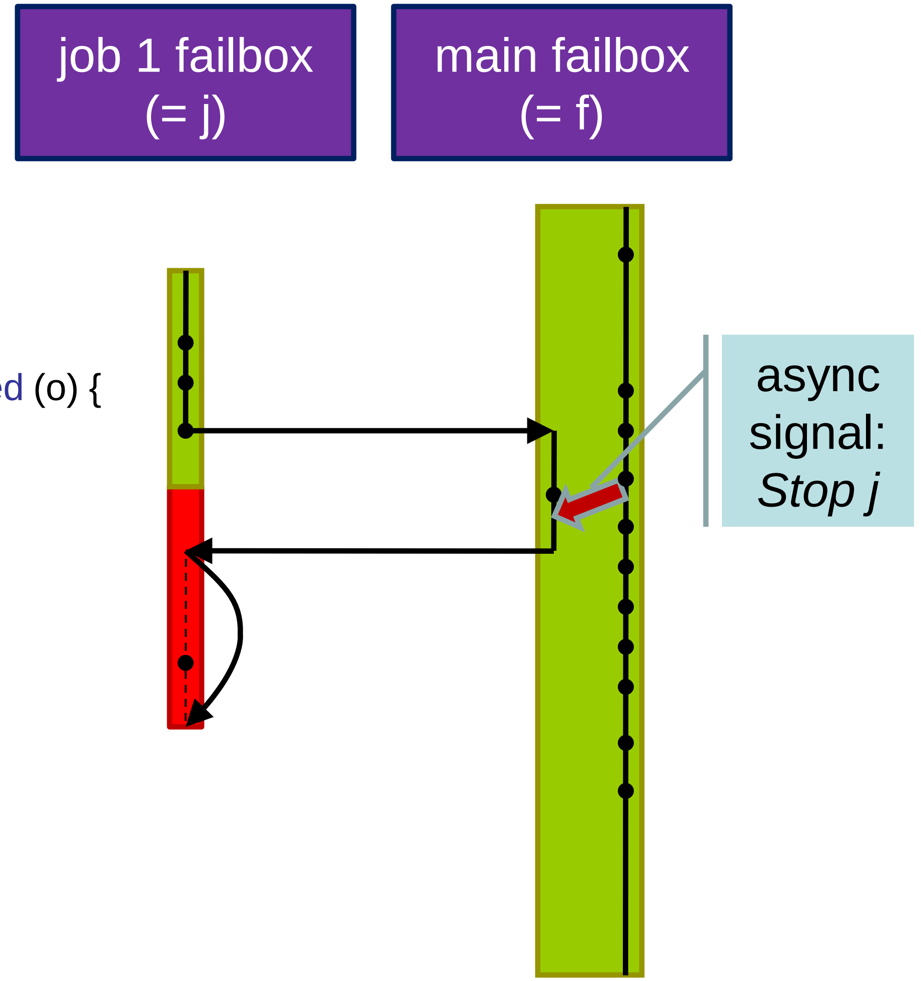
void run() {
    …

    ...
}



```
public static void main(String[] args) {
    …

    j.cancel();

}
```

Fail Fast for cancellation



```
void run() {
    …
    synchronized (o) {
        enter (f) {
            ...
        }
    }
    ...
}
```

```
public static void main(String[] args) {
    …


        j.cancel();


}
```

Example: Concurrent Interpreter

```
ArrayList<Failbox> jobs = new ArrayList<Failbox>();
ArrayList list = new ArrayList(); Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand(); Failbox j = new Failbox(null);
    if (cmd.equals(“cancelAll”))
    { for (Failbox j : jobs) t.cancel(); continue; }
    Thread t = new Thread() {
        public void run() {
            … compute(cmd); …
            … synchronized (list) { enter (f) { list.add(…); } } …
        }
    }; jobs.add(t); t.startInFailbox(j);
}
```

OK!

Overview of Presentation

- Purpose of exceptions: Dependency safety
- Conflicts with:
  - Non-exception-safe objects and:
  - Try-catch
  - Threads and locks
  - Thread.stop
  - Try-finally
- Solution: Failboxes!

Example: Interpreter 1

```
while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
    } catch (Throwable t) {t.printStackTrace();}
}
```

Example: Interpreter 2

```
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    try {
        … compute(cmd); …
        … list.add(…); …
    } catch (Throwable t) {t.printStackTrace();}
}
```

Example: Tidy Interpreter

```
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    try {
        …
        list.add(x);
        try {
            … compute(cmd); … list.sort(); …
        } finally {
            list.remove(x);
        }
        …
    } catch (Throwable t) {t.printStackTrace();}
}
```

Example: Tidy Interpreter

```
ArrayList list = new ArrayList();
while (true) {
    String cmd = readCommand();
    try {
        …
        list.add(x);
        try {
            … compute(cmd); … list.sort(); …
        } finally {
            list.remove(x);
        }
        …
    } catch (Throwable t) {t.printStackTrace();}
}
```

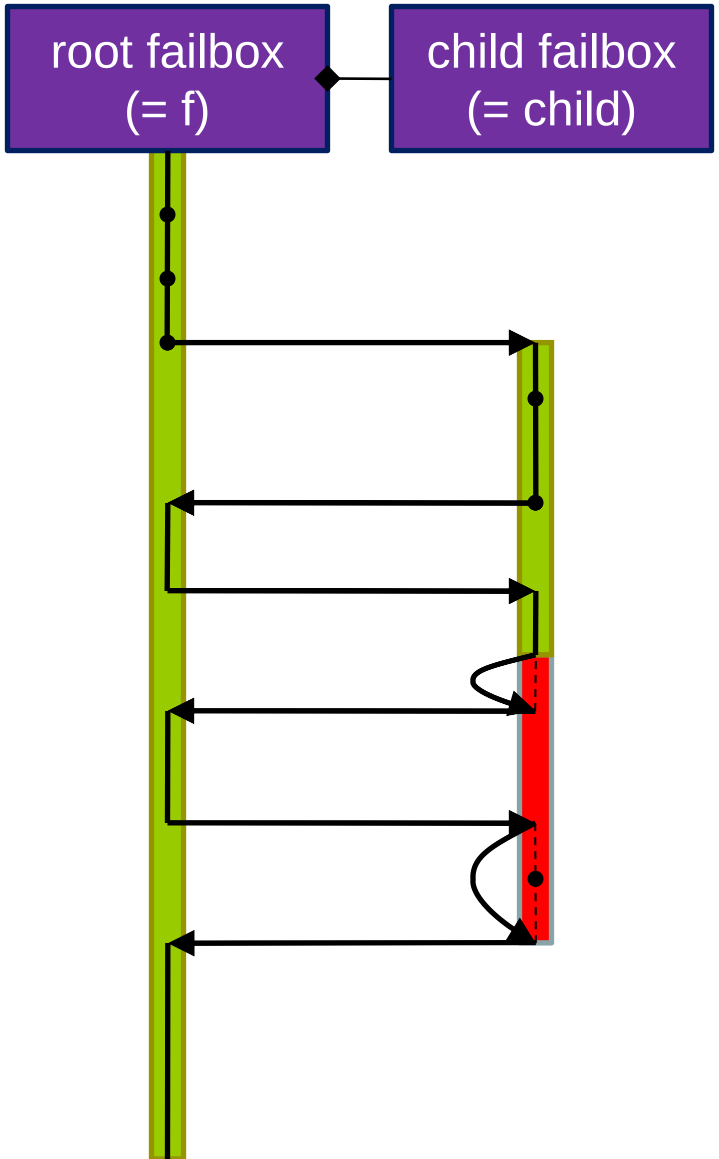
Bad!

Failboxes for safe compensation

```
ArrayList list = new ArrayList();
Failbox f = Failbox.getCurrent();
while (true) {
    String cmd = readCommand();
    try {
        …
        Failbox child = Failbox.getCurrent();
        enter (f) {
            list.add(x);
            enter (child) {
                … compute(cmd); … enter (f) { list.sort(); } …
            } catch (Throwable t) { }
            list.remove(x);
        }
        …
    } catch (Throwable t) {t.printStackTrace();}
}
```

OK!

Failboxes for safe compensation



```
public static void main(String[] args) {
    Failbox f = Failbox.getCurrent();
    …
    try {
        …
        Failbox child = Failbox.getCurrent();
        enter (f) {
            … // allocate
            enter (child) {
                throw new RuntimeException();
            } catch (Throwable t) { }
            … // clean up
        }
        …
    } catch (Throwable t) {
        …
    }
    …
}
```

Other goodies

- In paper:
  - Separation logic proof rules
  - Implementation issues
- On website:
  - Prototype implementations as C#, Java libraries
  - Machine-checked soundness proof in Coq
  - Prototype program verifier

Related work

- Languages as operating systems (e.g. Luna, DrScheme)
- Async exceptions in Haskell
- .NET Framework 2.0 reliability features
- Killing applets [Rudys et al. 2001]
- Transactions
- Compensation stacks [Weimer et al. 2004]
- Erlang
- Eiffel SCOOP
- Class handlers [Dony 1990]

Failboxes:

- Minimal programming overhead
- Minimal reasoning overhead
- Minimal run-time overhead
- Compositional

Future work

- Assessing applicability
- Assessing usability
- Inferring enter blocks
- Application to async & callback patterns

Conclusion

- Purpose of exceptions: Dependency safety
- Conflicts with:
  - Non-exception-safe objects and:
  - Try-catch
  - Threads and locks
  - Thread.stop
  - Try-finally
- Solution: Failboxes!