Error Handling via the Happens-Before Relation

Nicholas D. Matsakis
Thomas R. Gross
ETH Zurich

Exception Handling

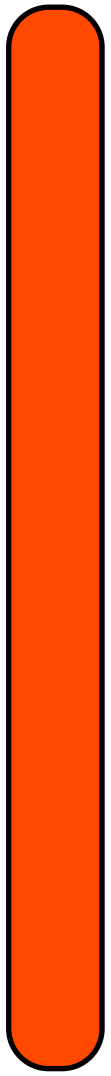
- Goal of this talk:
  - Extend exception handling into a parallel setting

Sequential Exceptions

```
void child()
{
  statement 1;

  statement 2;

  statement 3;
}
```

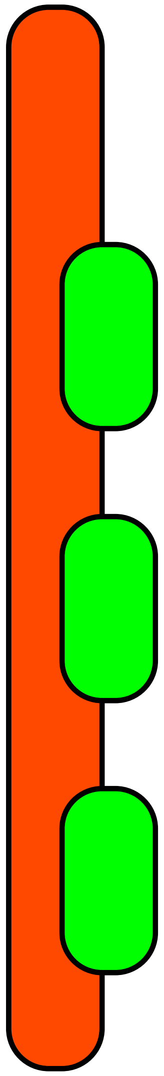
Sequential Exceptions

```
void child()
{
  statement 1;

  statement 2;

  statement 3;
}
```



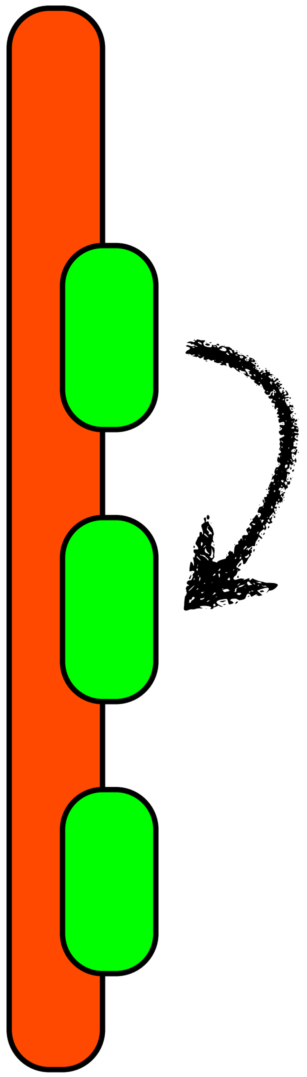
Sequential Exceptions

```
void child()
{
   statement 1;

   statement 2;

   statement 3;
}
```



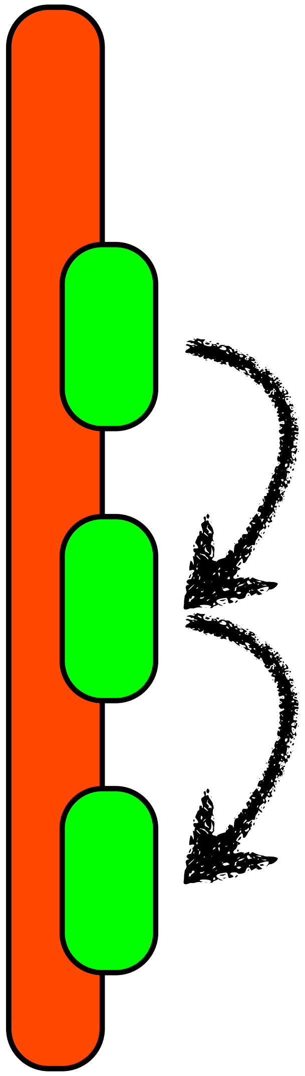
Sequential Exceptions

```
void child()
{
  statement 1;

  statement 2;

  statement 3;
}
```



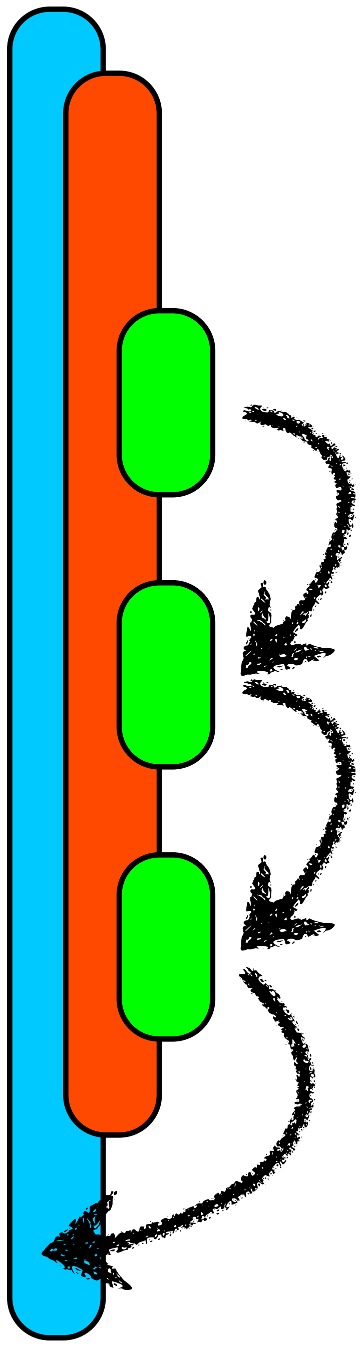
Sequential Exceptions

```
void child()
{
   statement 1;

   statement 2;

   statement 3;
}
```



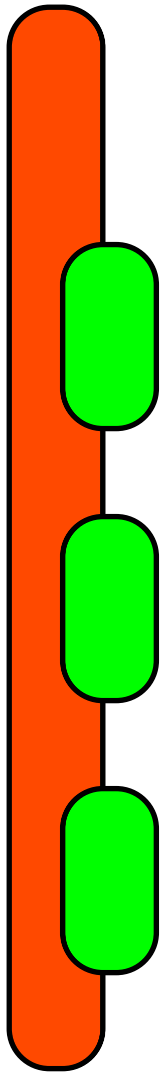
Sequential Exceptions

```
void child()
{
   statement 1;

   statement 2;

   statement 3;
}
```



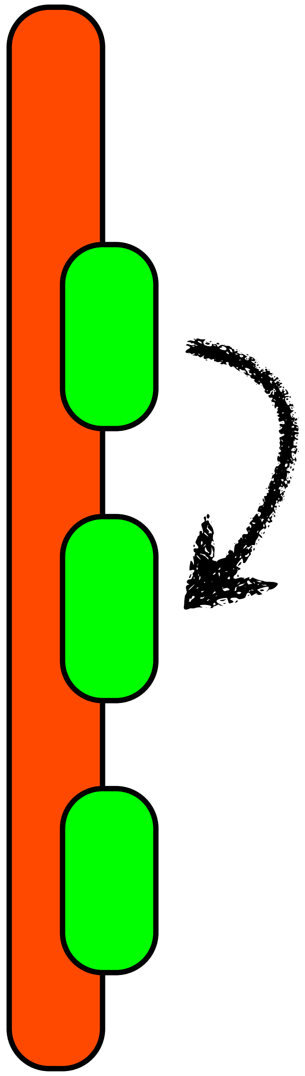
Sequential Exceptions

```
void child()
{
   statement 1;

   statement 2;

   statement 3;
}
```



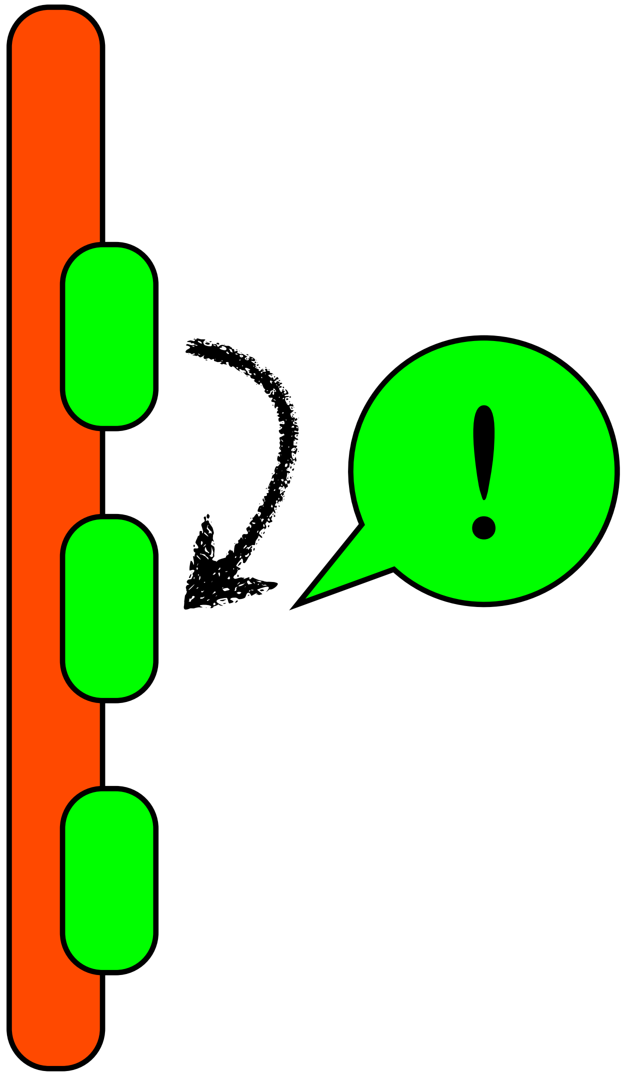
Sequential Exceptions

```
void child()
{
   statement 1;

   statement 2;

   statement 3;
}
```



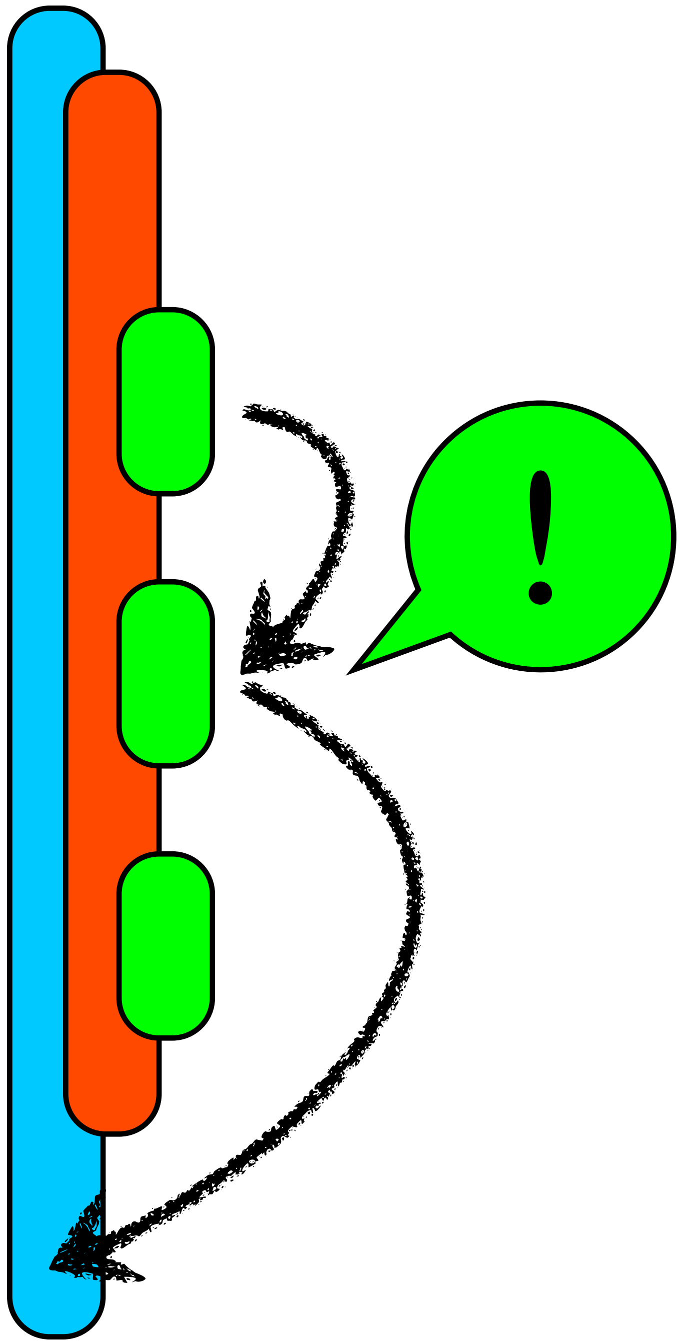
Sequential Exceptions

```
void child()
{
  statement 1;

  statement 2;

  statement 3;
}
```



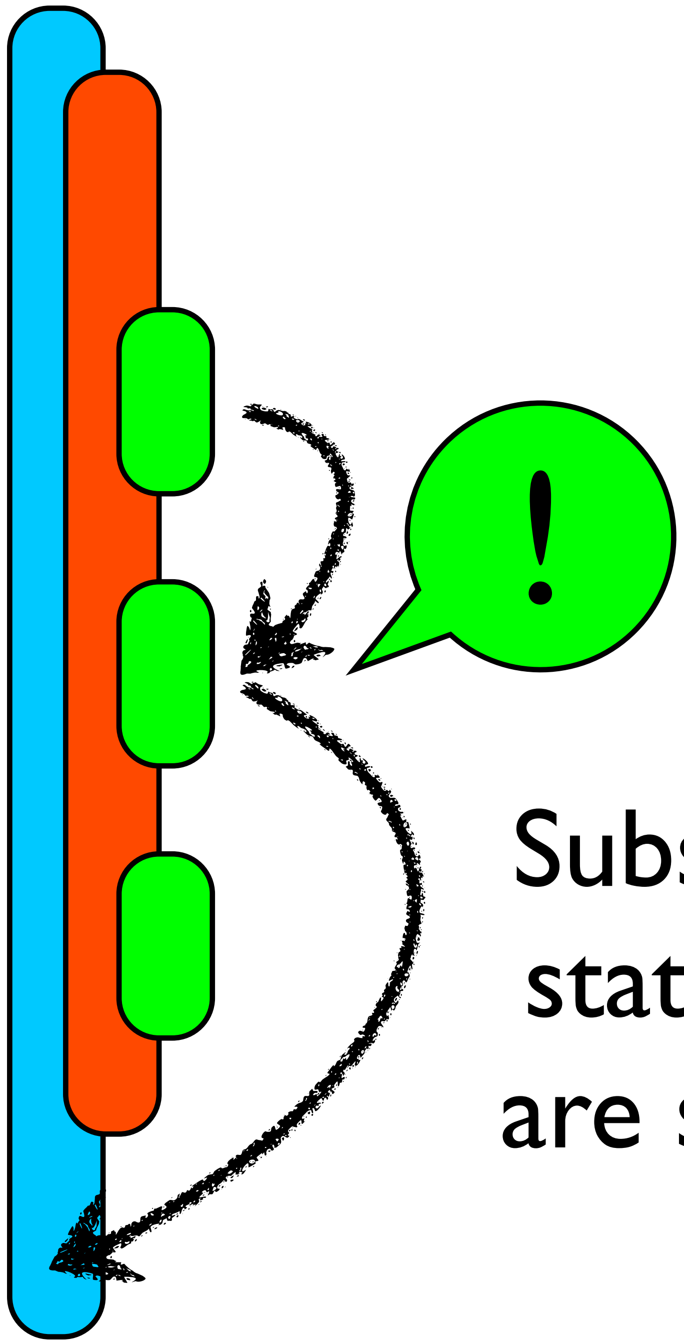
Sequential Exceptions

```
void child()
{
   statement 1;

   statement 2;

   statement 3;
}
```



Sequential Exceptions

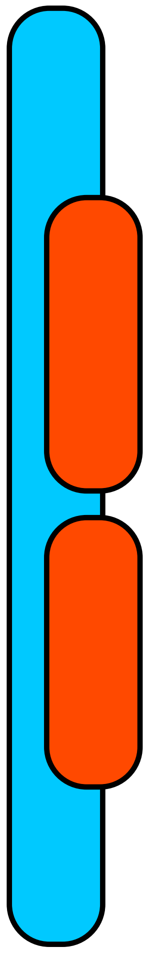
```
void child()
{
   statement 1;

   statement 2;

   statement 3;
}
```



Subsequent statements are skipped.

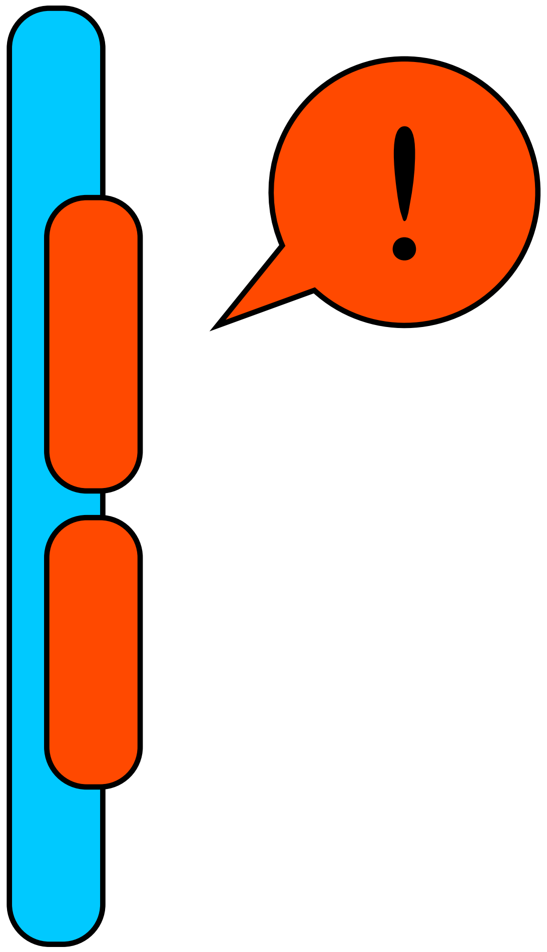
Sequential Exceptions

```
void parent()
{
  child();

  ...;
}
```



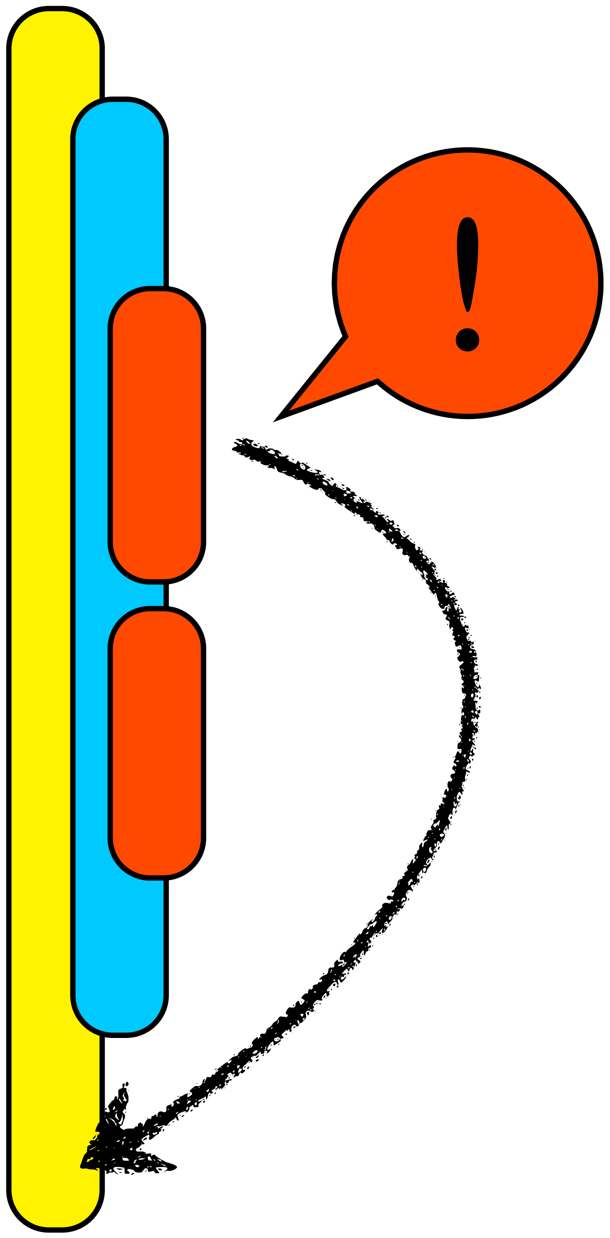
Sequential Exceptions

```
void parent()
{
  child();

  ...;
}
```



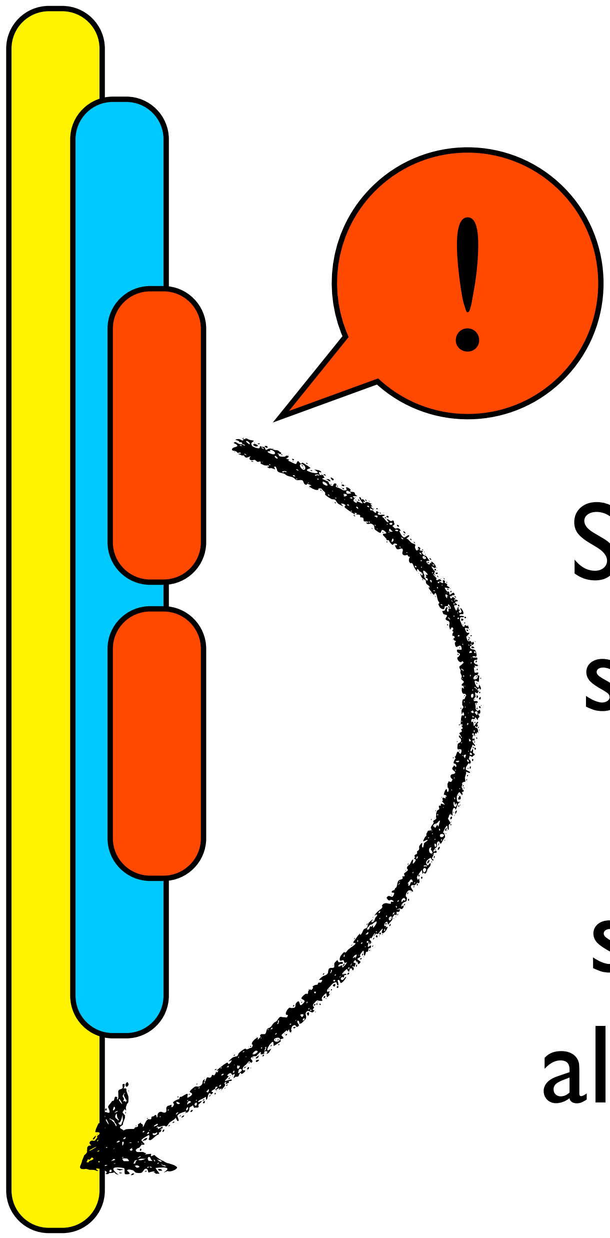
Sequential Exceptions

```
void parent()
{
  child();

  ...;
}
```



Sequential Exceptions

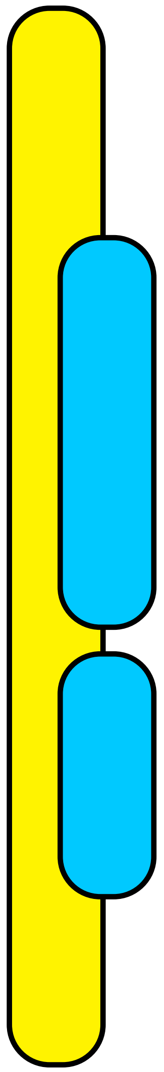
```
void parent()
{
  child();

  ...;
}
```



Subsequent statements in outer scopes are also skipped.

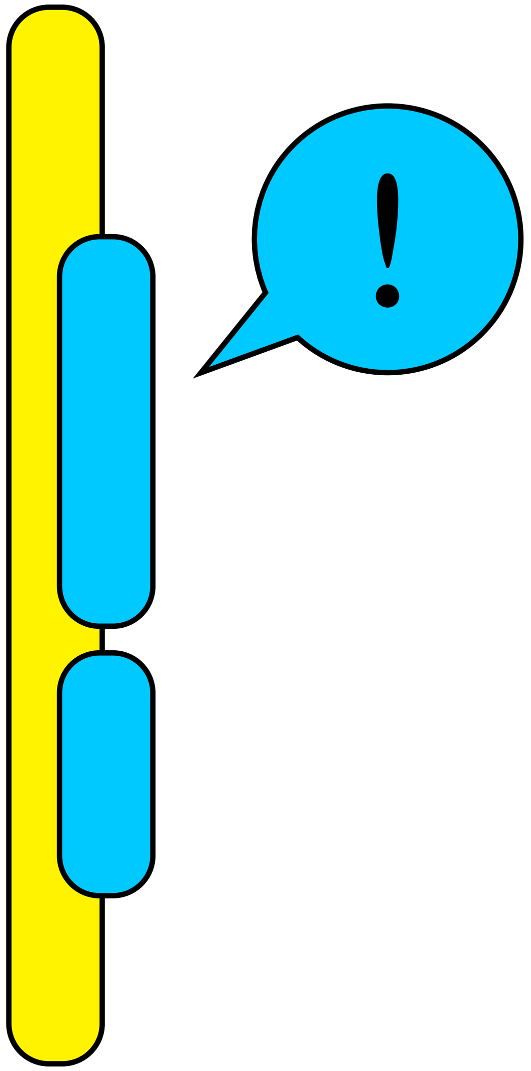
Sequential Exceptions

```
void grandparent()
{
  try {
    parent();
    ...
  }
  catch (...) {
  }
}
```



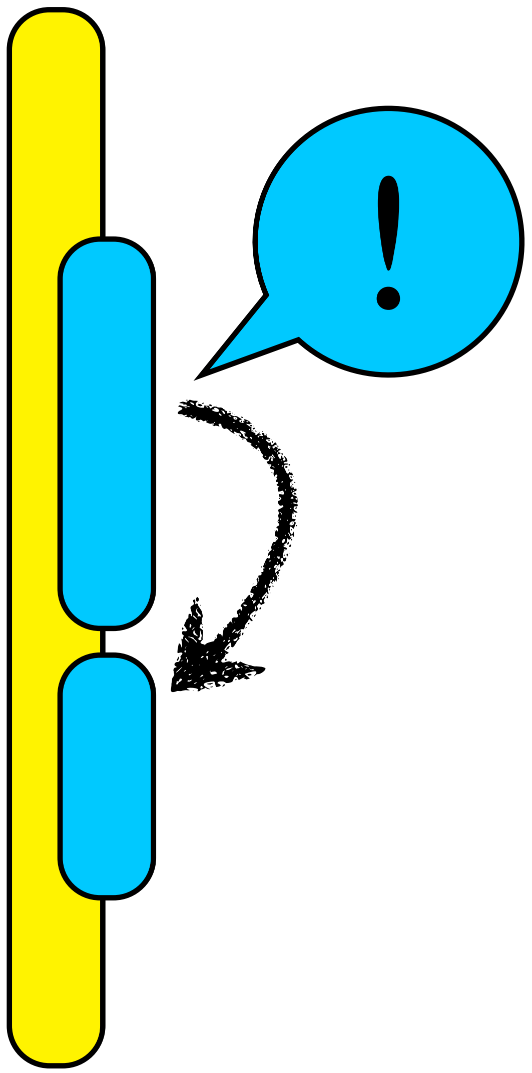
Sequential Exceptions

```
void grandparent()
{
  try {
    parent();
    ...
  }
  catch (...) {
  }
}
```



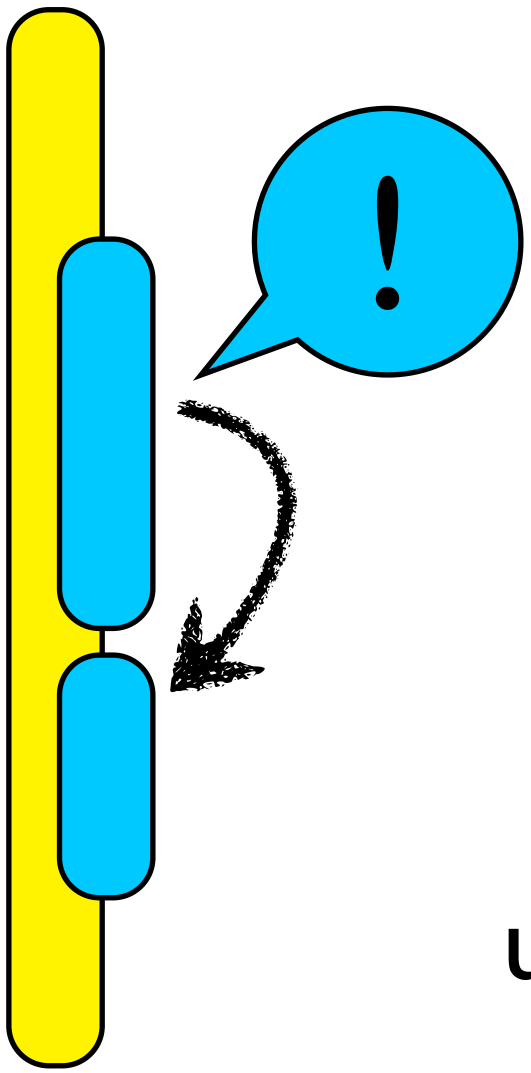
Sequential Exceptions

```
void grandparent()
{
  try {
    parent();
    ...
  }
  catch (...) {
  }
}
```



Sequential Exceptions

```
void grandparent()
{
  try {
    parent();
    ...
  }
  catch (...) {
  }
}
```



Continues until caught.

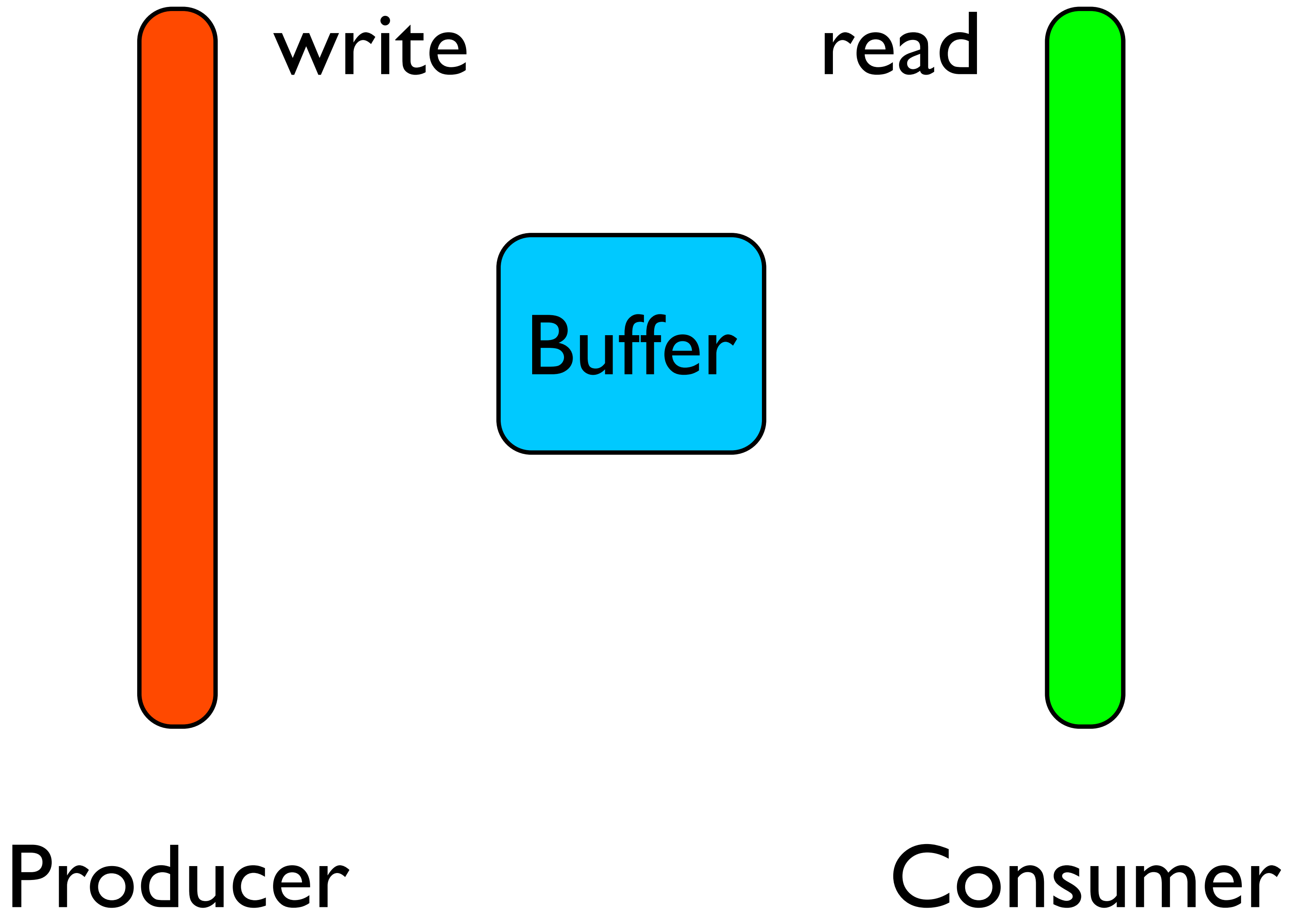
Key Points

- After an error occurs:
  - Subsequent statements are skipped
  - Until the error is caught in some enclosing scope

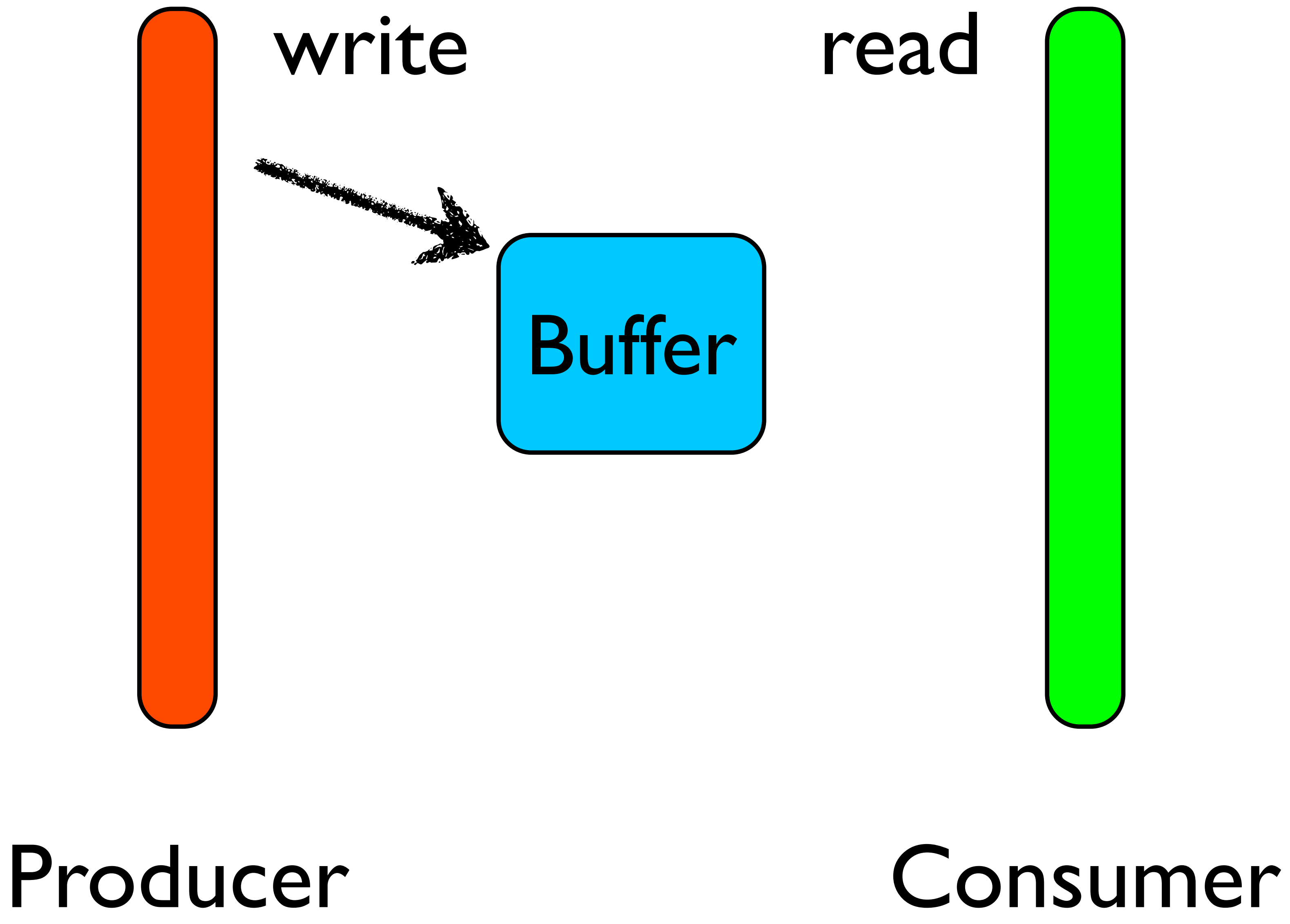
Exceptions and Threads

- In a sequential setting, subsequent statements are easy to identify.
- Not with threads.

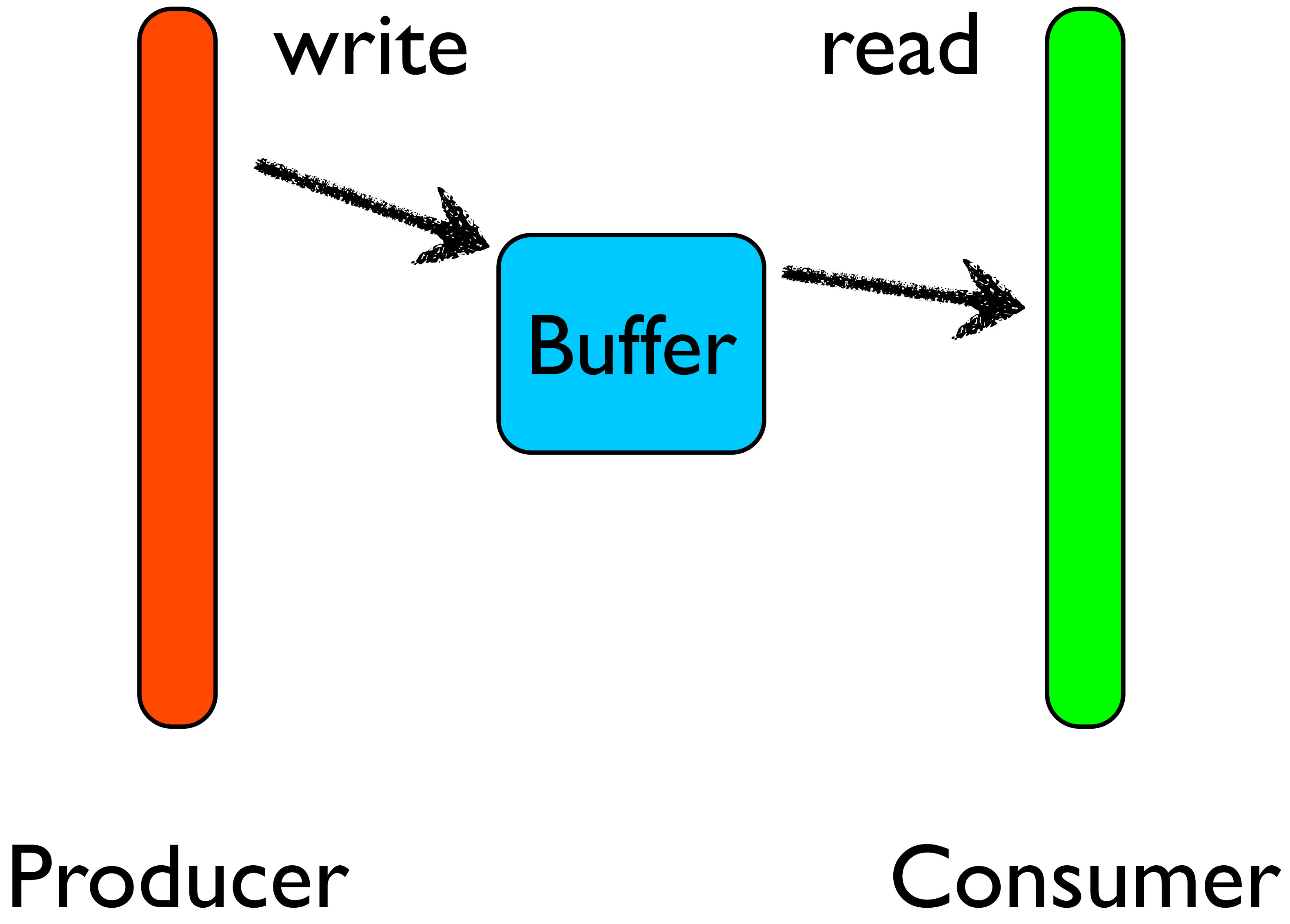
Threads



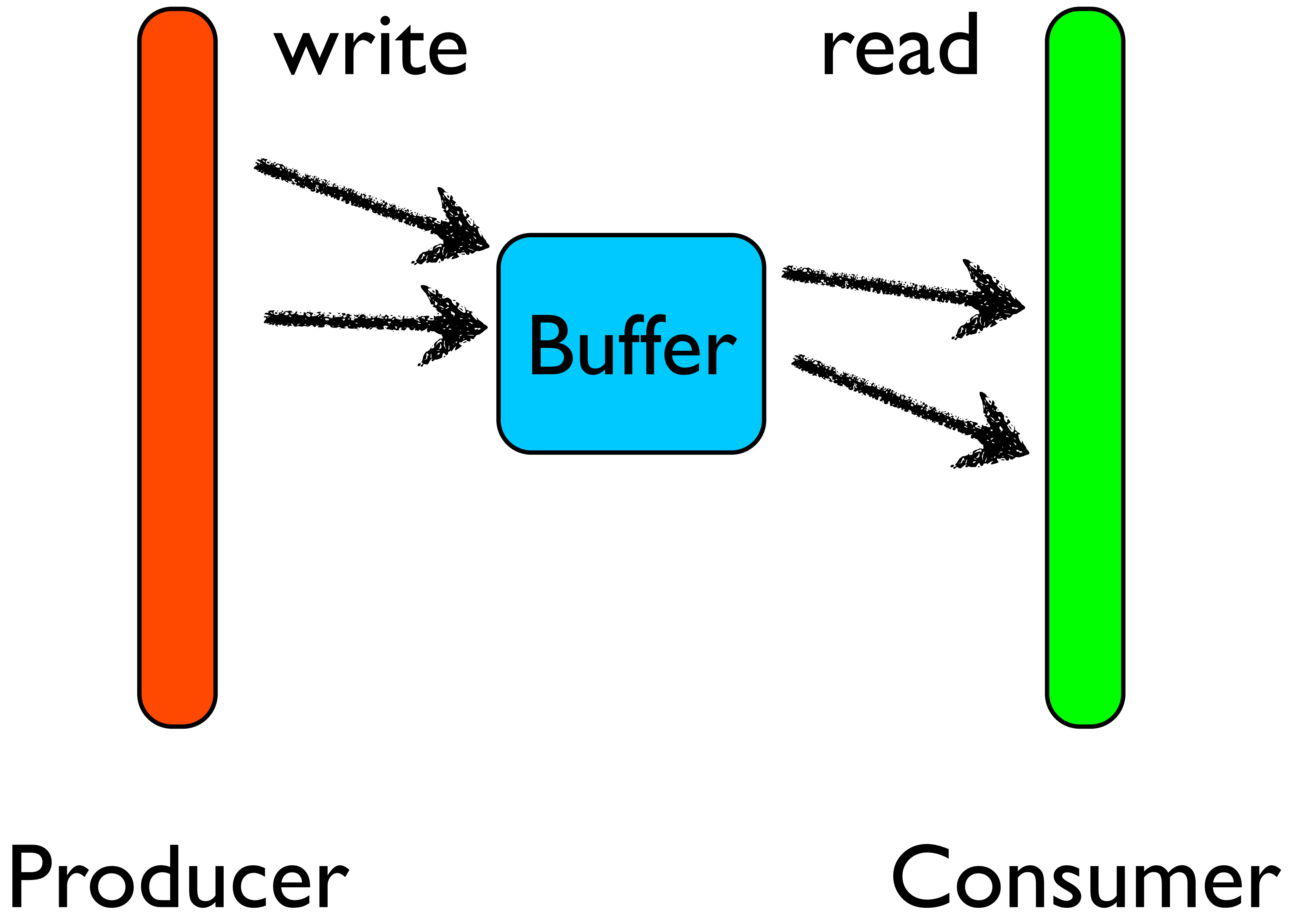
Threads



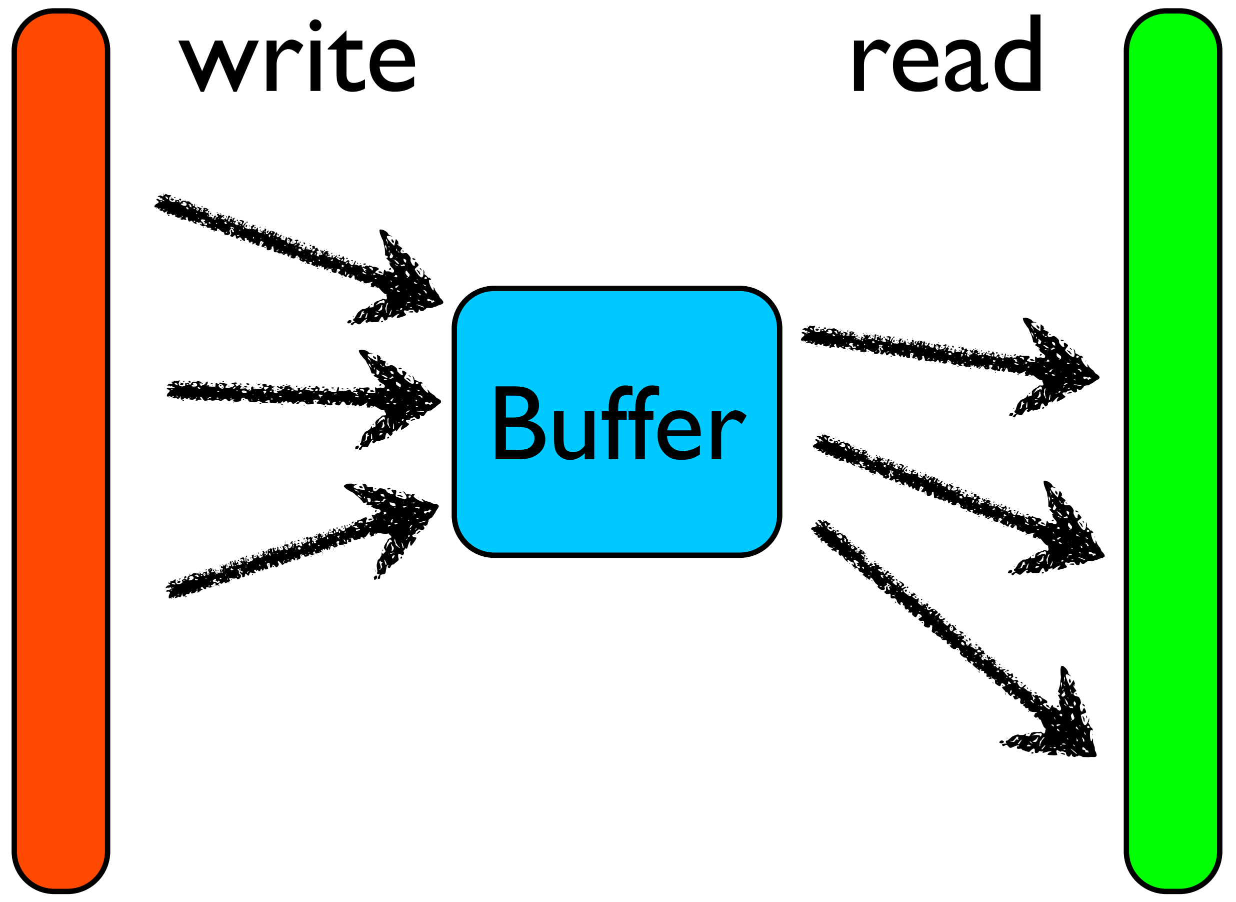
Threads



Threads

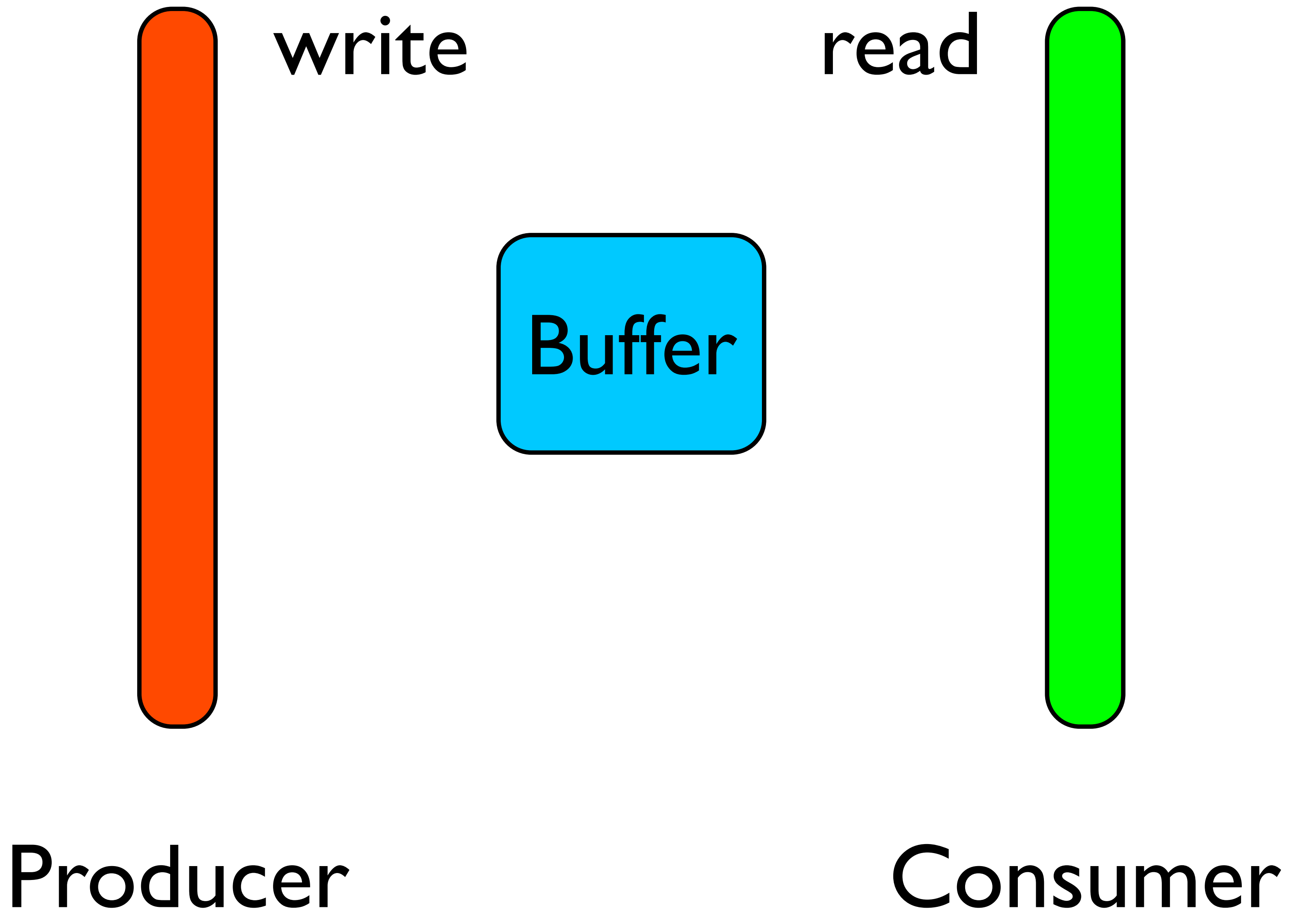


Threads

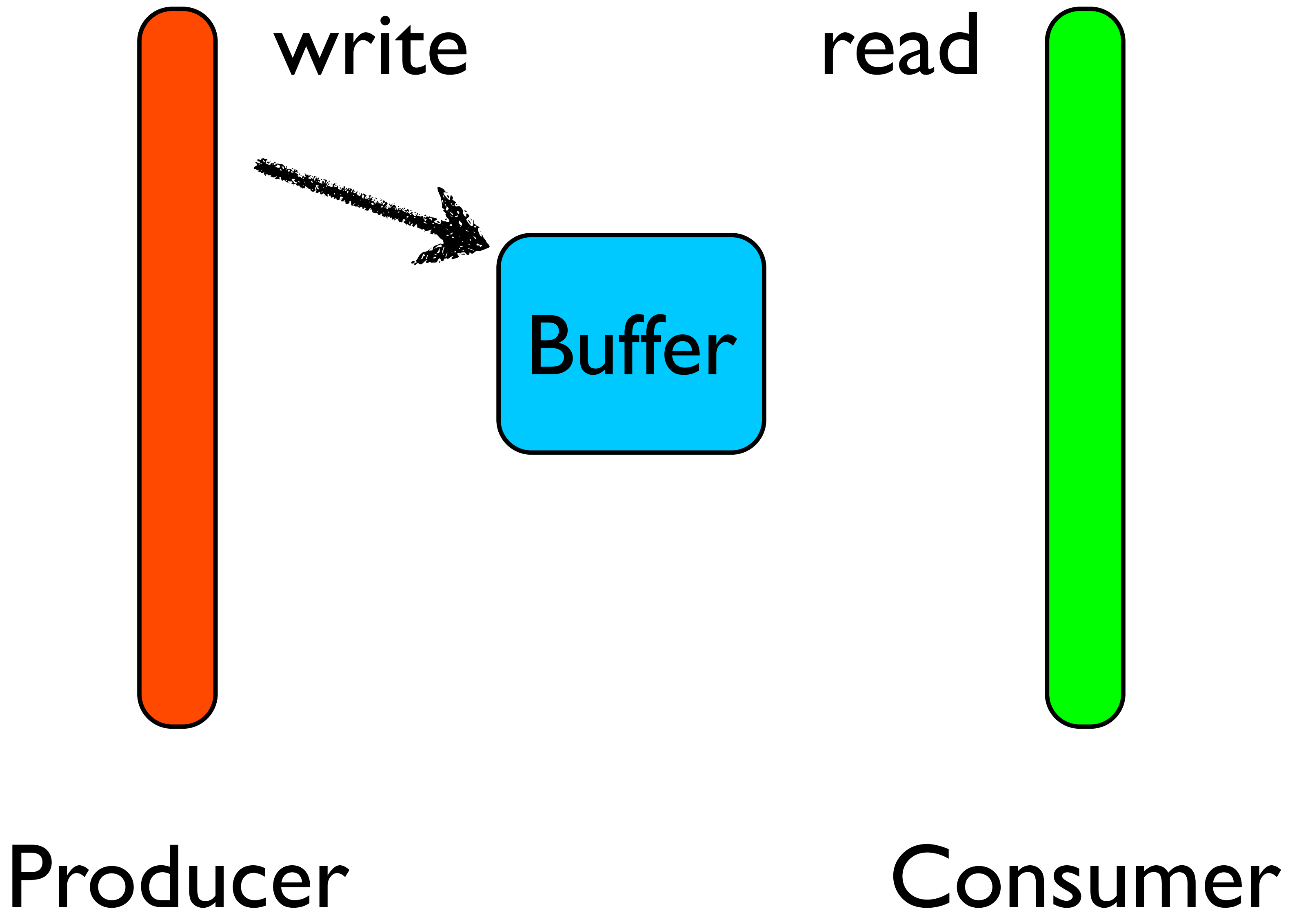


Producer Consumer

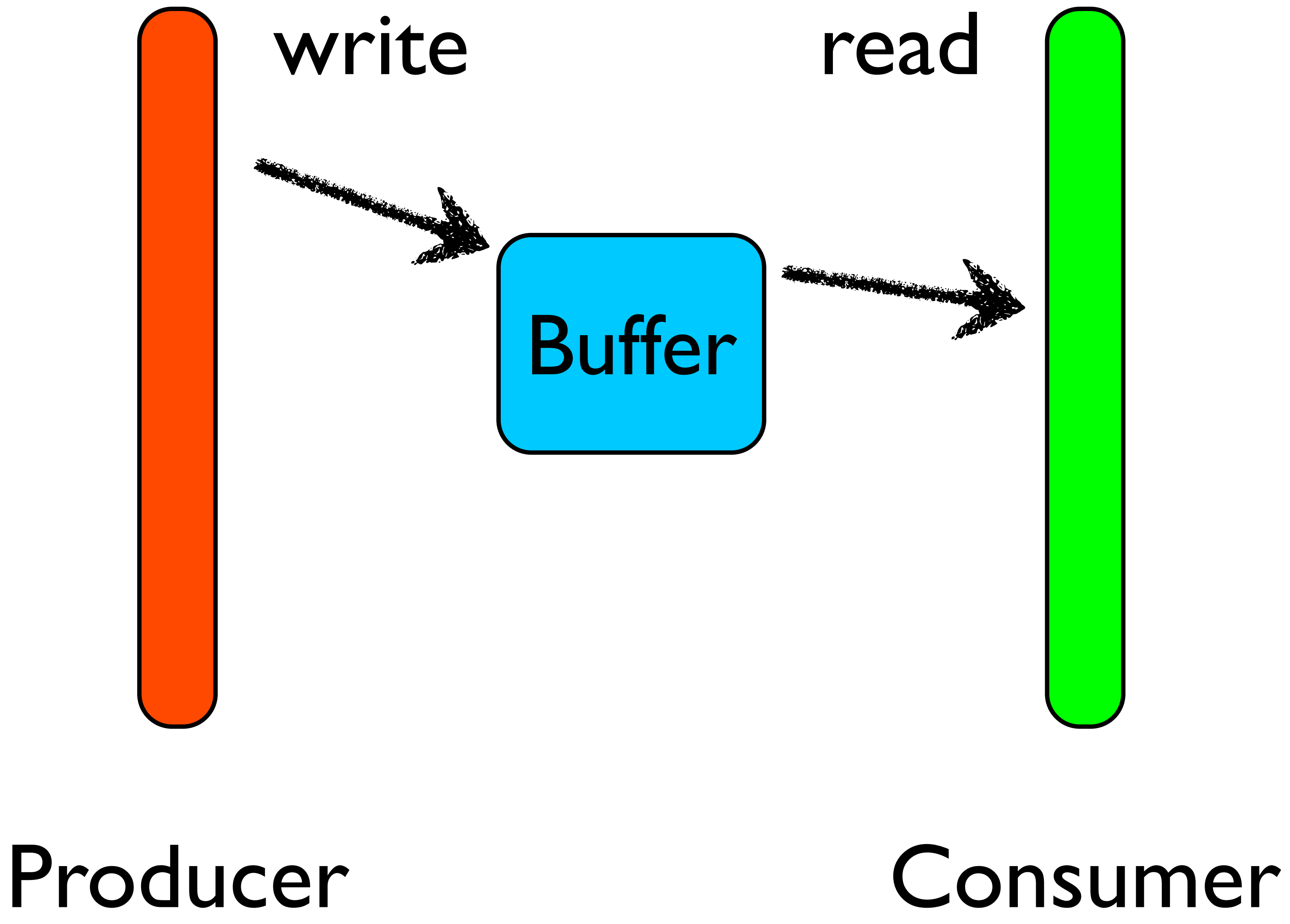
Threads



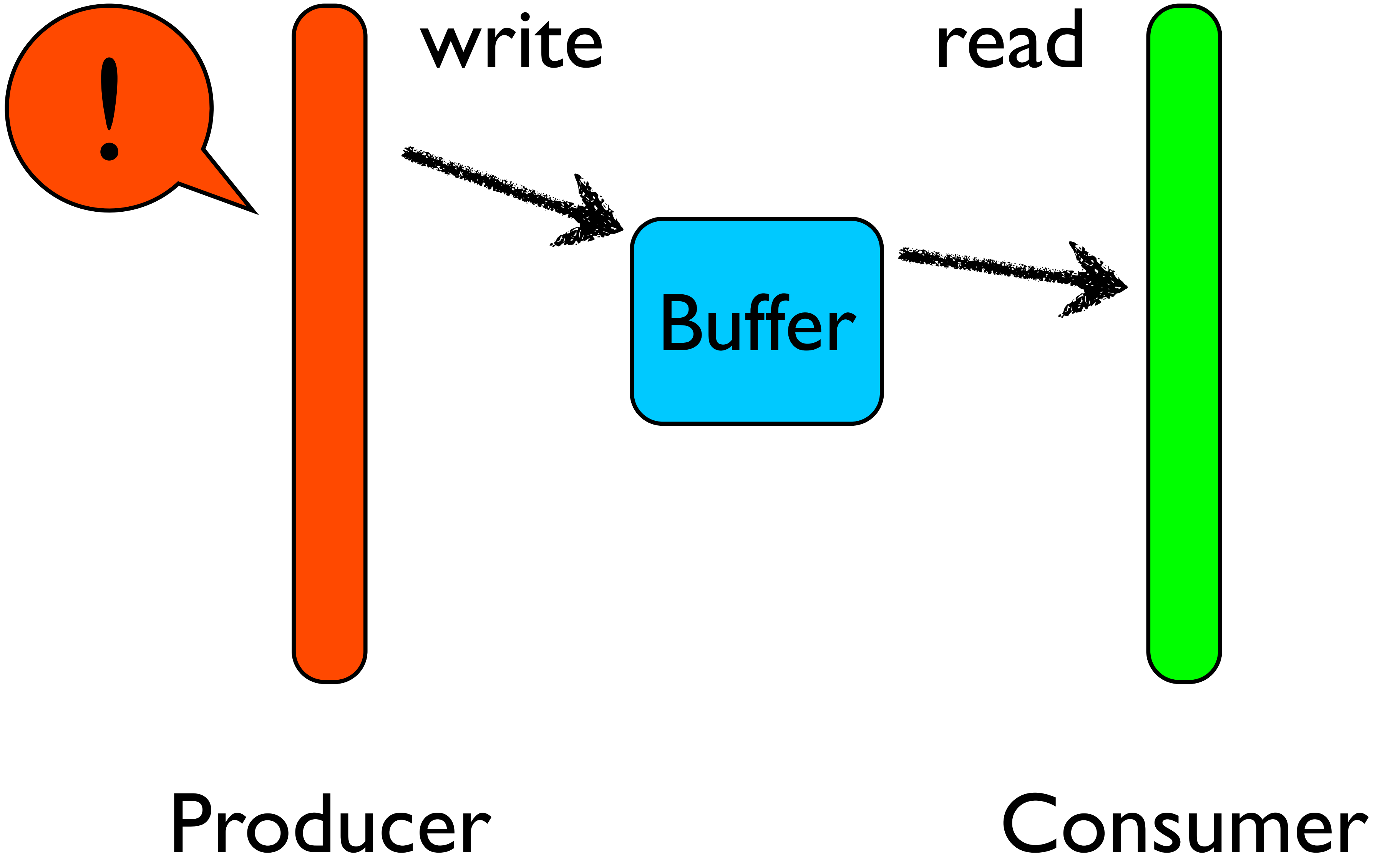
Threads



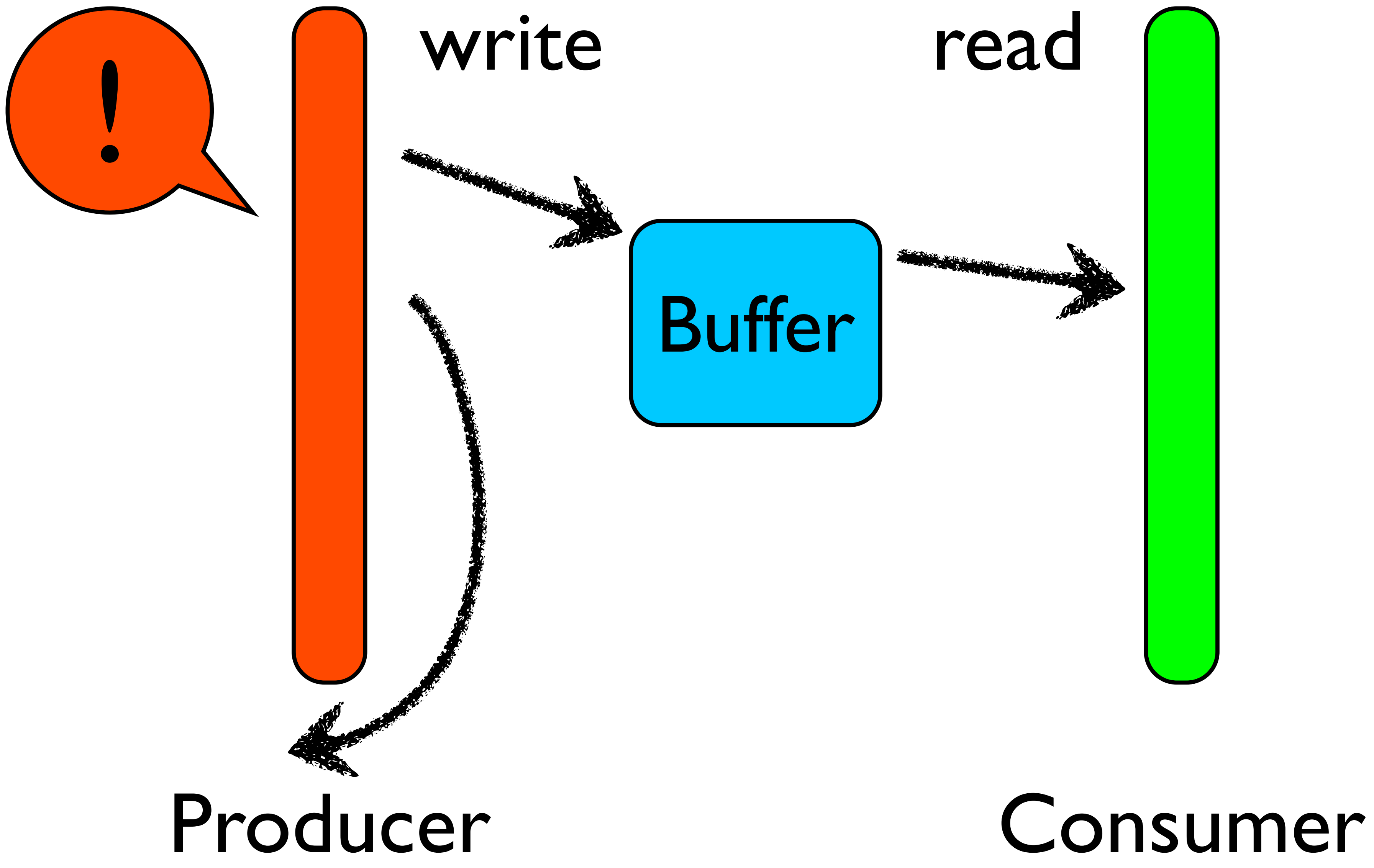
Threads



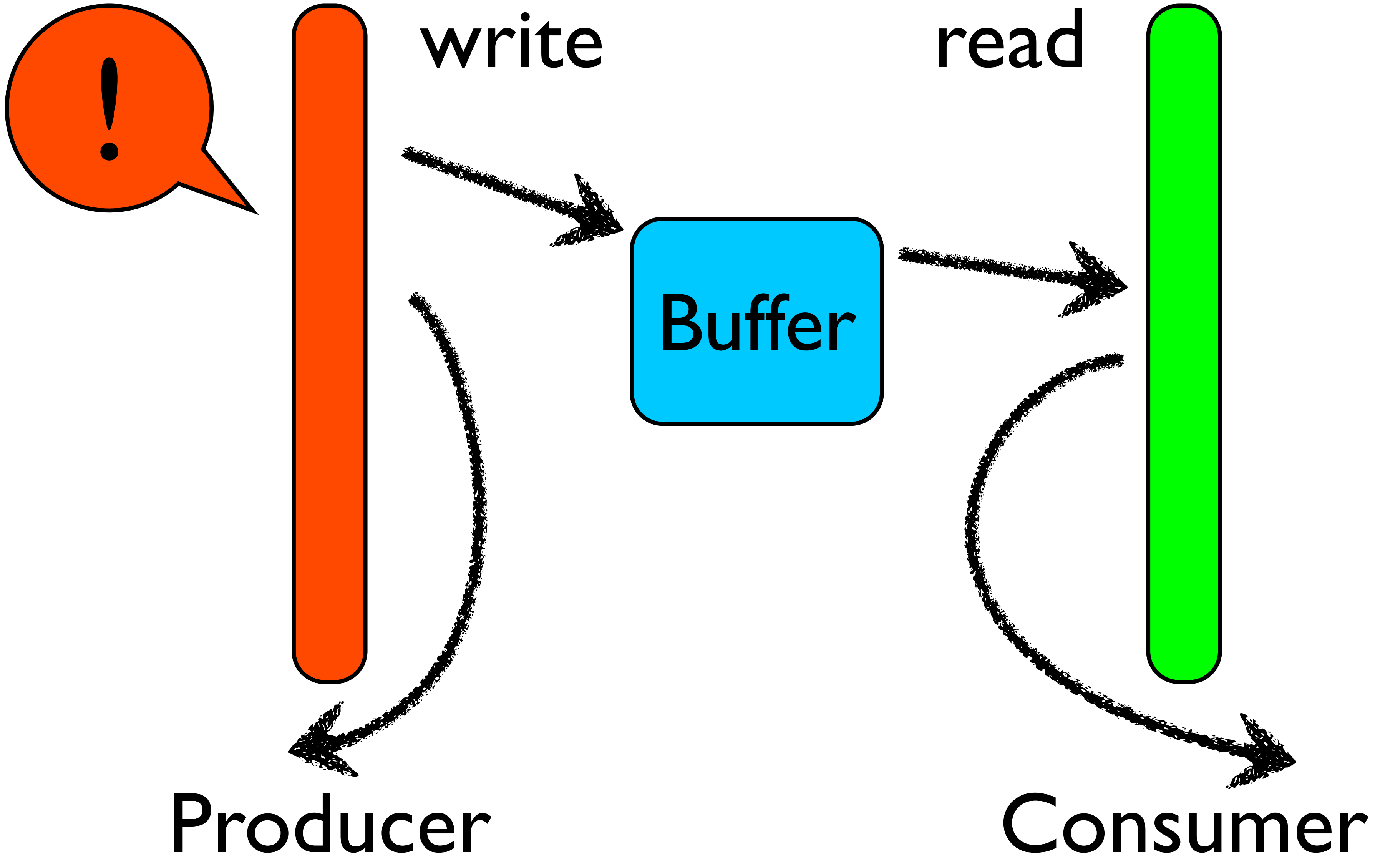
Threads



Threads



Threads



Invisible Ordering

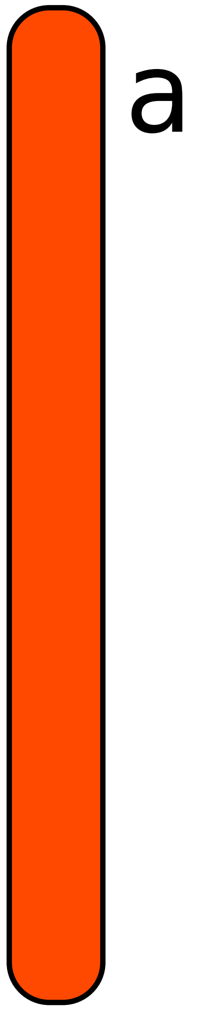
- Ordering between statements in different threads is never stated explicitly
- Created by side-effects
    - signals, locks

Intervals

- An alternative model for parallel programs
- Based on an explicit *happens-before* relation

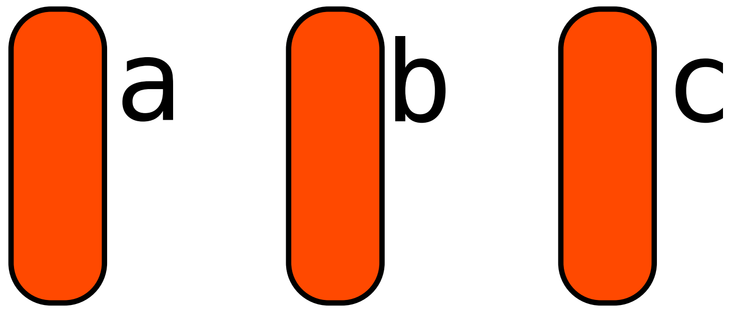
Interval

```
Interval a =
  interval {
    statement 1;
    statement 2;
  };
```



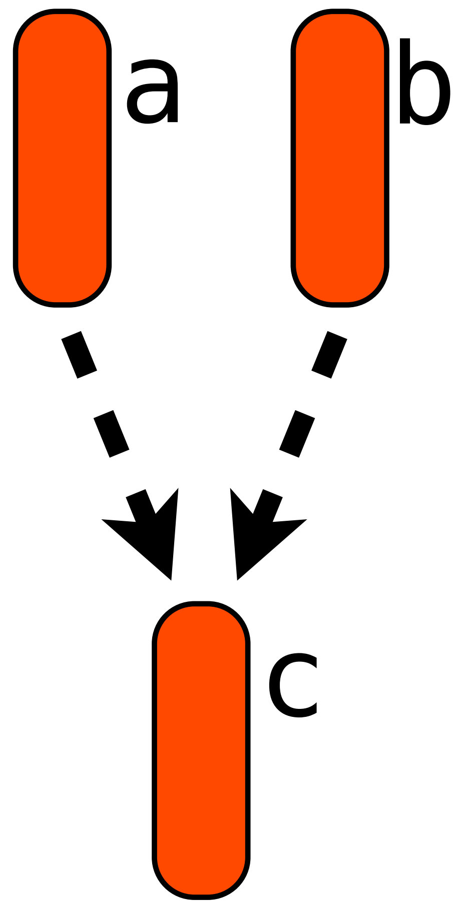
Interval

```
Interval a =
  interval {...};

Interval b =
  interval {...};

Interval c =
  interval {...};
```



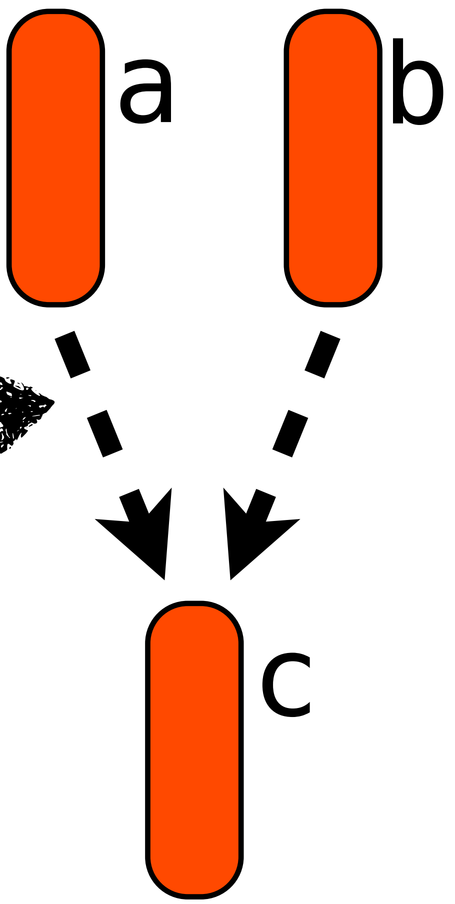
Happens Before

```
Interval a =
  interval {...};
Interval b =
  interval {...};
Interval c =
  interval {...};

a.end.addHb(c.start);
b.end.addHb(c.start);
```



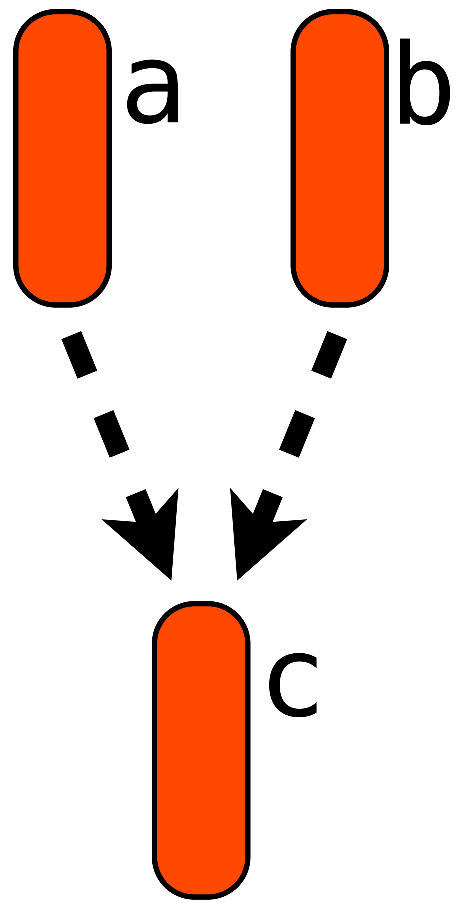
Happens Before

```
Interval a =
  interval {...};
Interval b =
  interval {...};
Interval c =
  interval {...};

a.end.addHb(c.start);
b.end.addHb(c.start);
```



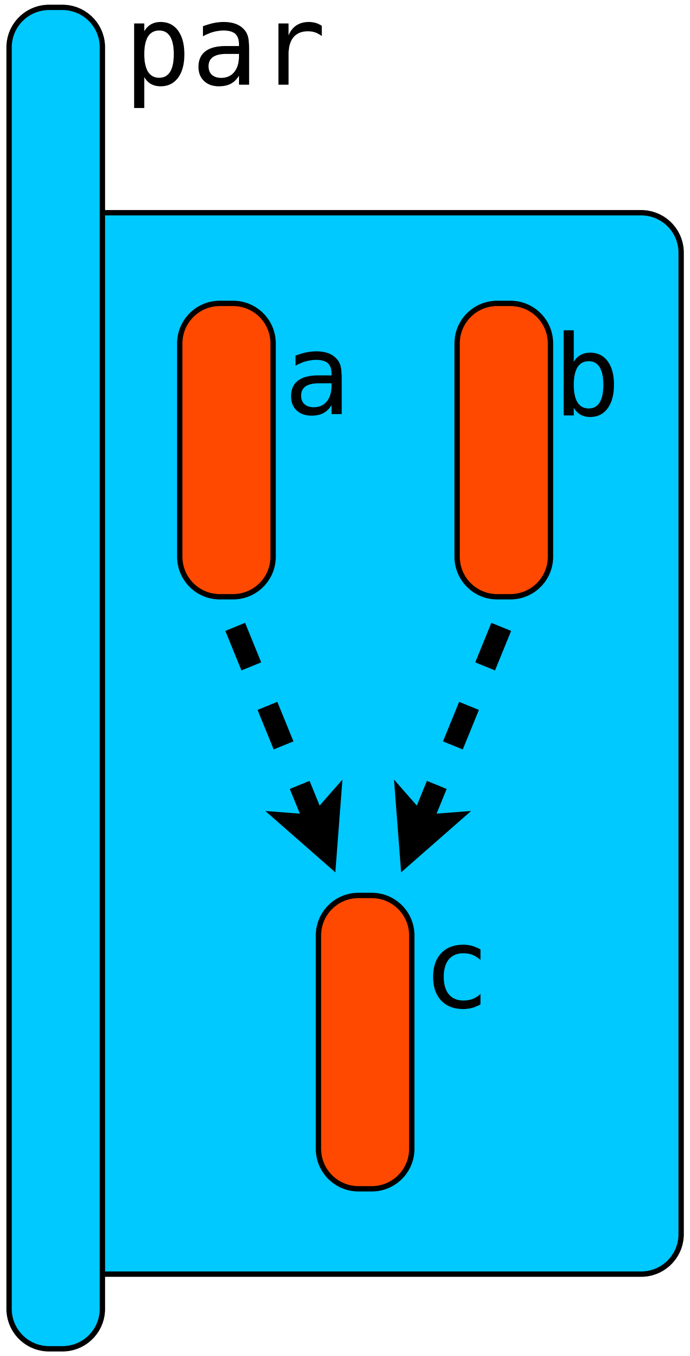
Happens Before

```
Interval a =
  interval {...};
Interval b =
  interval {...};
Interval c =
  interval {...};

a.end.addHb(c.start);
b.end.addHb(c.start);
```



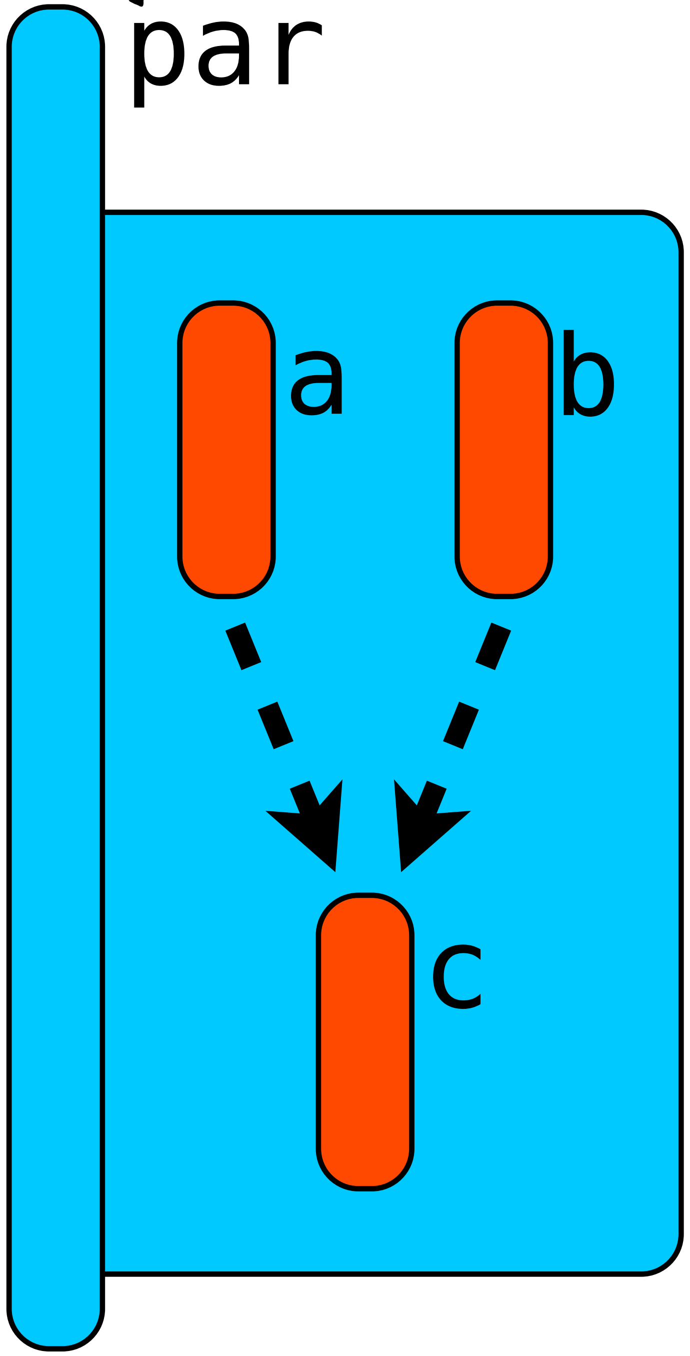
Hierarchy

```
void method(Interval par)
{
  a = interval(par) {...}
  b = interval(par) {...}
  c = interval(par) {...}
  a.end.addHb(c.start);
  b.end.addHb(c.start);
}
```



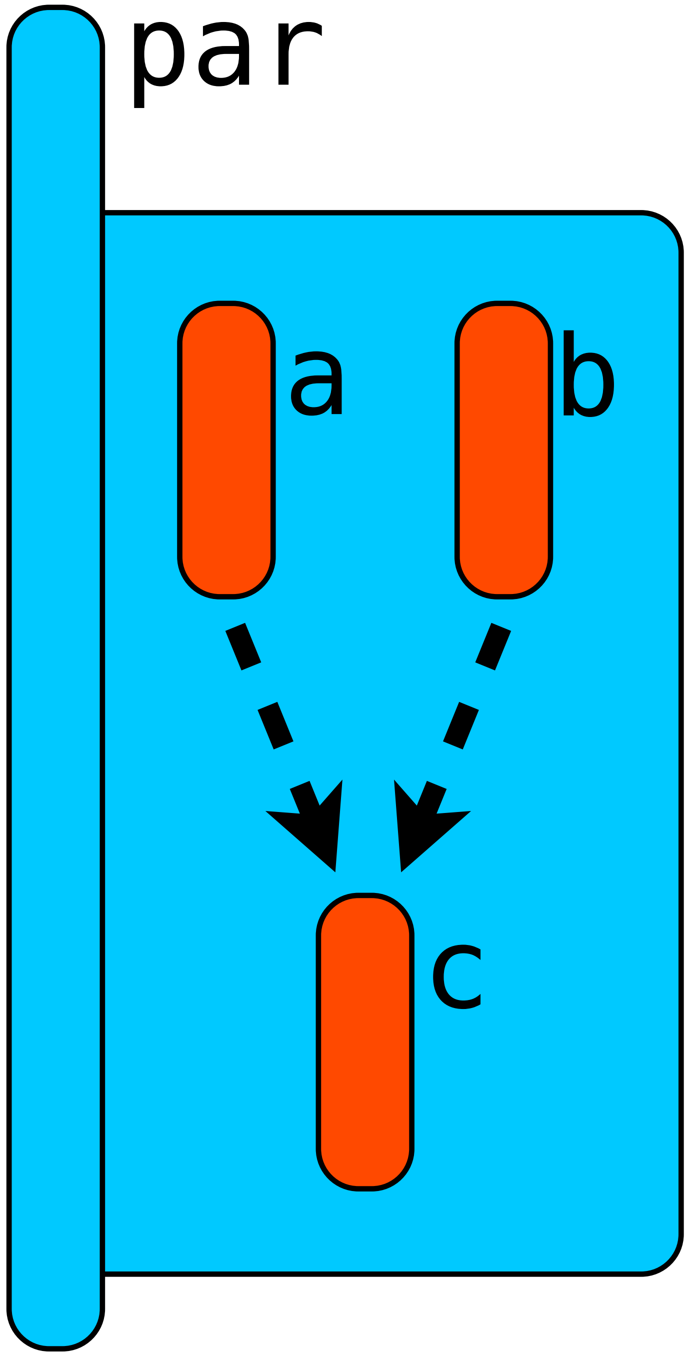
Hierarchy

```
void method(Interval par)
{
  a = interval(par) {...}
  b = interval(par) {...}
  c = interval(par) {...}
  a.end.addHb(c.start);
  b.end.addHb(c.start);
}
```



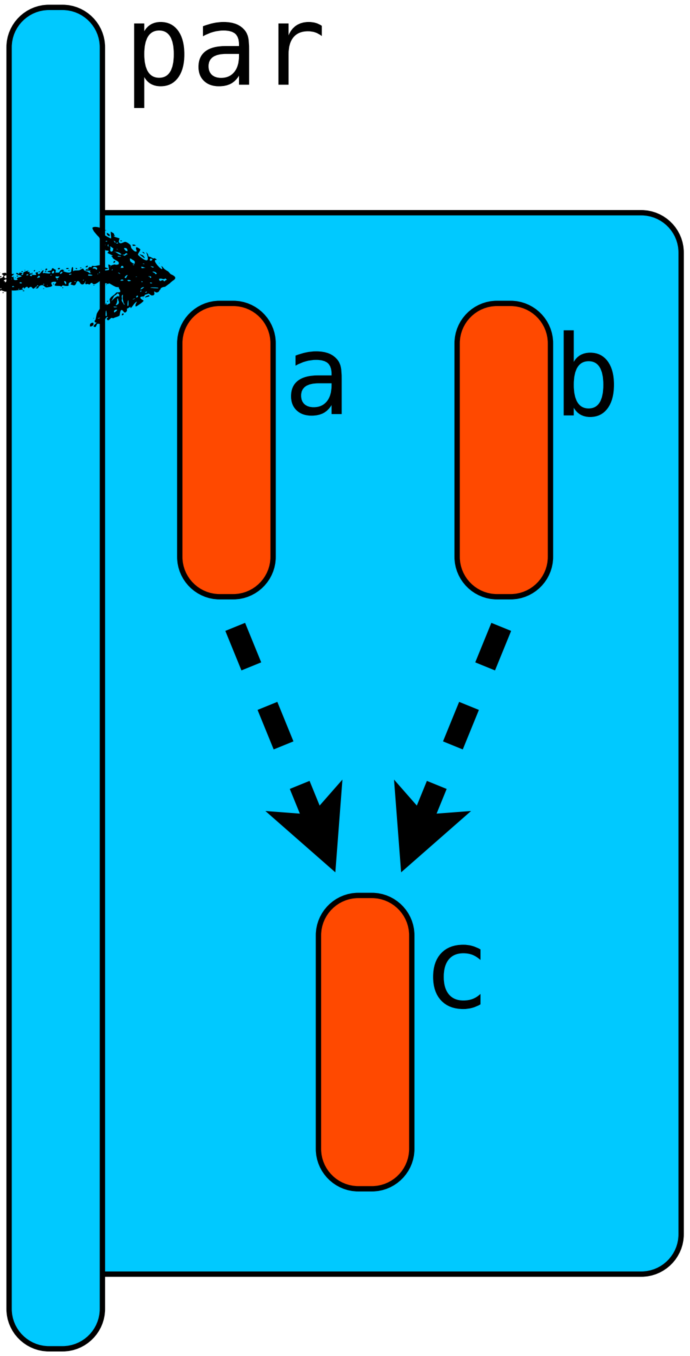
Hierarchy

```
void method(Interval par)
{
  a = interval(par) {...}
  b = interval(par) {...}
  c = interval(par) {...}
  a.end.addHb(c.start);
  b.end.addHb(c.start);
}
```



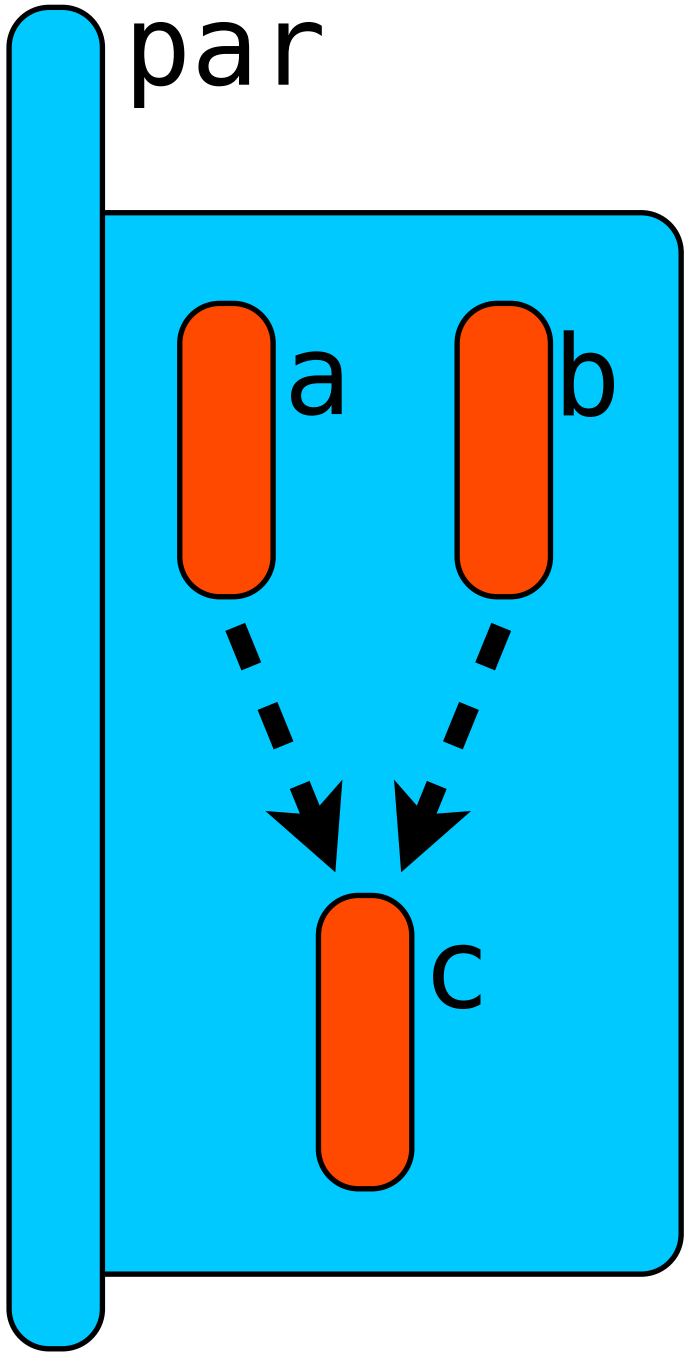
Hierarchy

```
void method(Interval par)
{
  a = interval(par) {...}
  b = interval(par) {...}
  c = interval(par) {...}
  a.end.addHb(c.start);
  b.end.addHb(c.start);
}
```



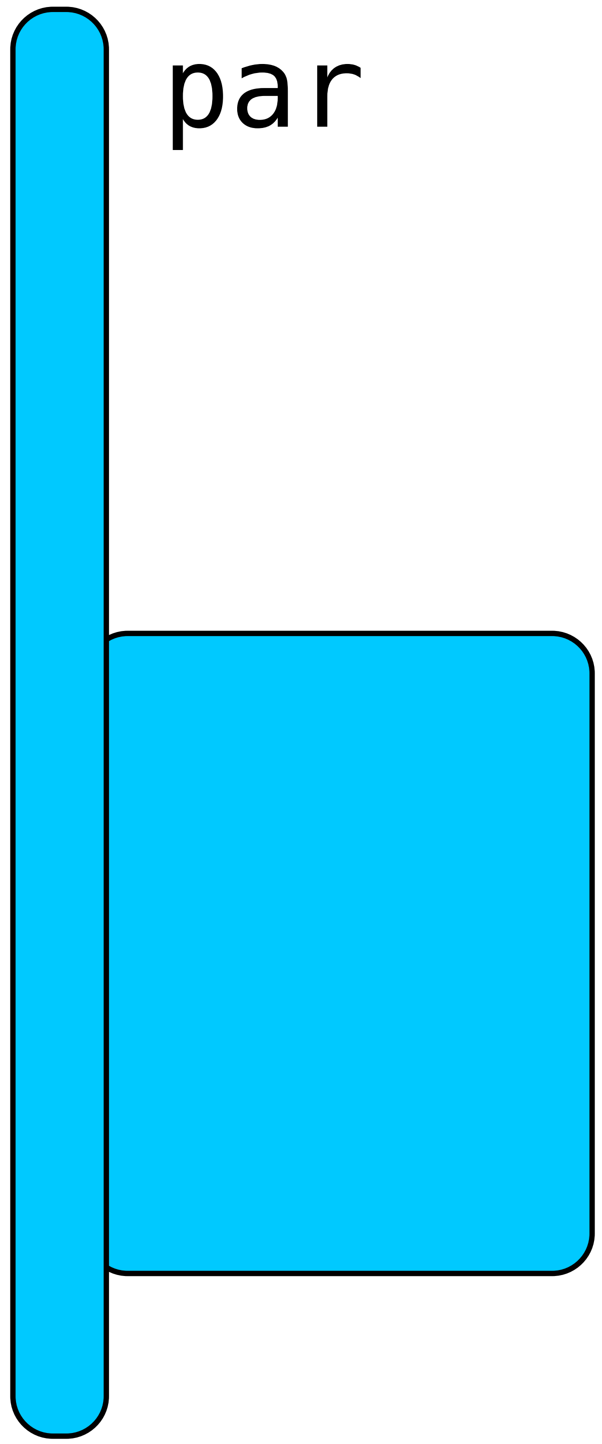
Hierarchy

```
void method(Interval par)
{
  a = interval(par) {...}
  b = interval(par) {...}
  c = interval(par) {...}
  a.end.addHb(c.start);
  b.end.addHb(c.start);
}
```



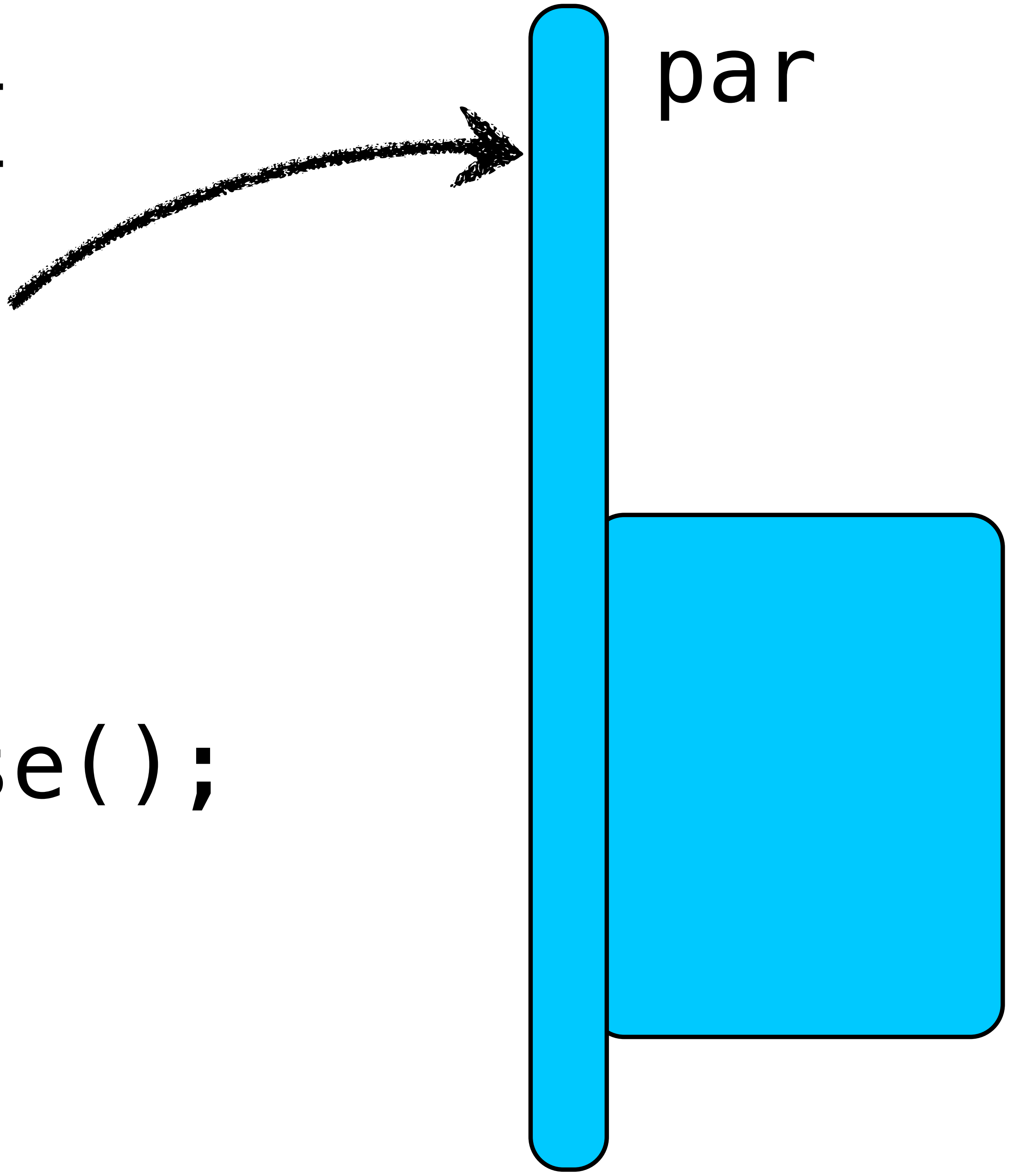
Two-Phase Execution

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



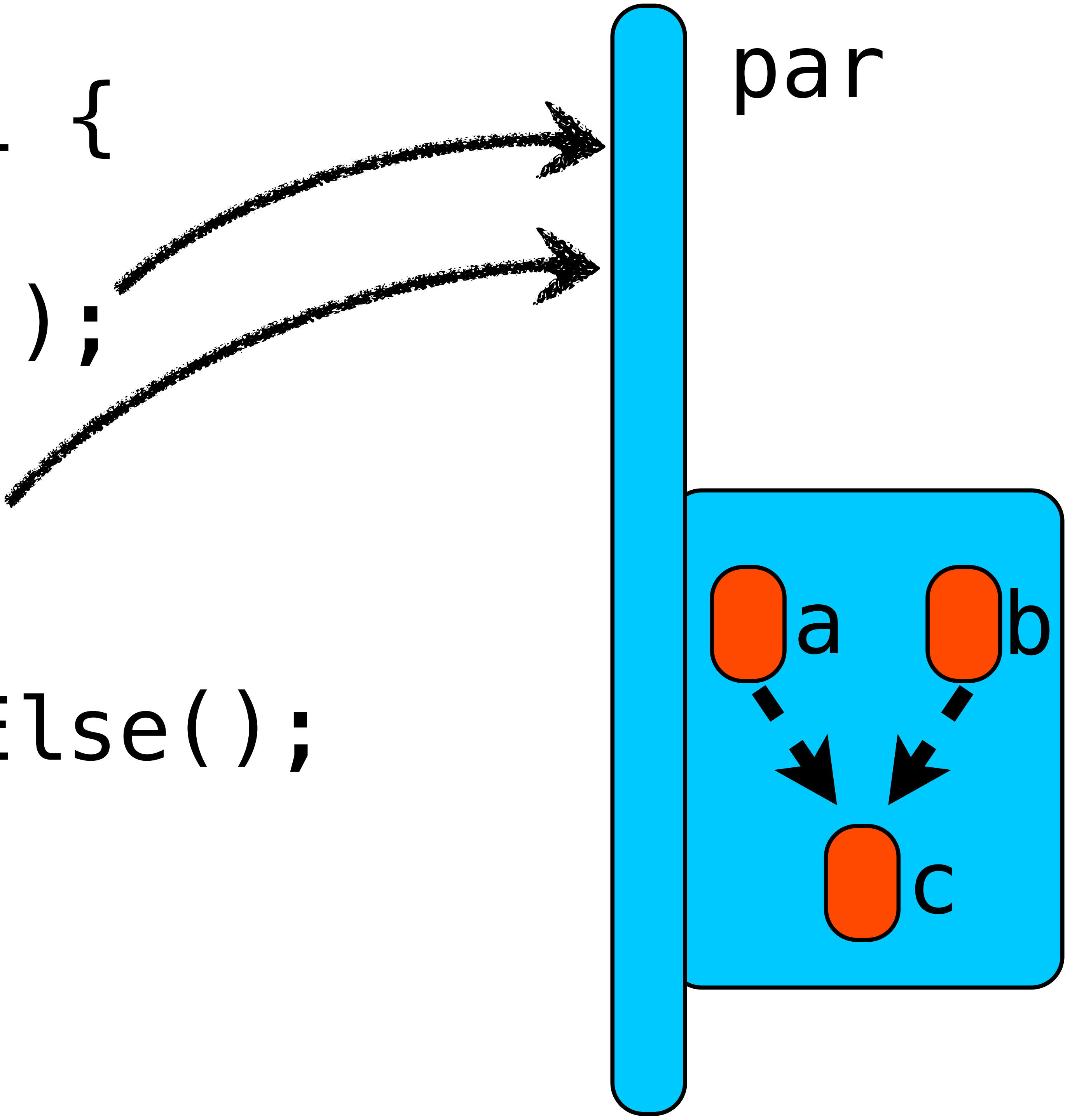
Two-Phase Execution

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



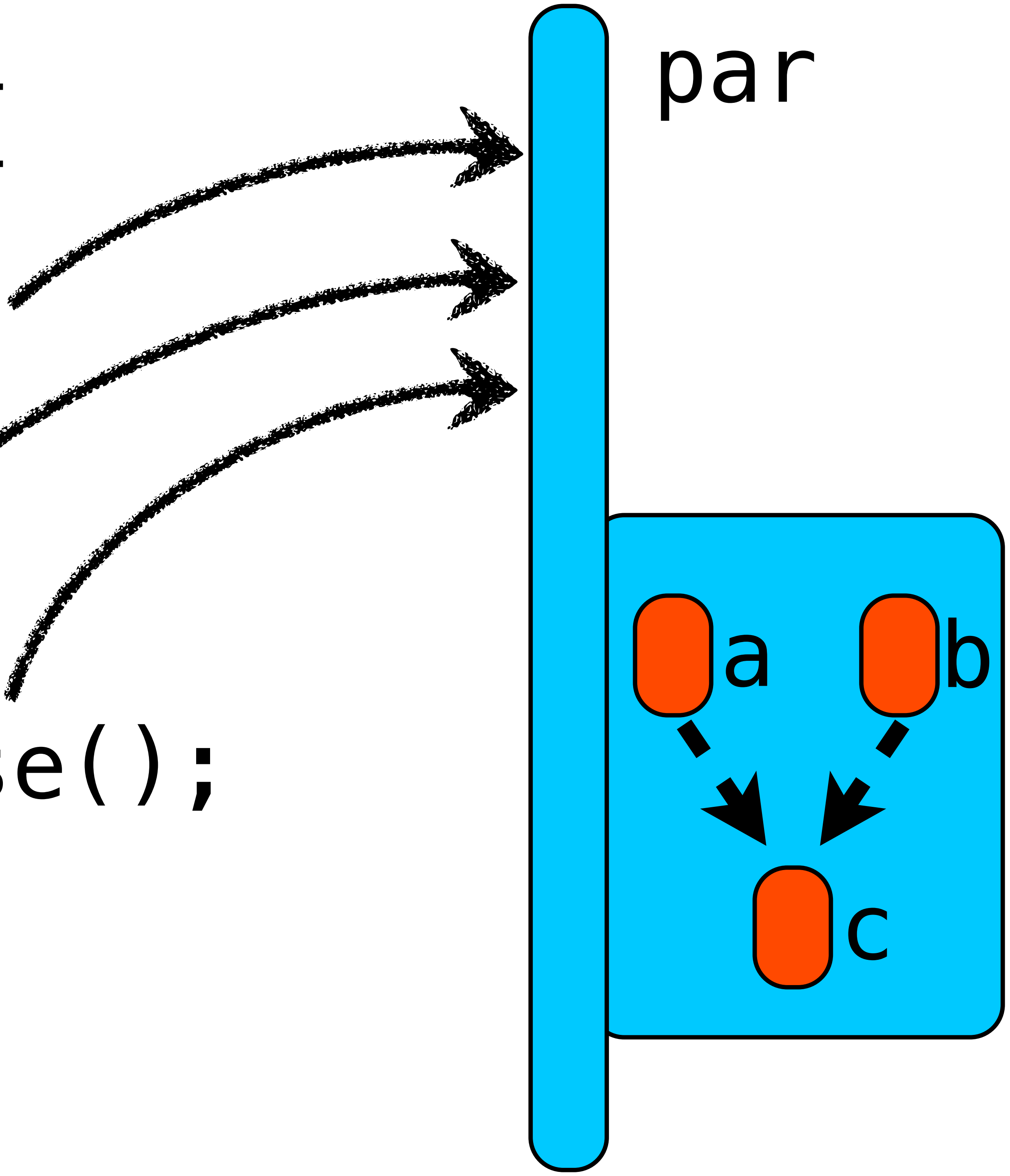
Two-Phase Execution

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



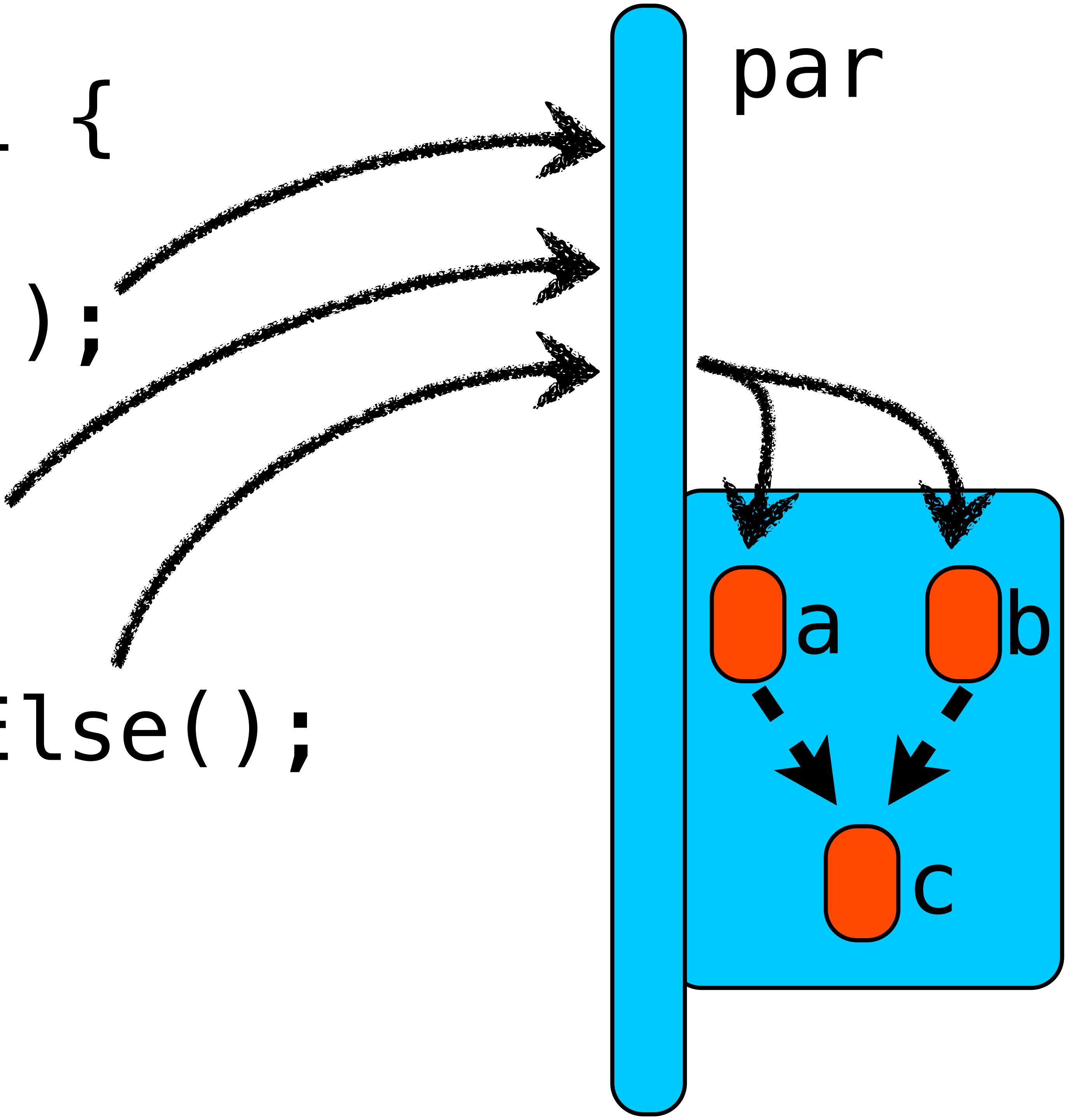
Two-Phase Execution

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



Two-Phase Execution

```
par = interval {

  doSomething();

  method(par);

  doSomethingElse();

}
```

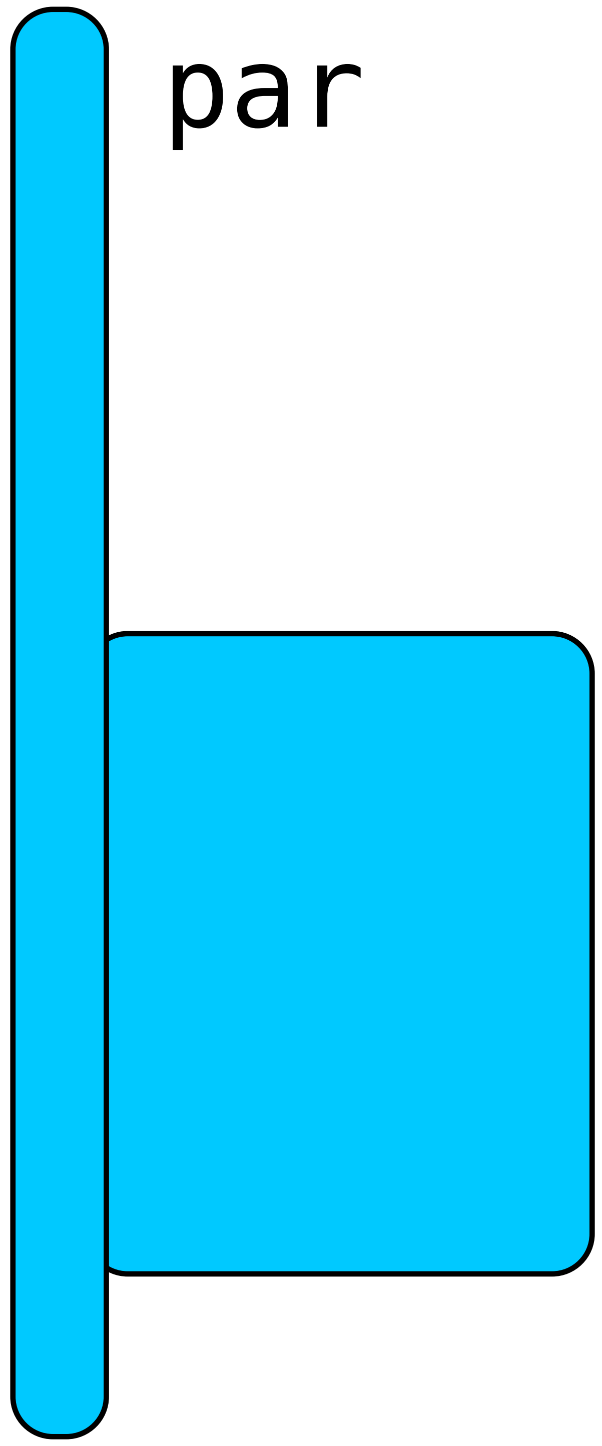


Error Handling

- If an exception is thrown at point P and caught at point Q:
  - Statements that happen after P and before Q are skipped.
- An interval may catch errors that occur in it or its children.

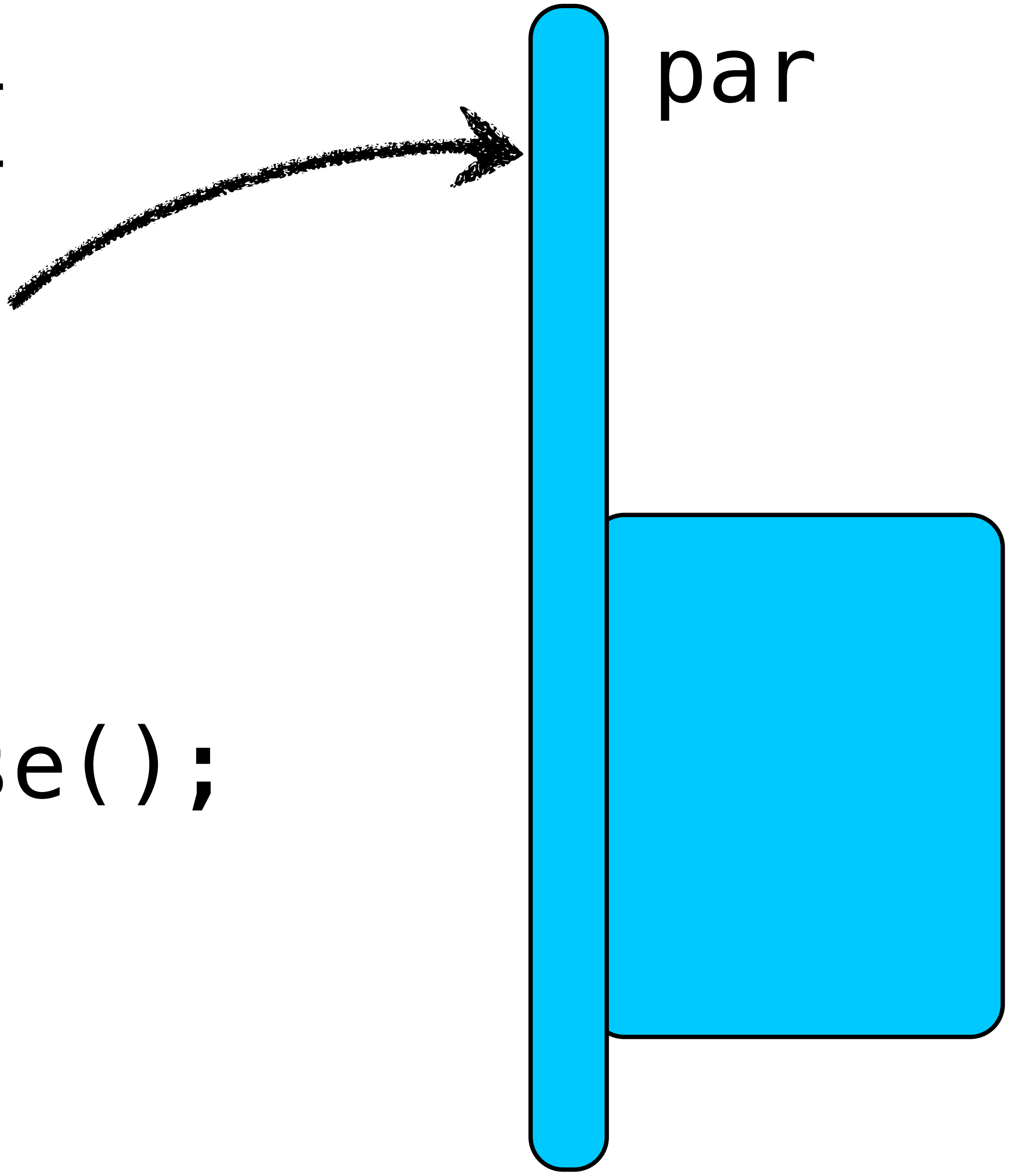
Everything Goes Well

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



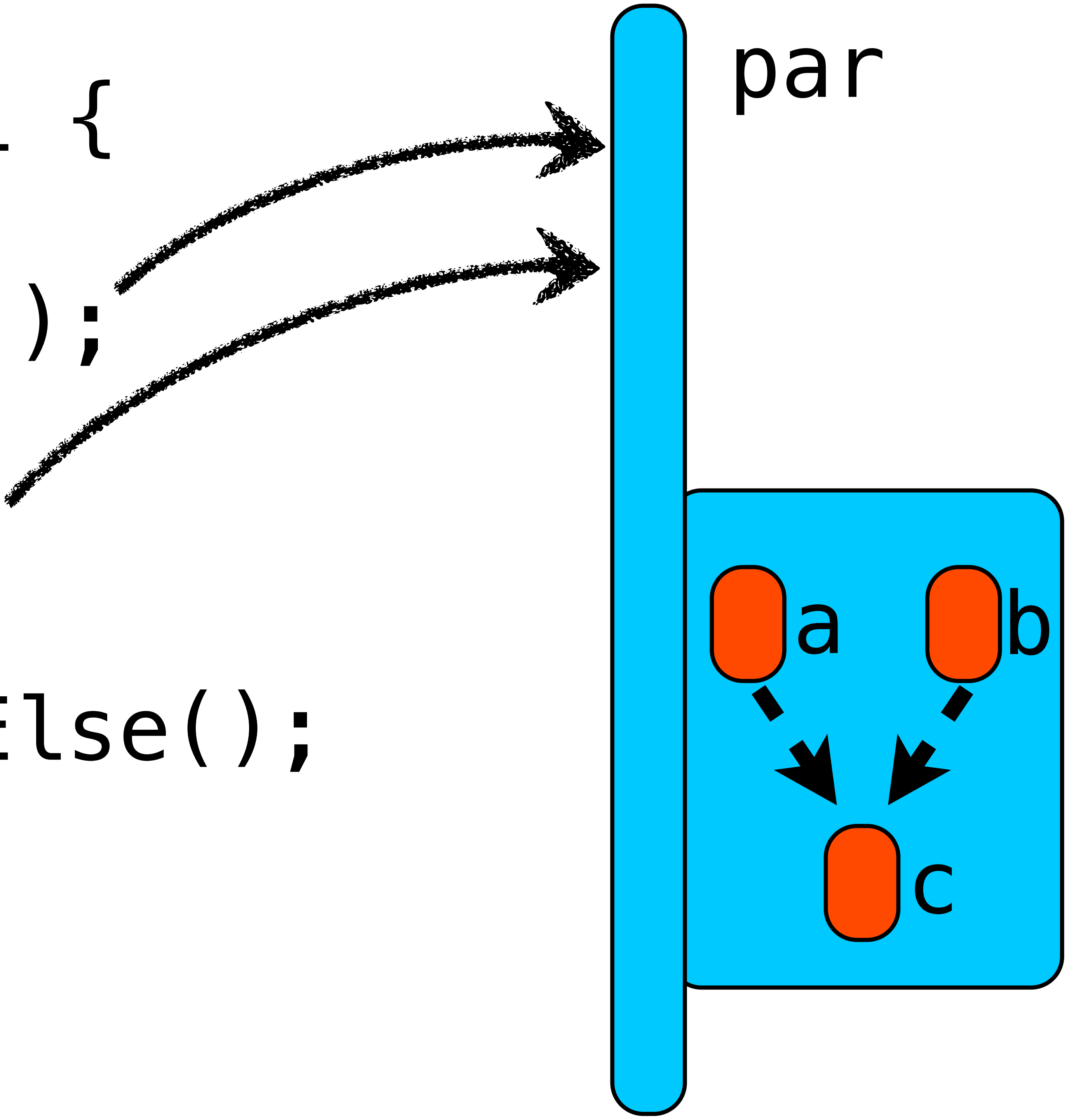
Everything Goes Well

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



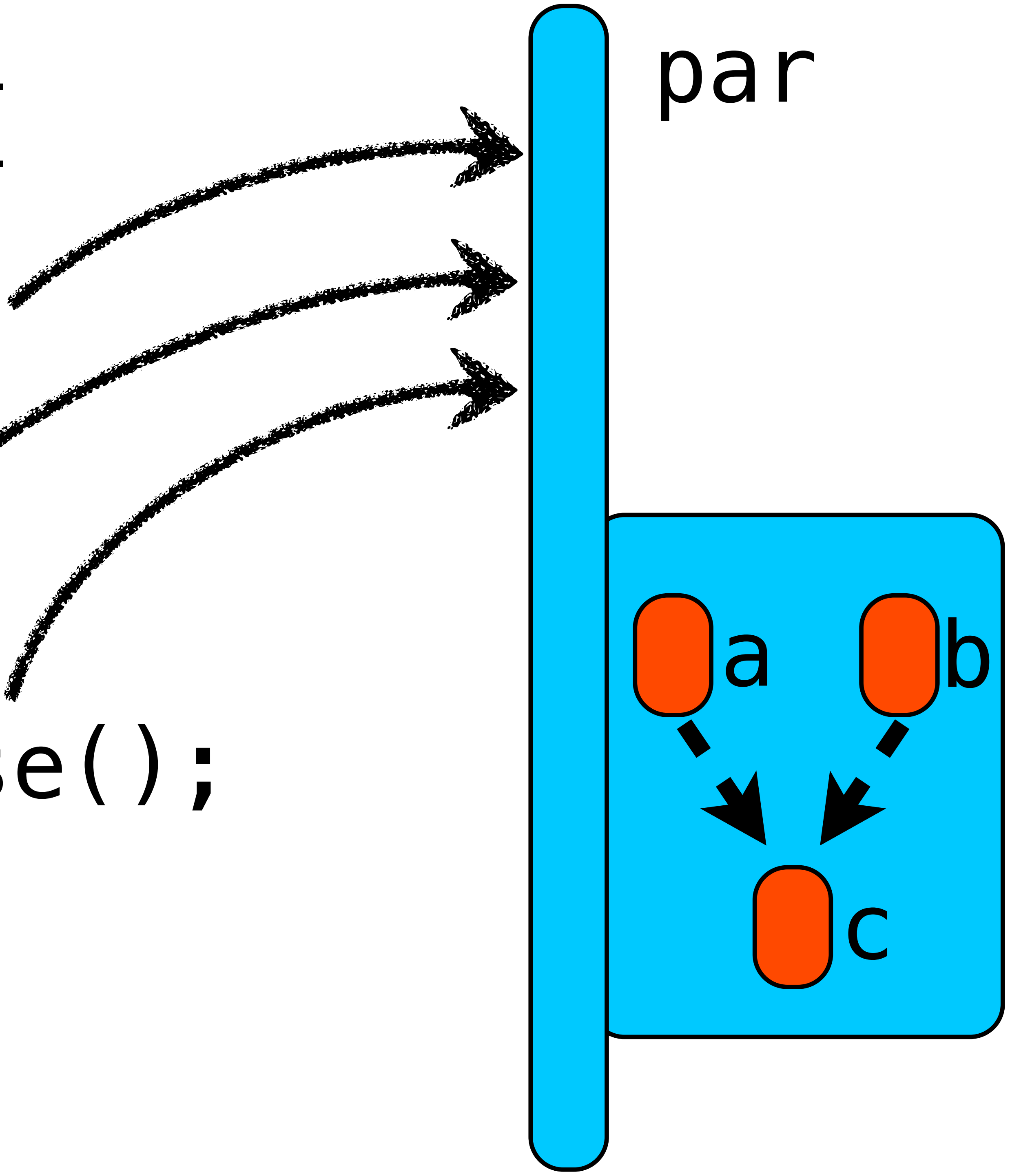
Everything Goes Well

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



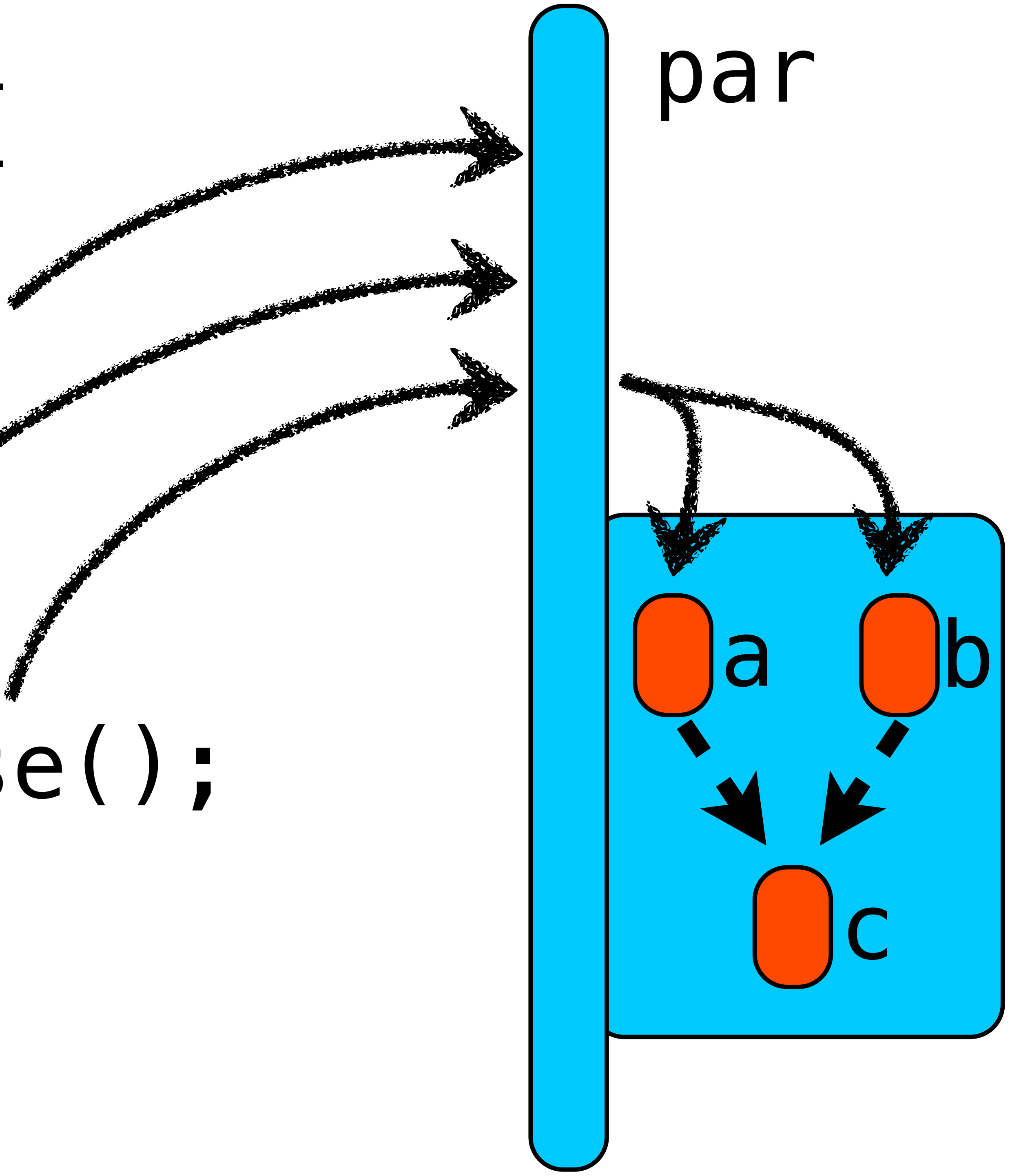
Everything Goes Well

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



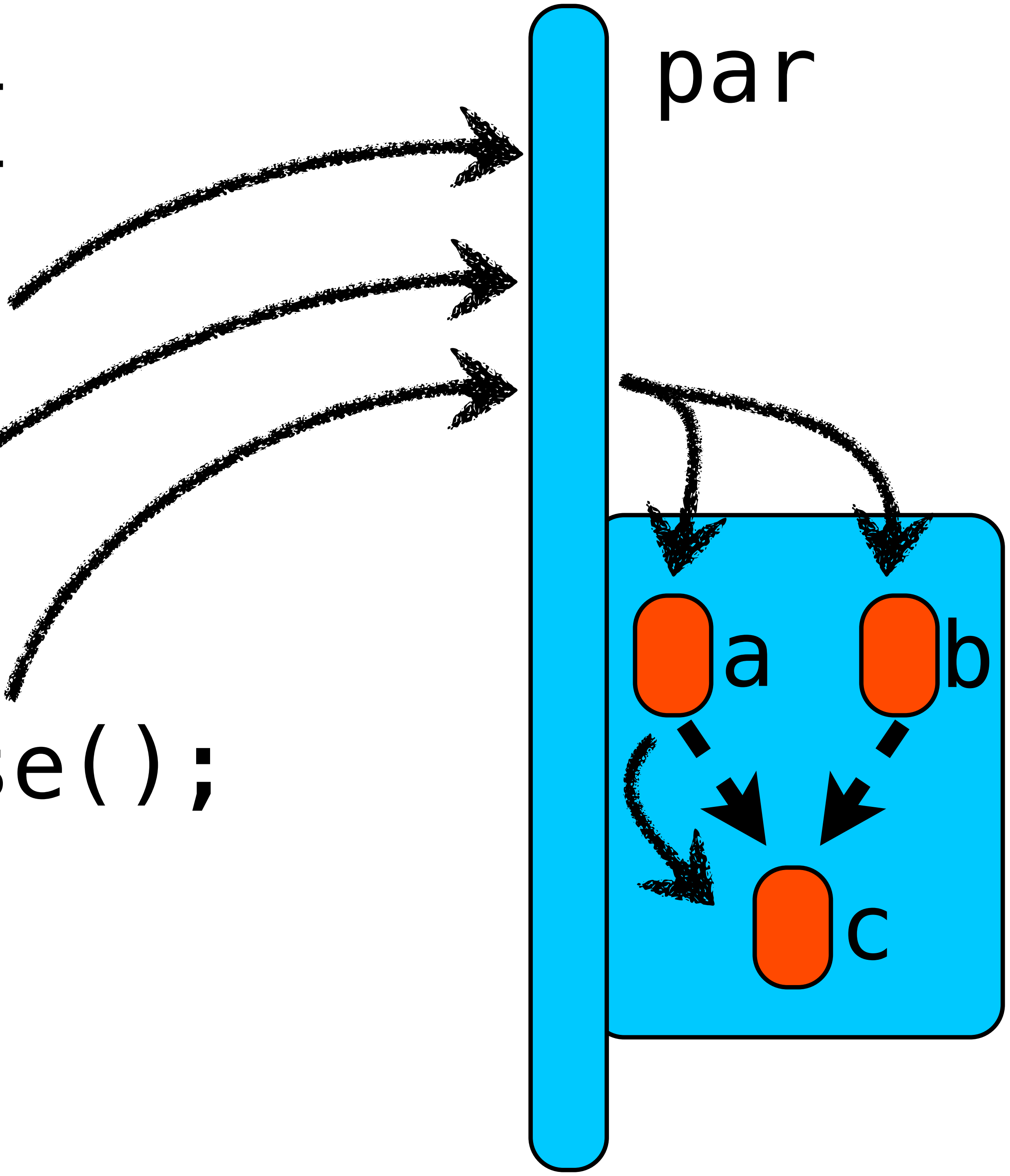
Everything Goes Well

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



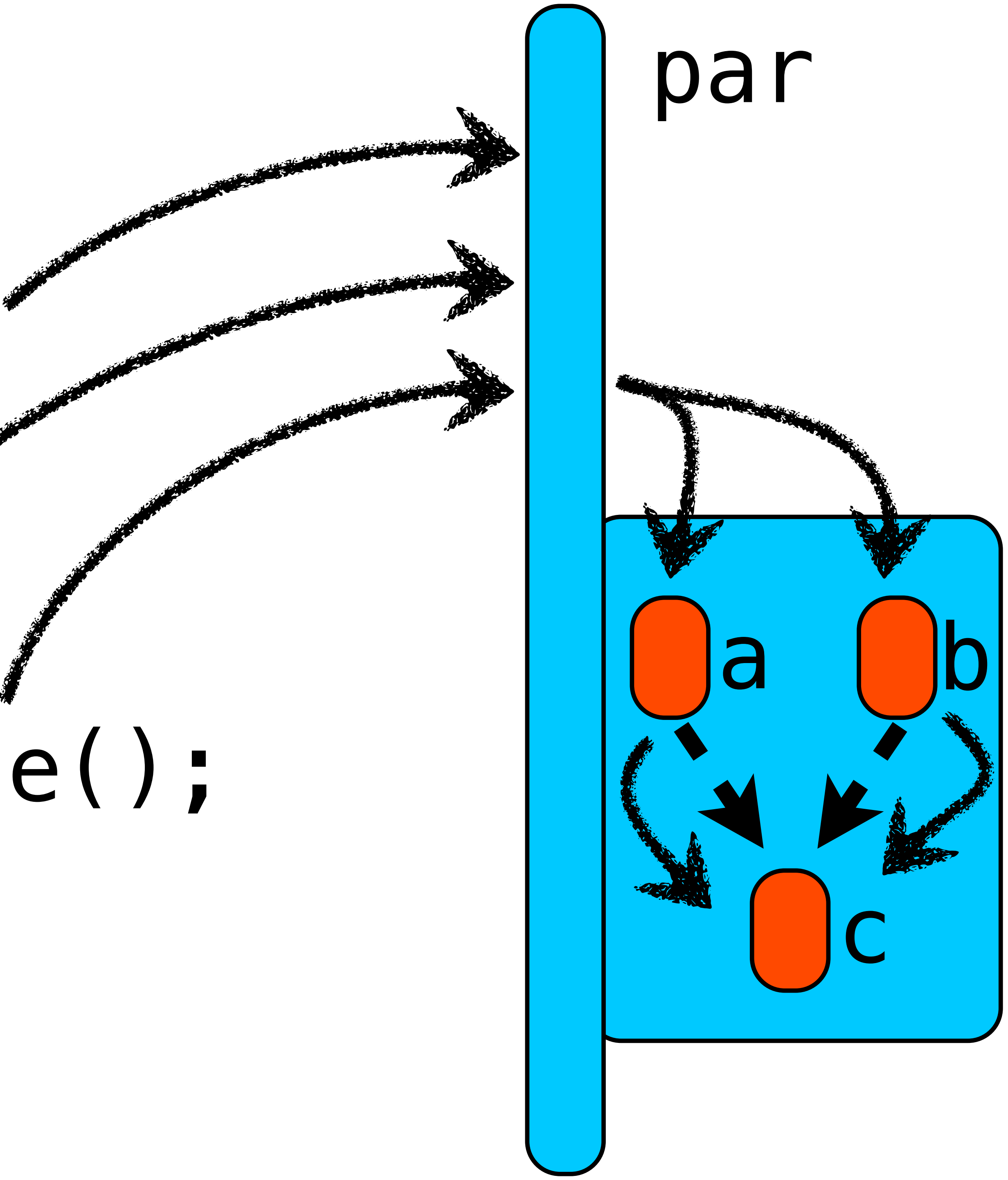
Everything Goes Well

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



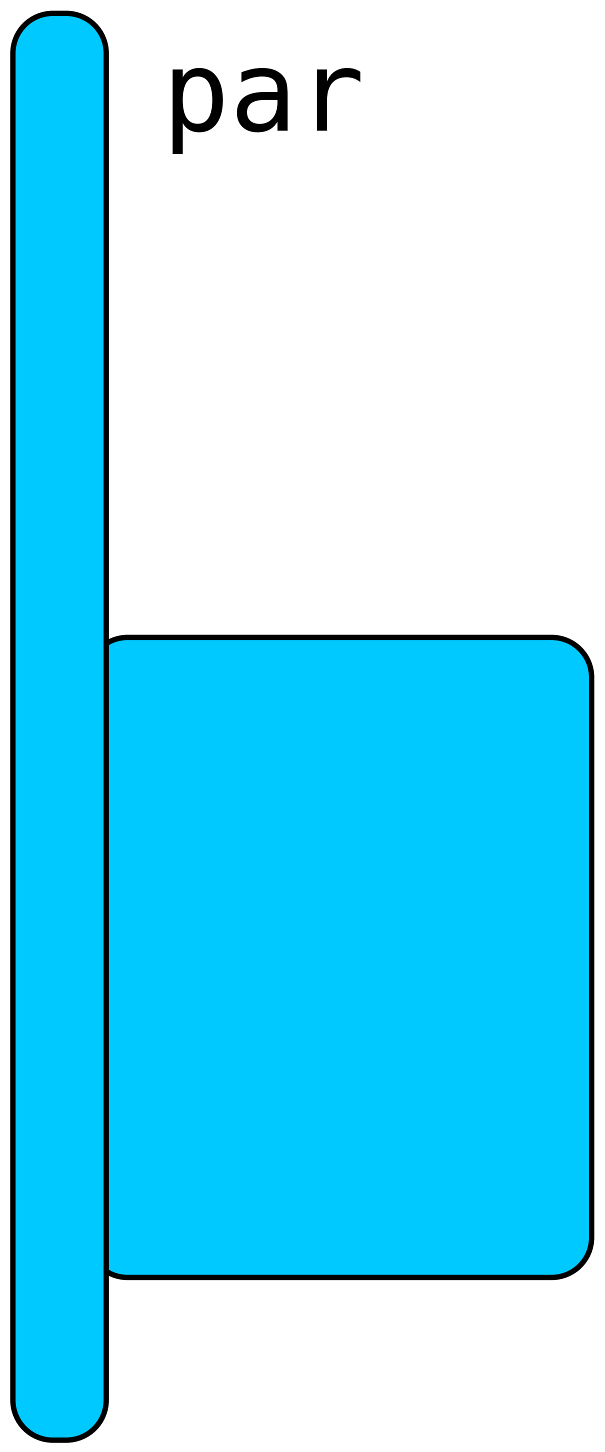
Everything Goes Well

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



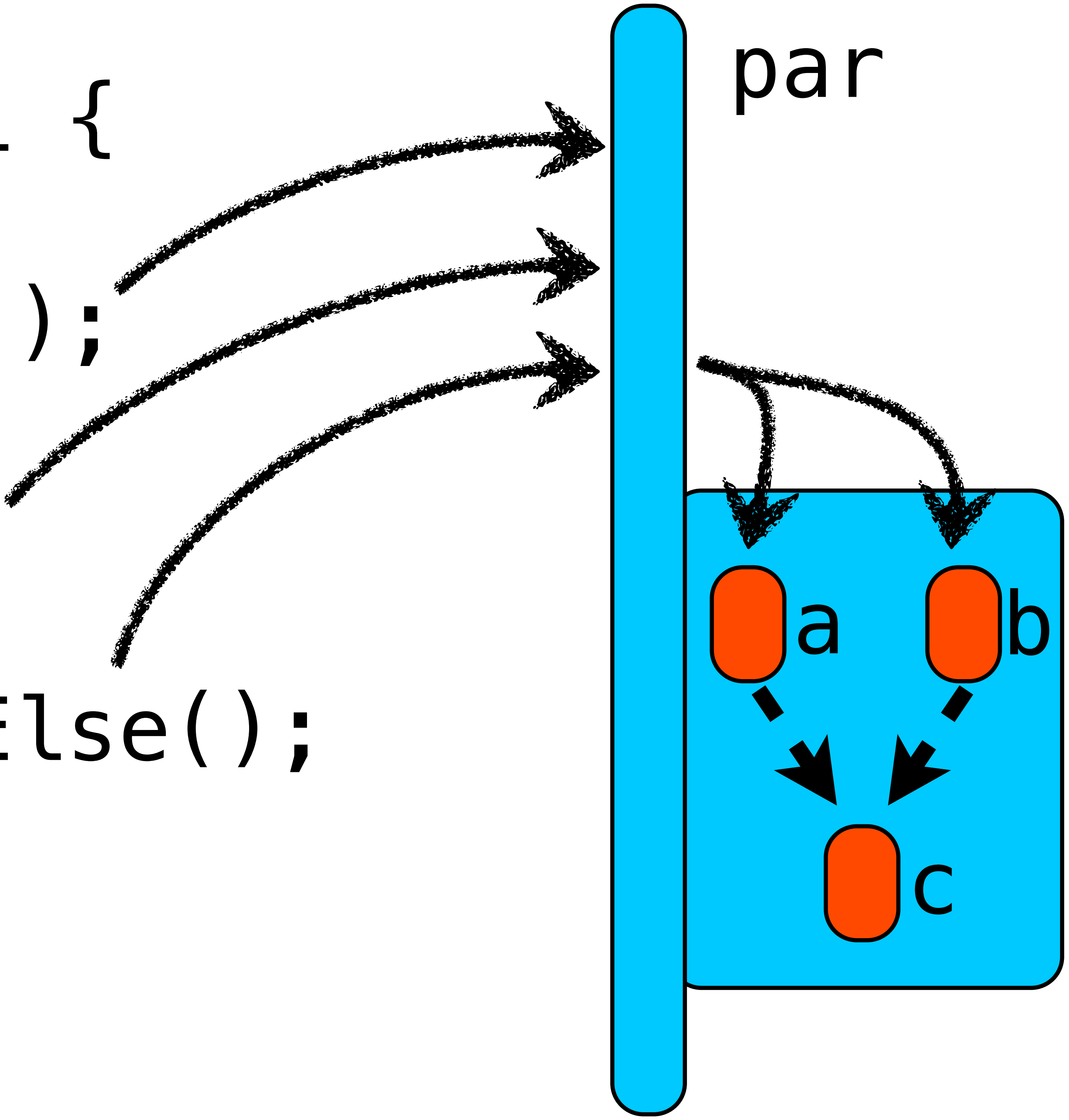
Child Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



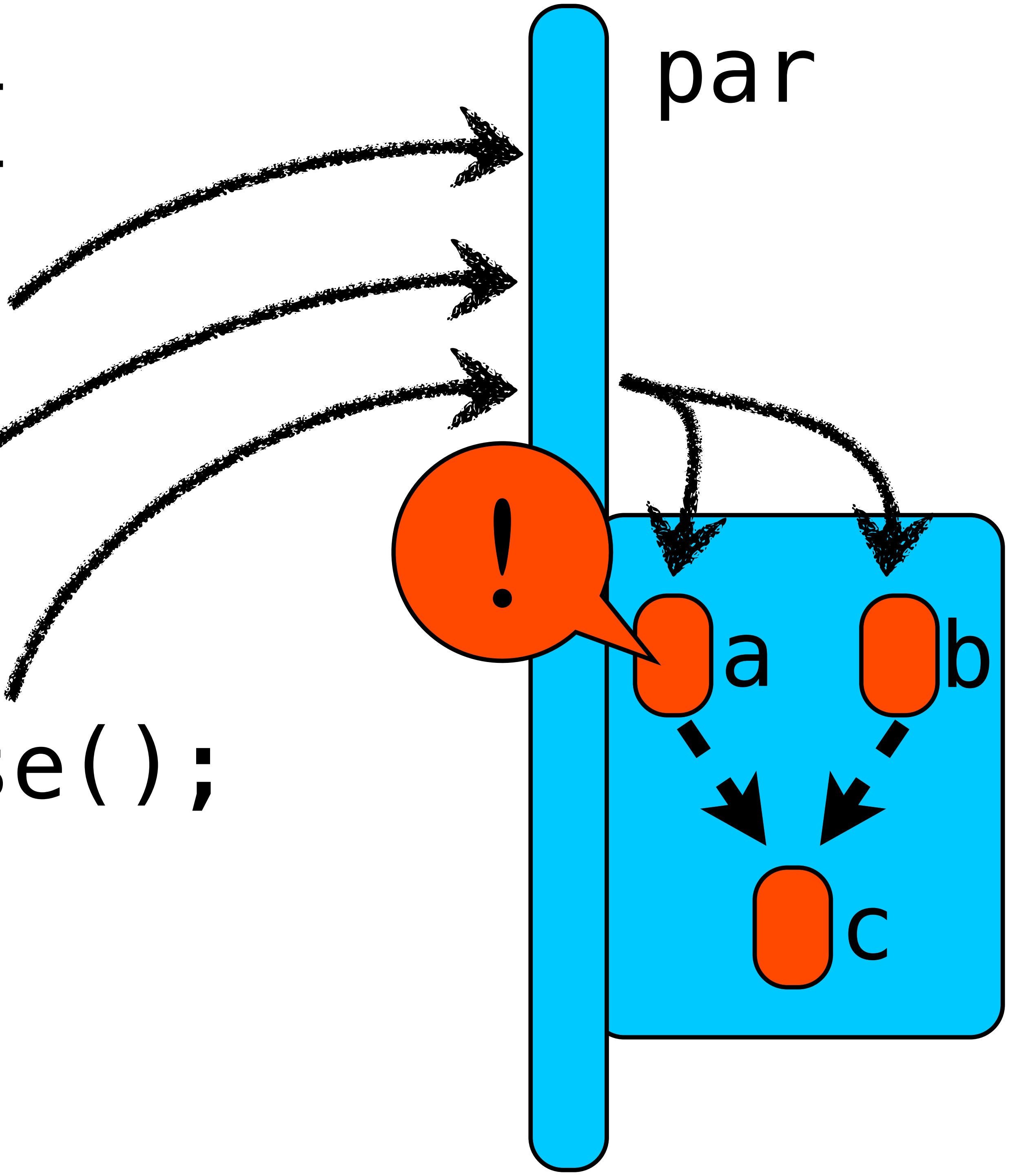
Child Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



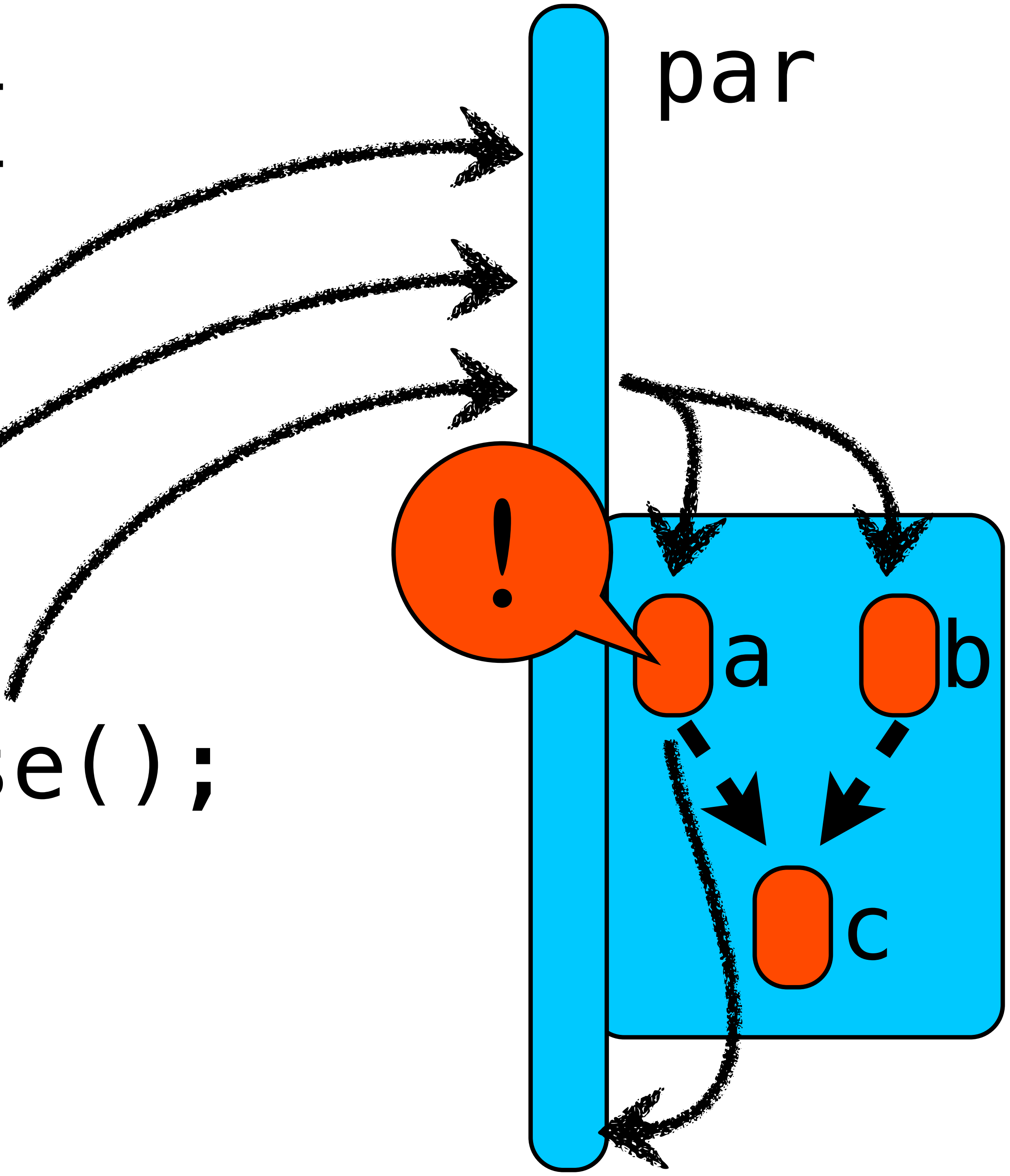
Child Fails

```
par = interval {
   doSomething();
   method(par);
   doSomethingElse();
}
```



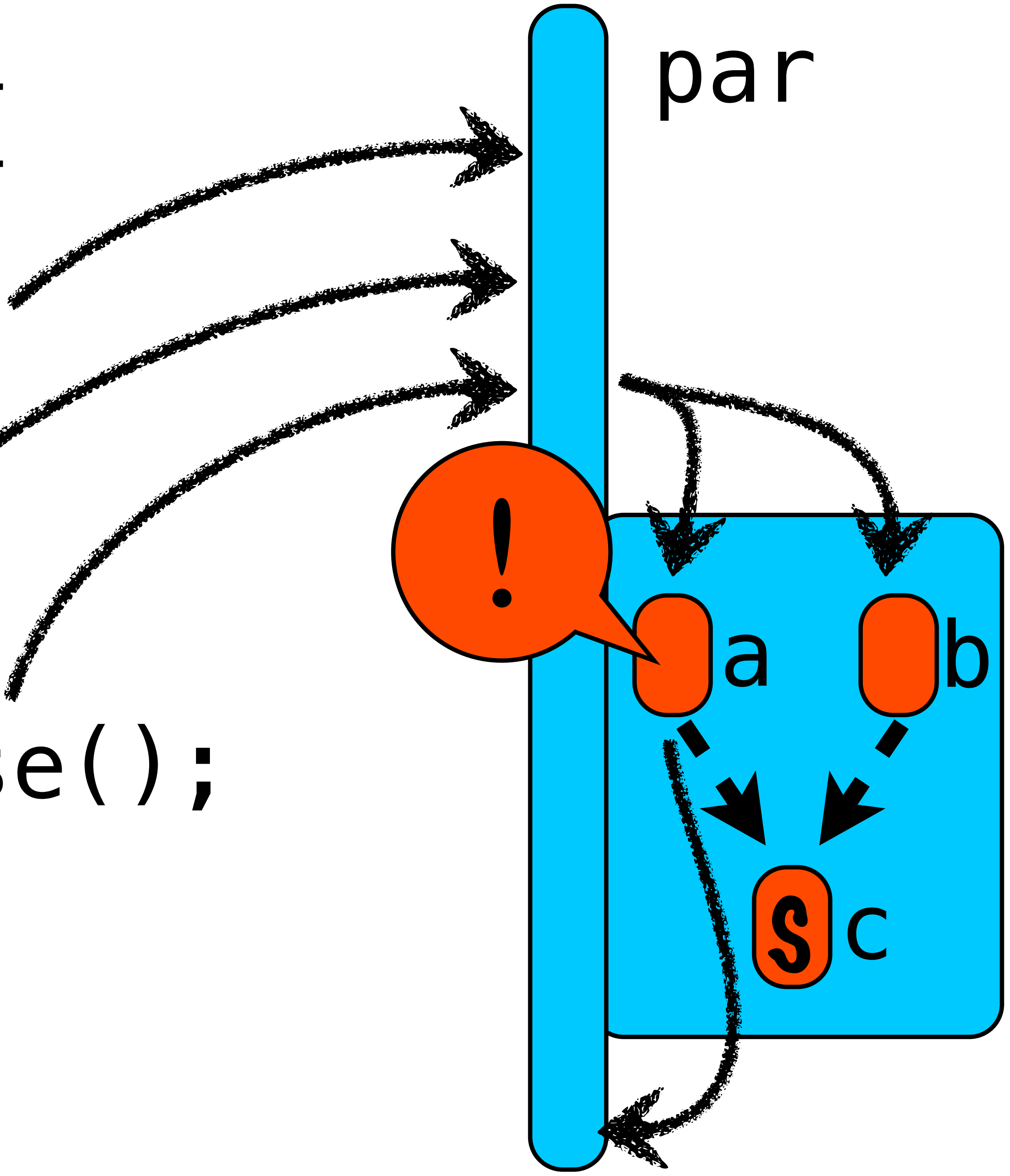
Child Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



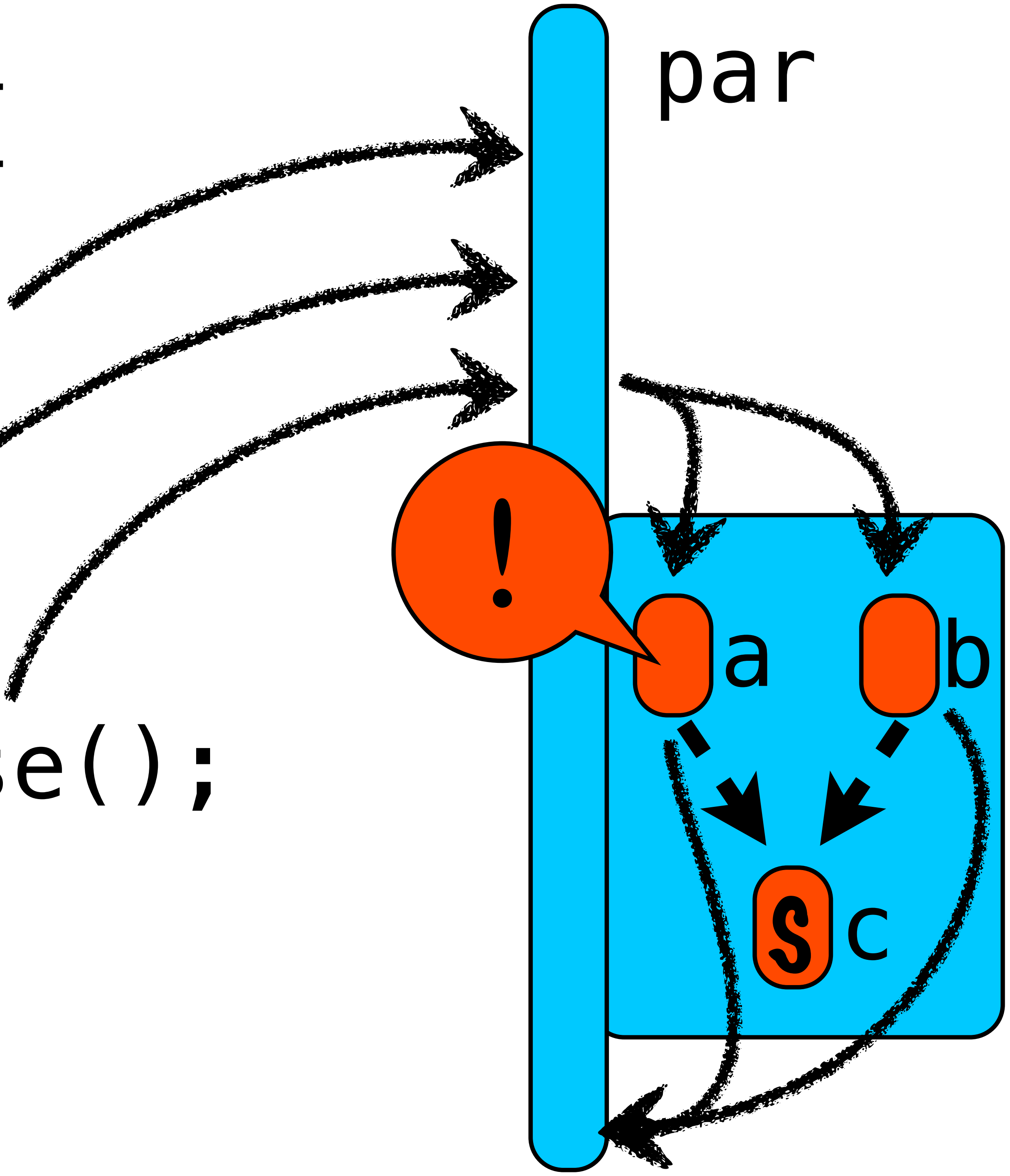
Child Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



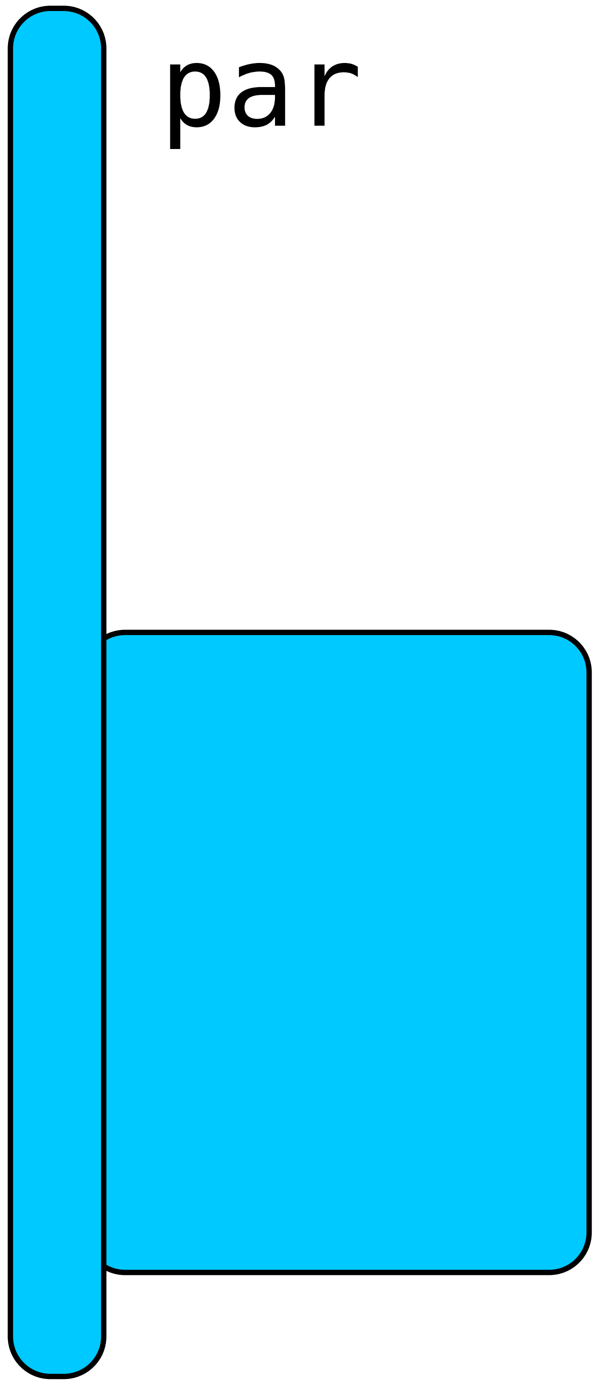
Child Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



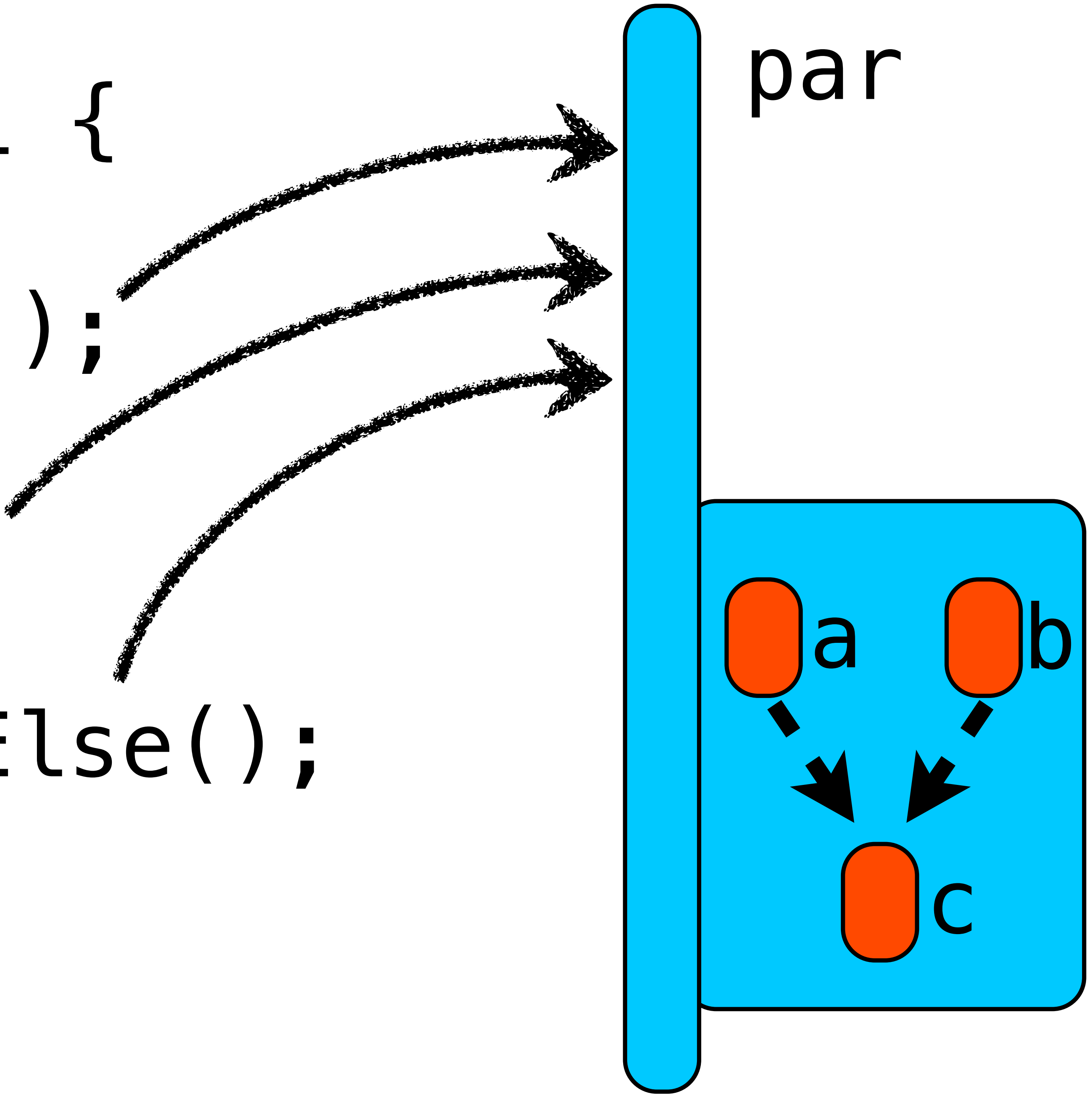
Parent Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



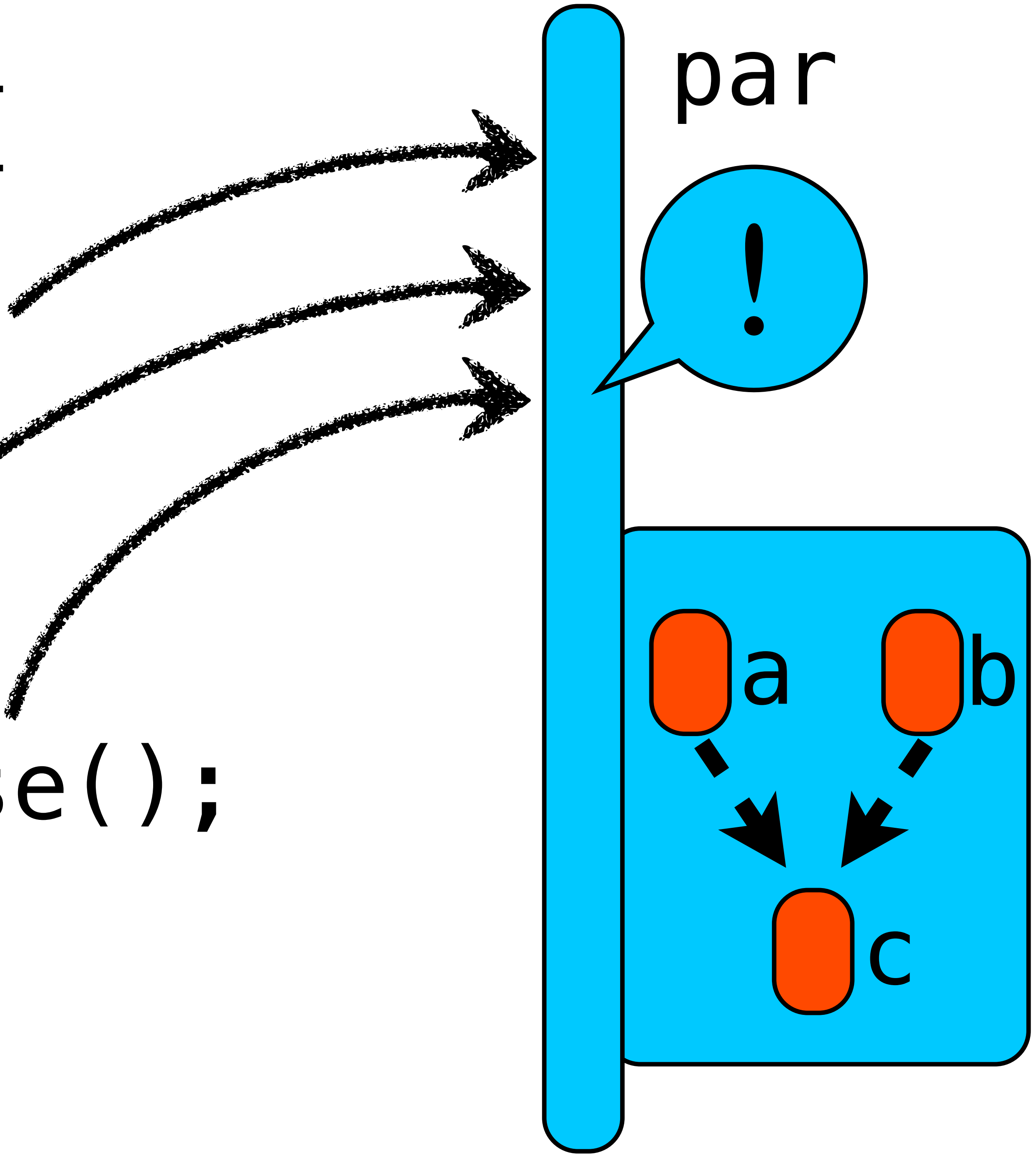
Parent Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



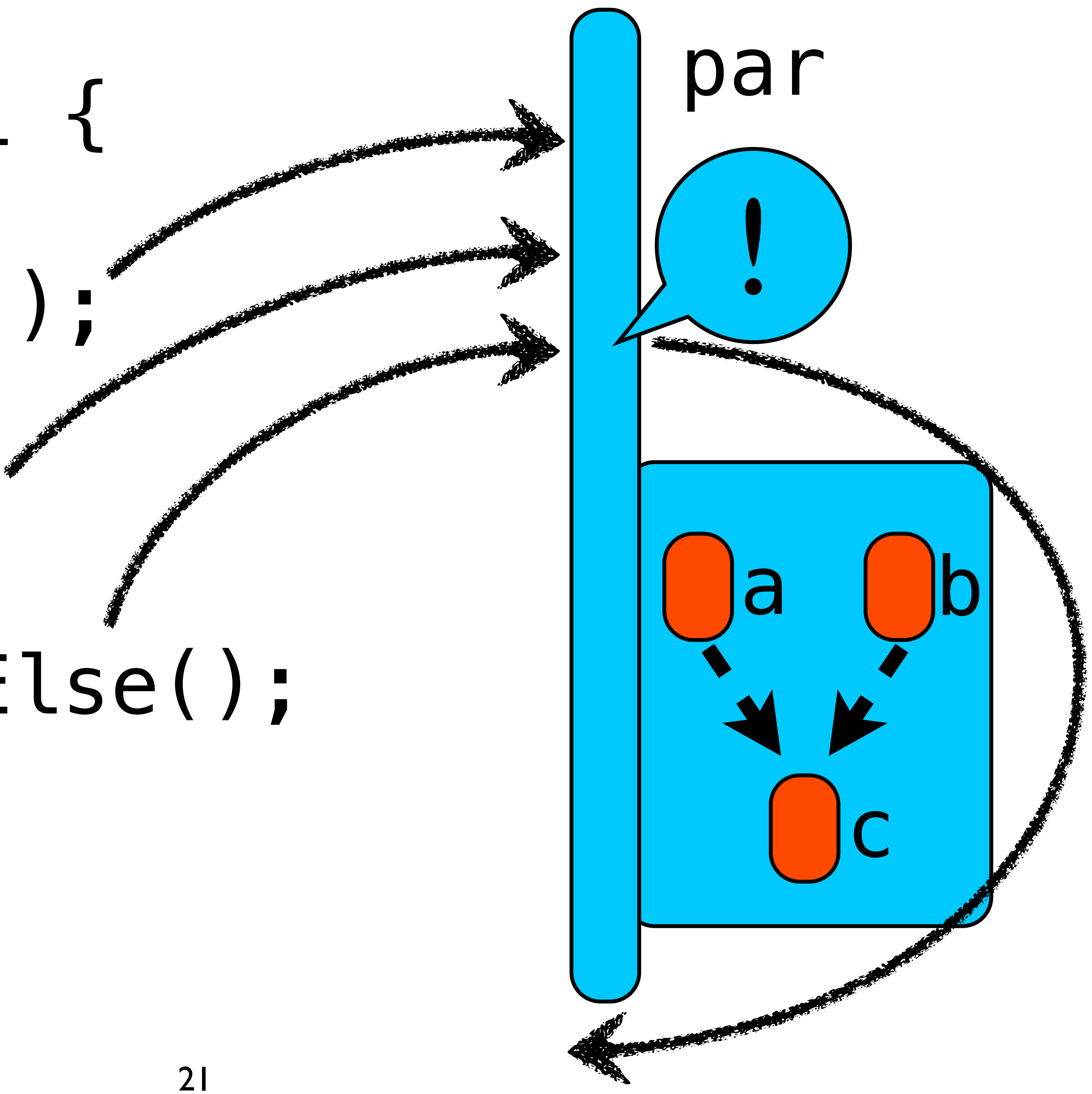
Parent Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



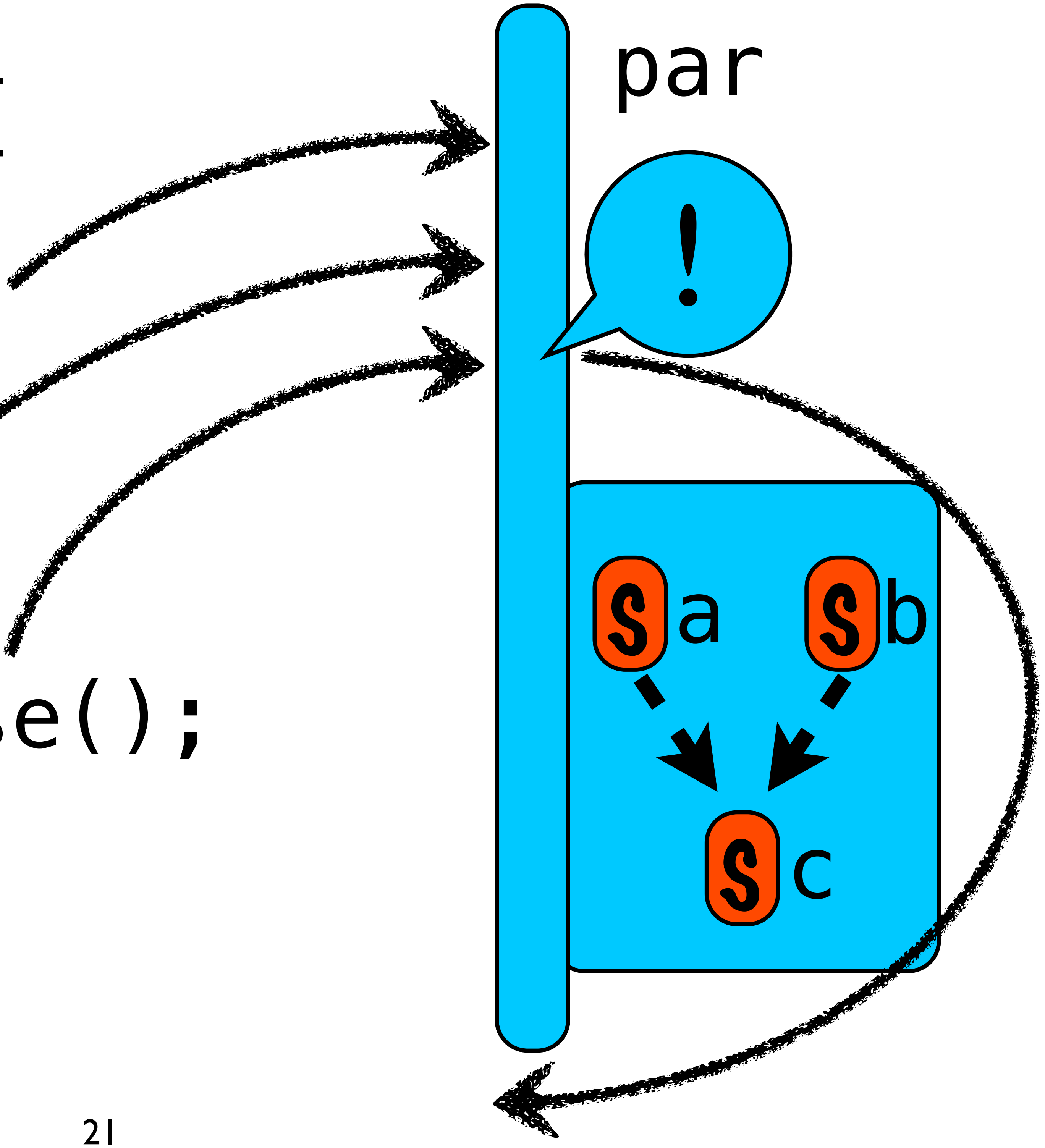
Parent Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



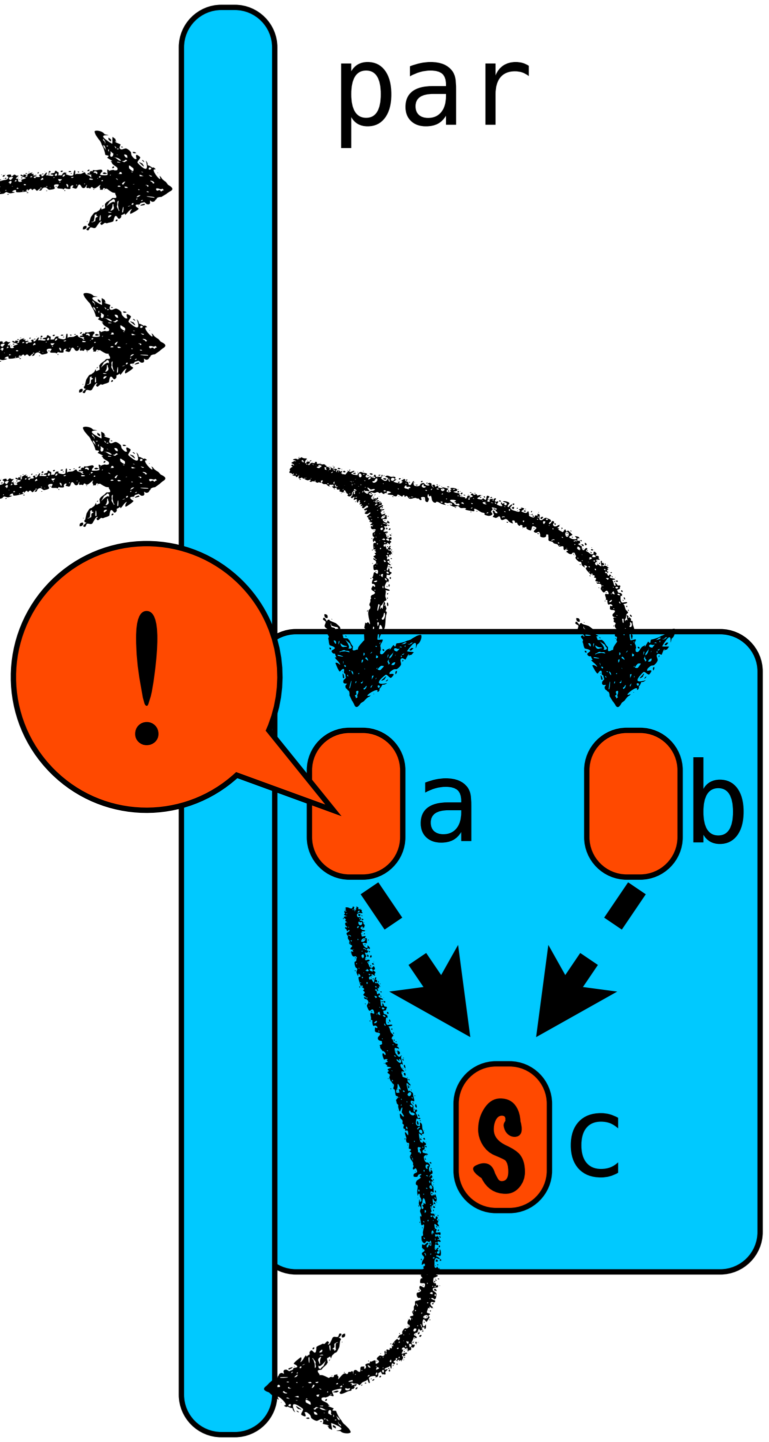
Parent Fails

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



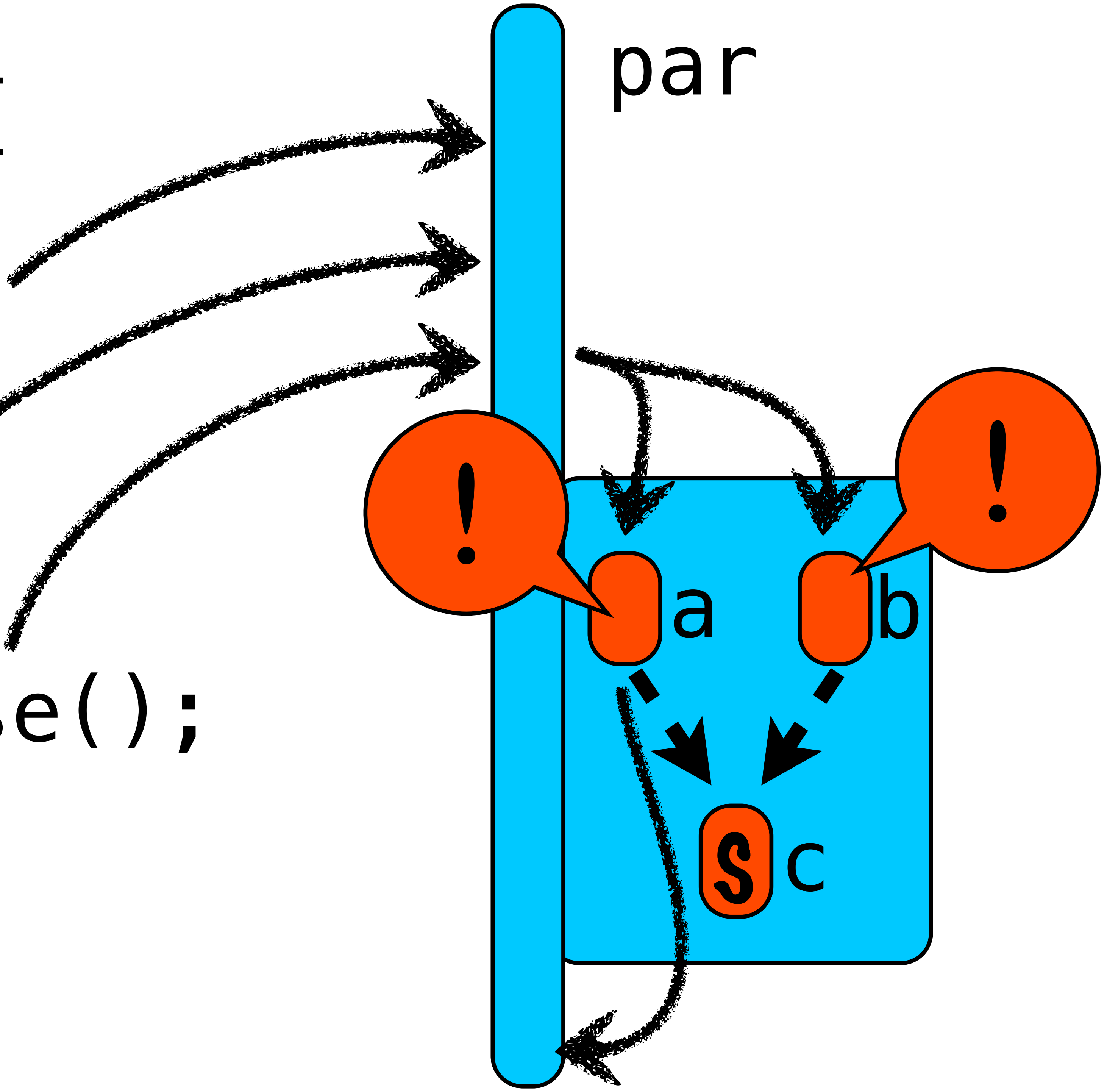
Multiple Children Fail

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



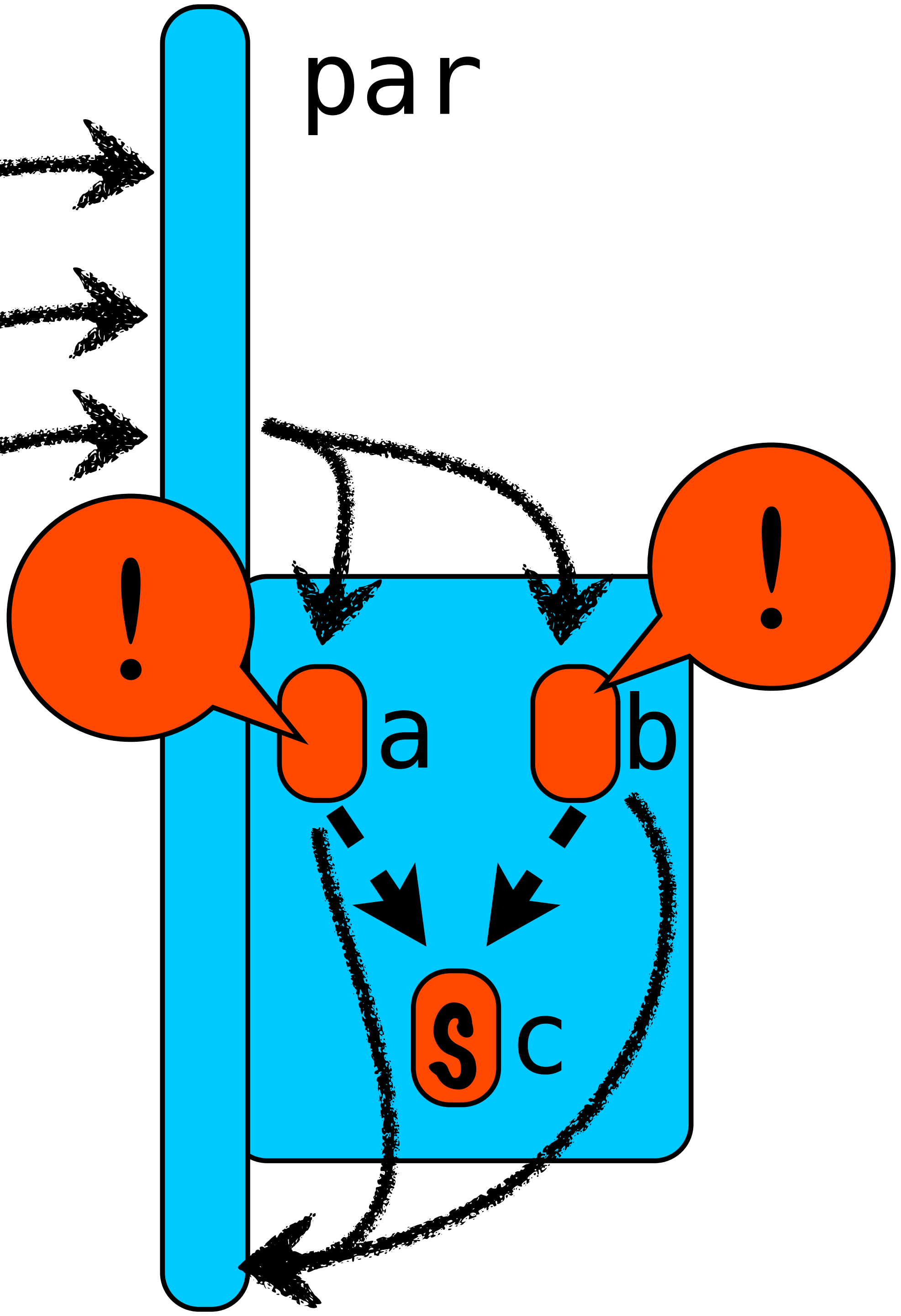
Multiple Children Fail

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



Multiple Children Fail

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
```



Catching Errors

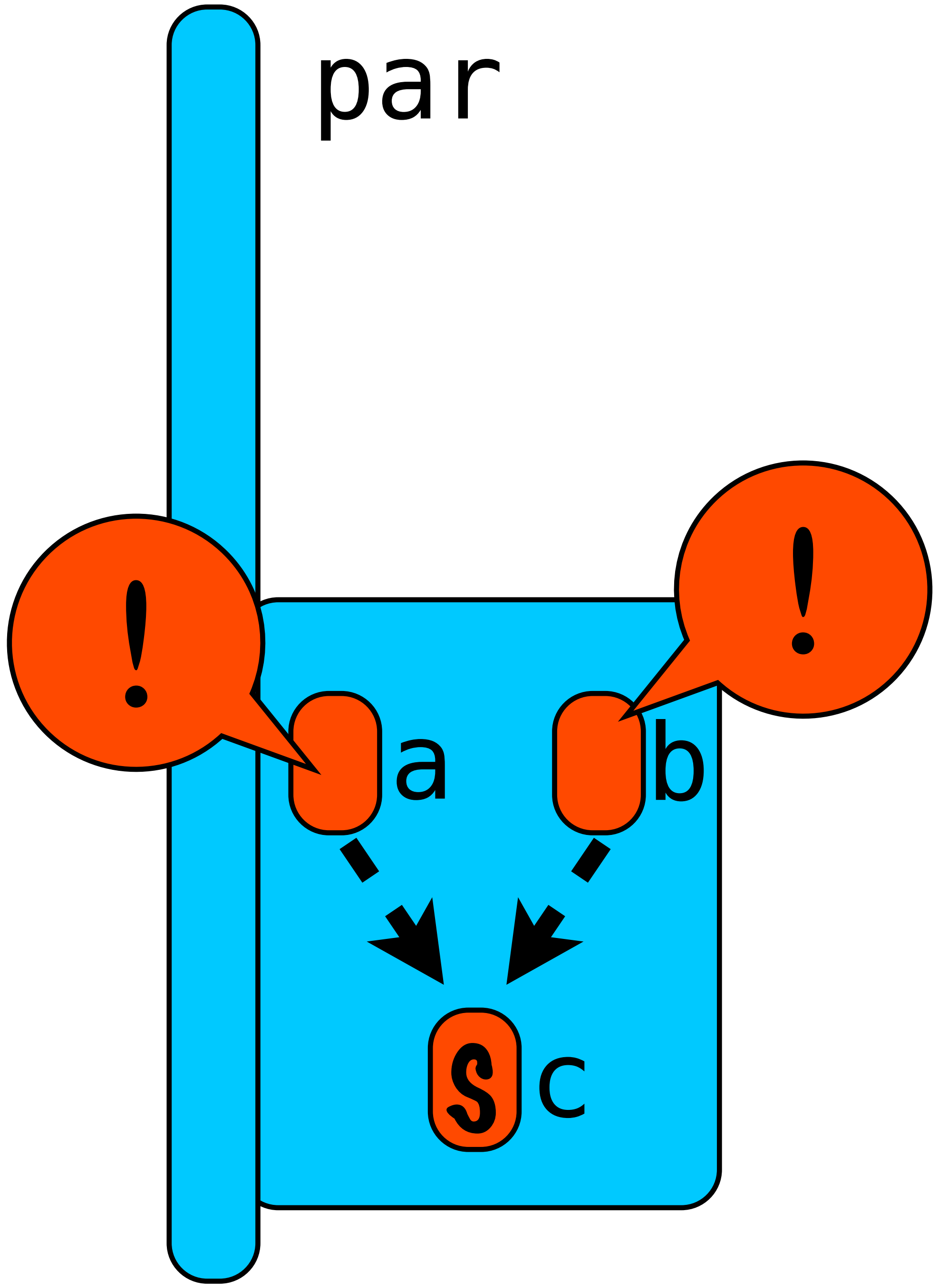
- An interval may catch errors that occur in it or its children.

Catching Errors

```
interval
{
  ...
}
catch(Set<Throwable> errors)
{
  // The default: handle no
  // errors at all.
  return errors;
}
```

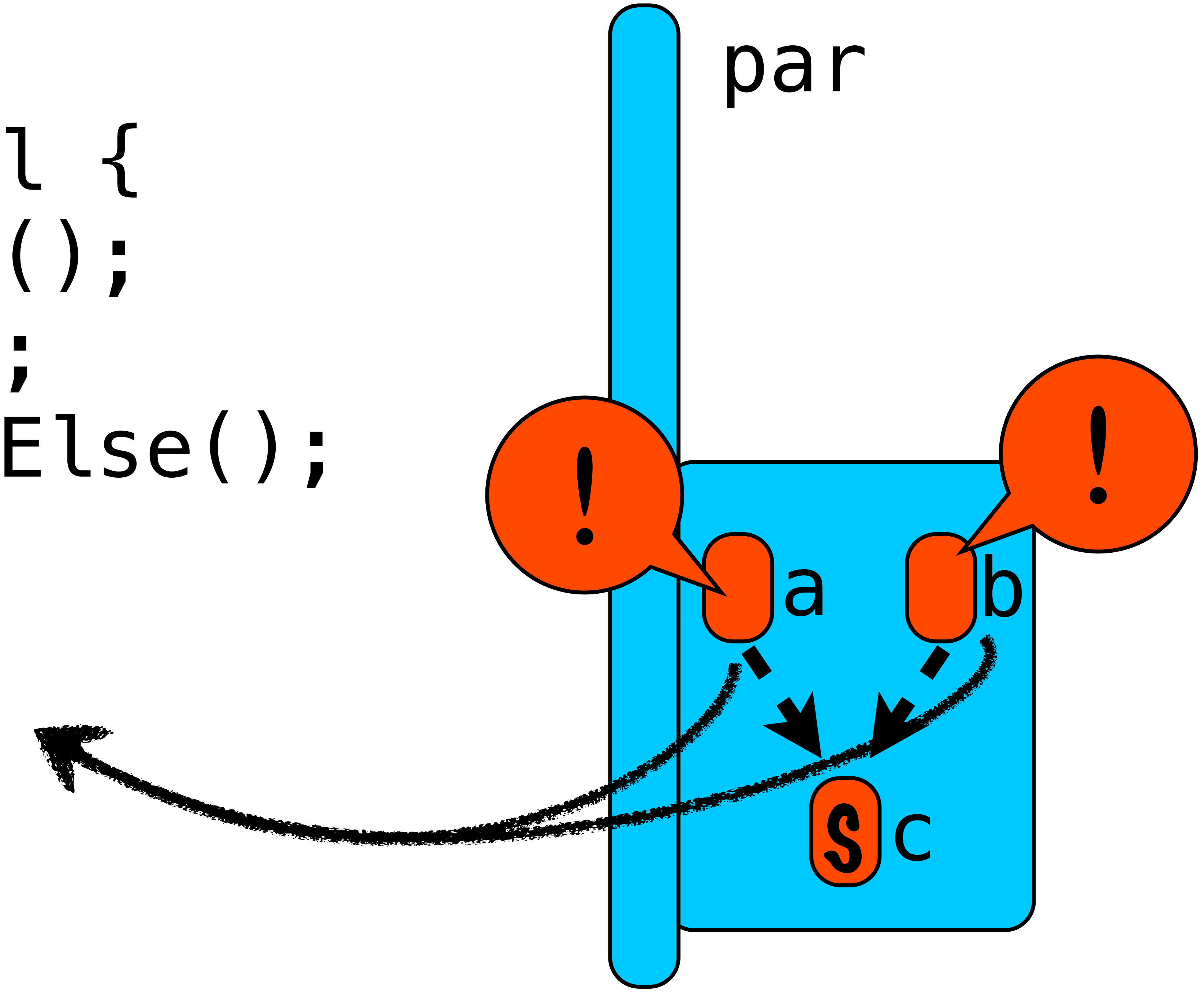
Catching Errors

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
catch (...) {
  ...
}
```

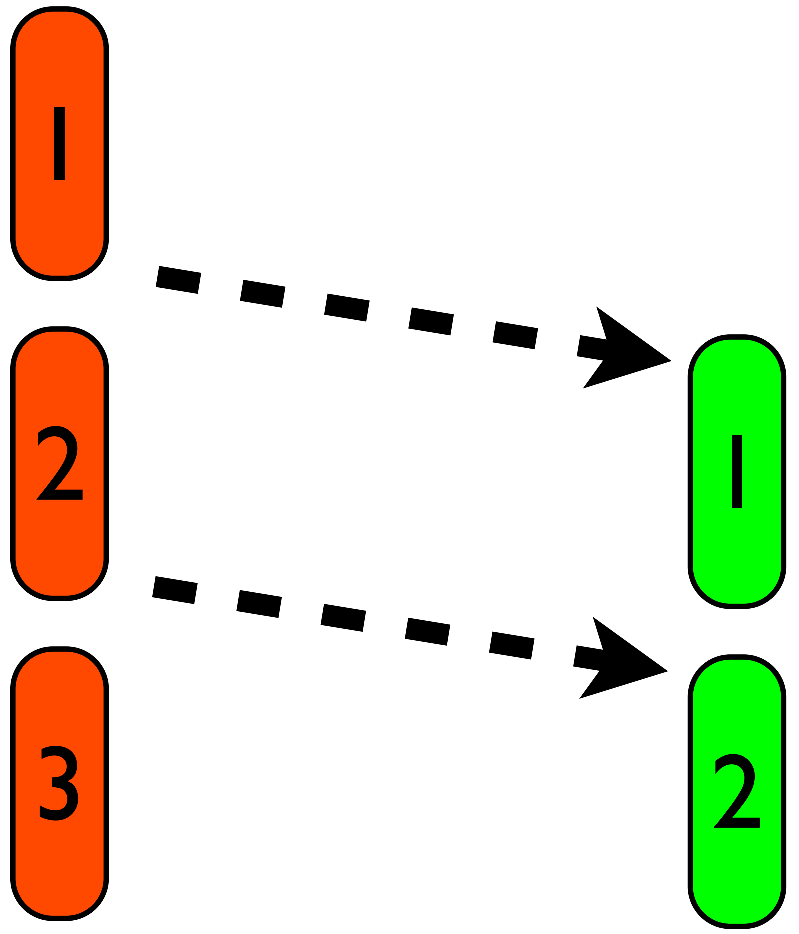


Catching Errors

```
par = interval {
  doSomething();
  method(par);
  doSomethingElse();
}
catch (...) {
  ...
}
```

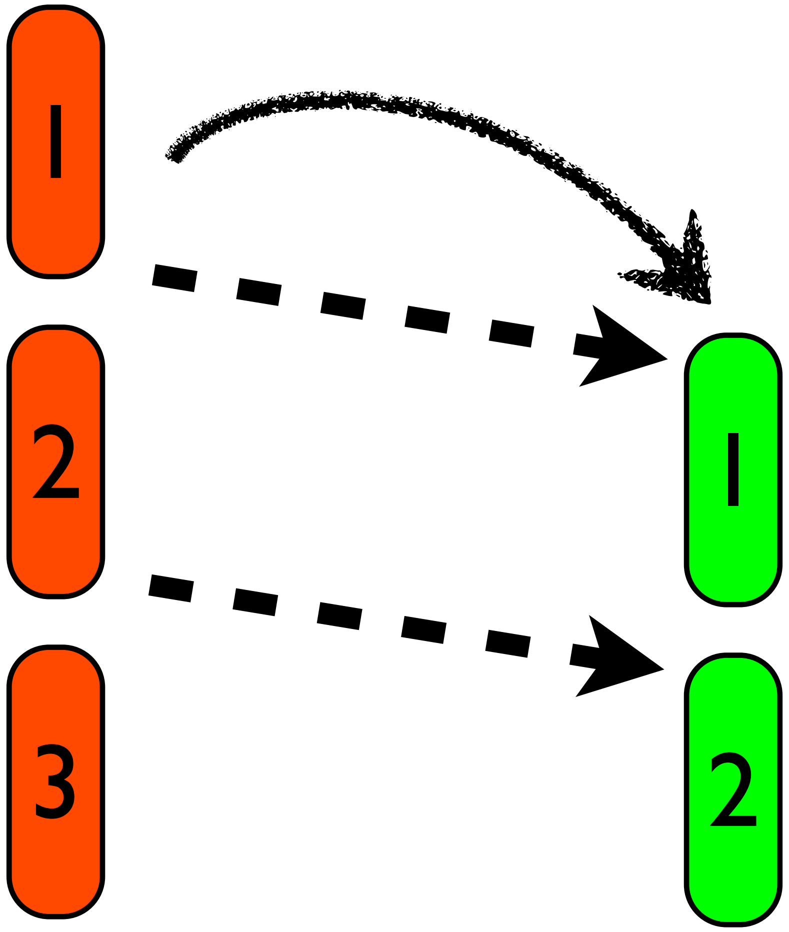


Producer Consumer



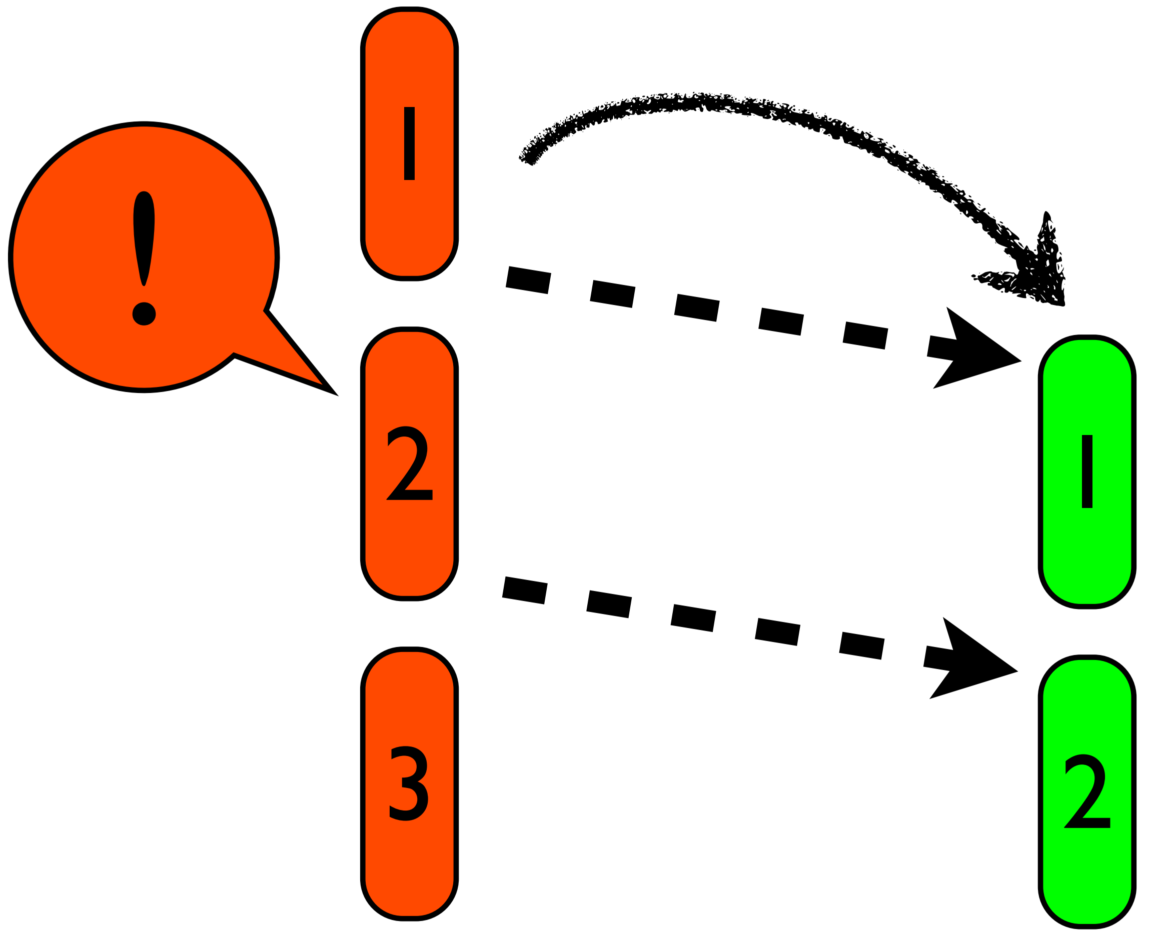
Parent Producer Consumer

Producer Consumer



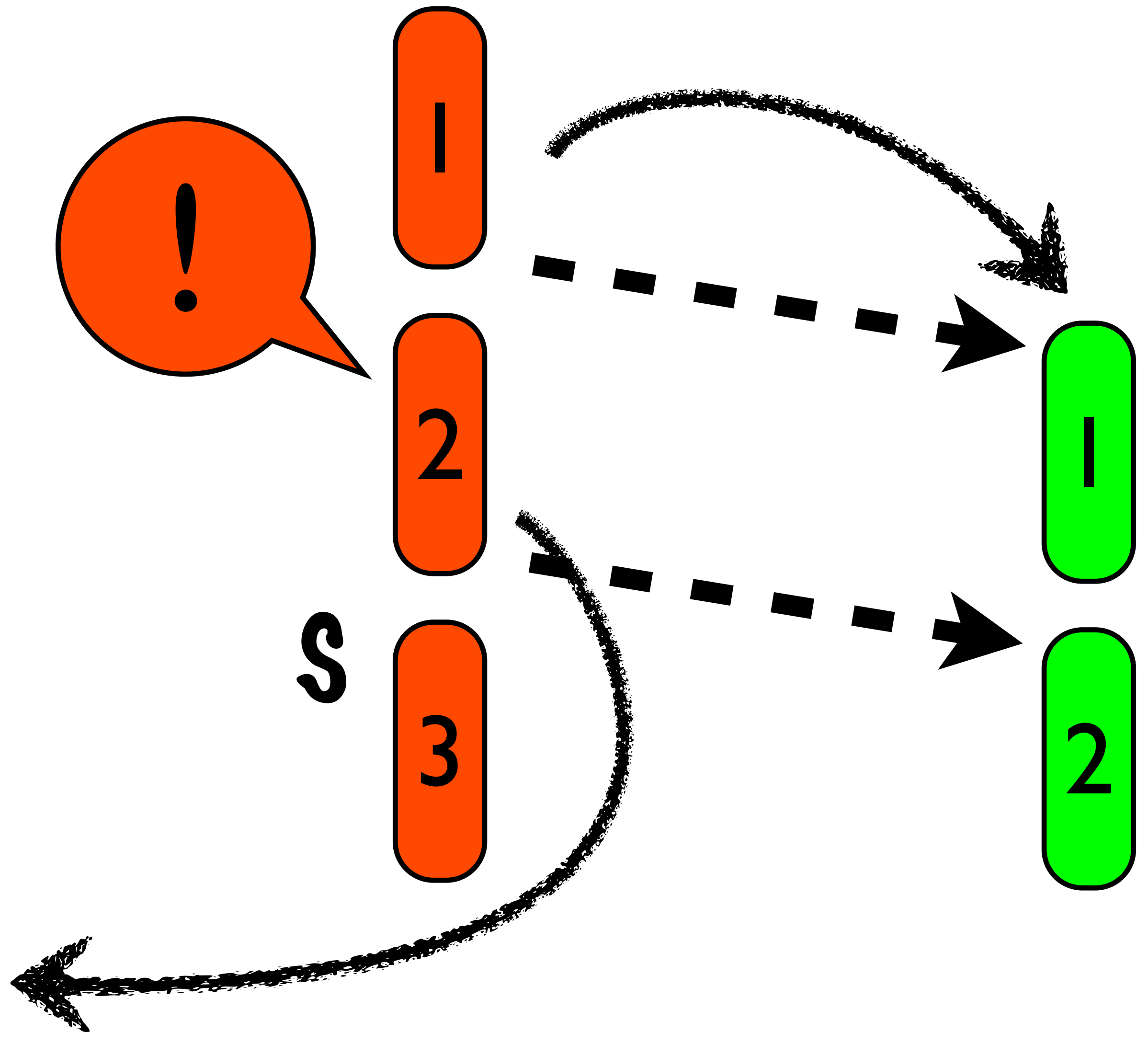
Parent Producer Consumer

Producer Consumer



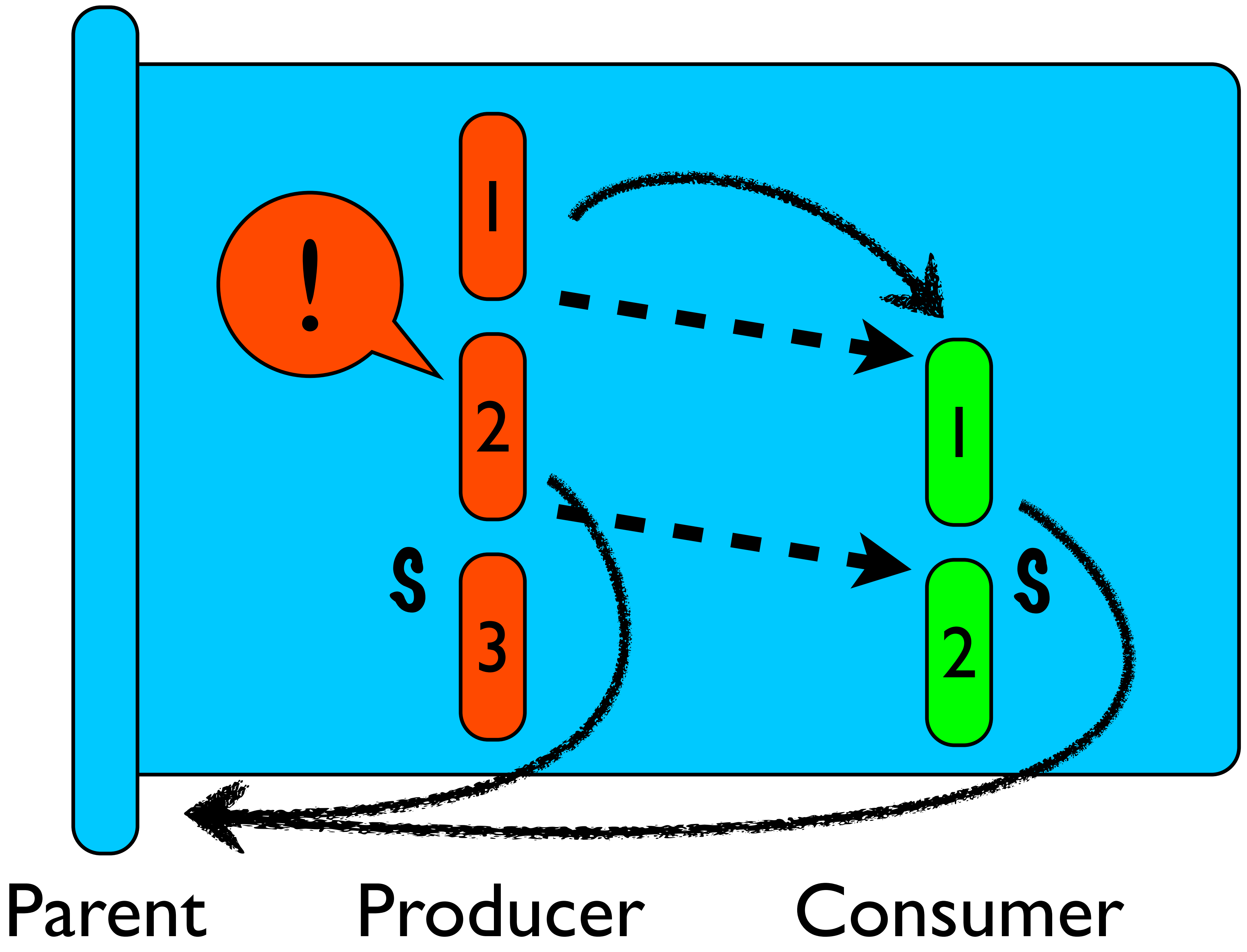
Parent Producer Consumer

Producer Consumer



Parent Producer Consumer

Producer Consumer



Experience

- Interval library is publicly available
  - http://intervals.inf.ethz.ch
- Used to implement a number of examples
  - Bounded-Buffer Producer-Consumer
  - Java Grande Forum
  - etc

Related Work

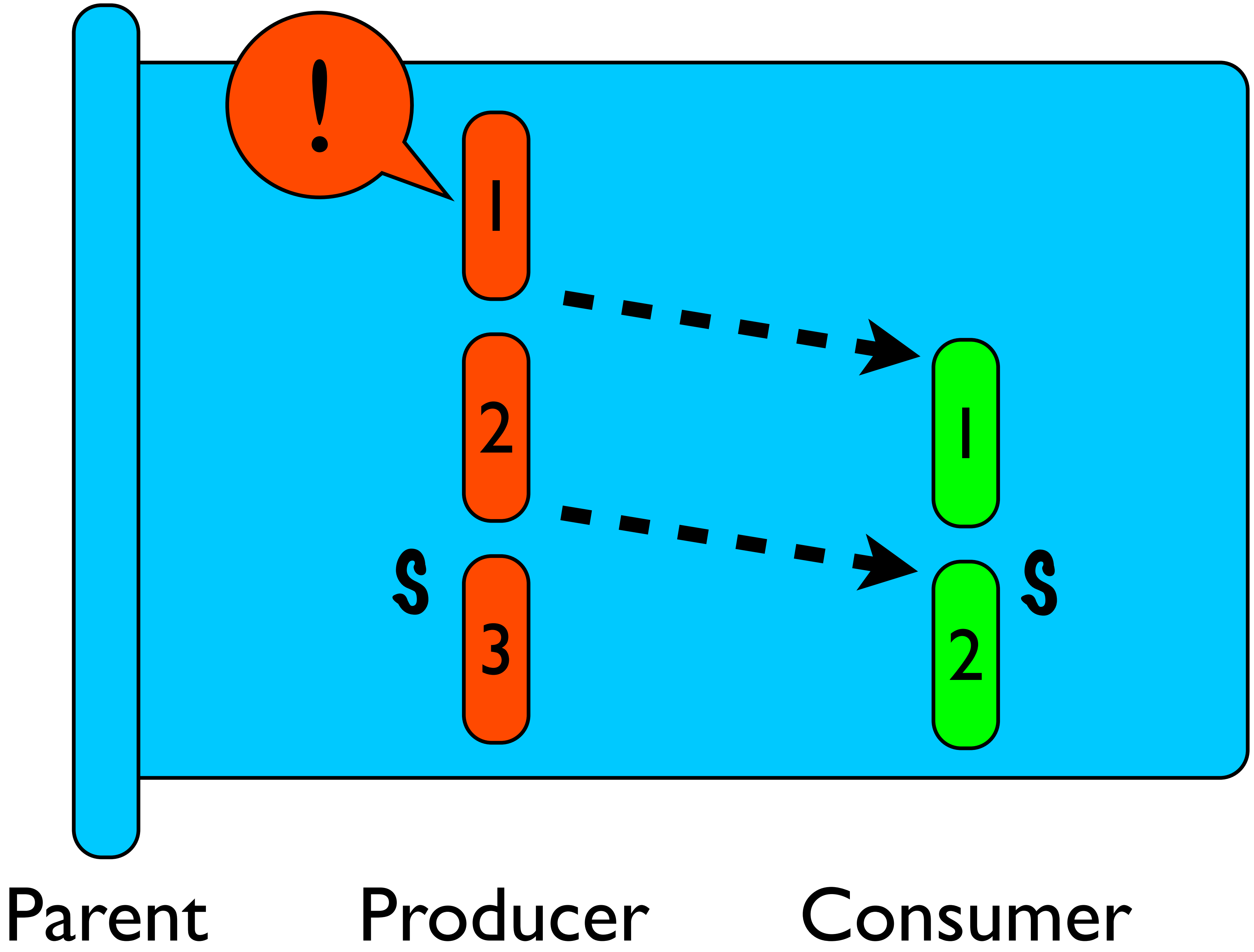
- JCilk:
  - Purely hierarchical model.
  - Sibling computations aborted as well.
- Microsoft Parallel Extensions:
  - Errors must be “observed”
- Failboxes:
  - Propagate errors through data structures rather than control flow.

Conclusions

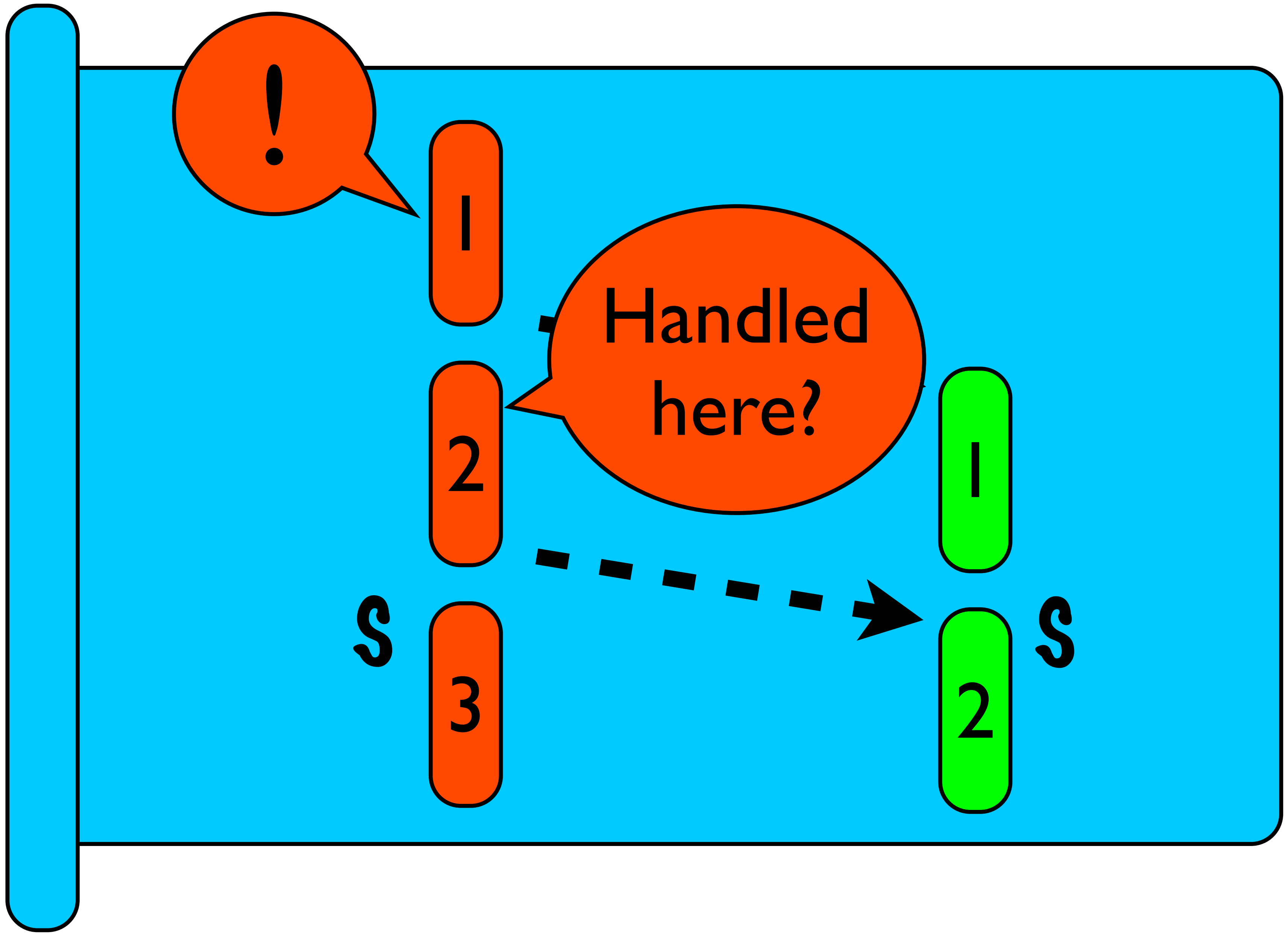
- Typical threaded model does not provide enough information to propagate errors.
- Explicit happens-before relation allows sequential exception handling to be generalized to a parallel setting.
- Our model provides a deterministic, well-specified set of points where an error can be handled.

Backup Slides

Successors

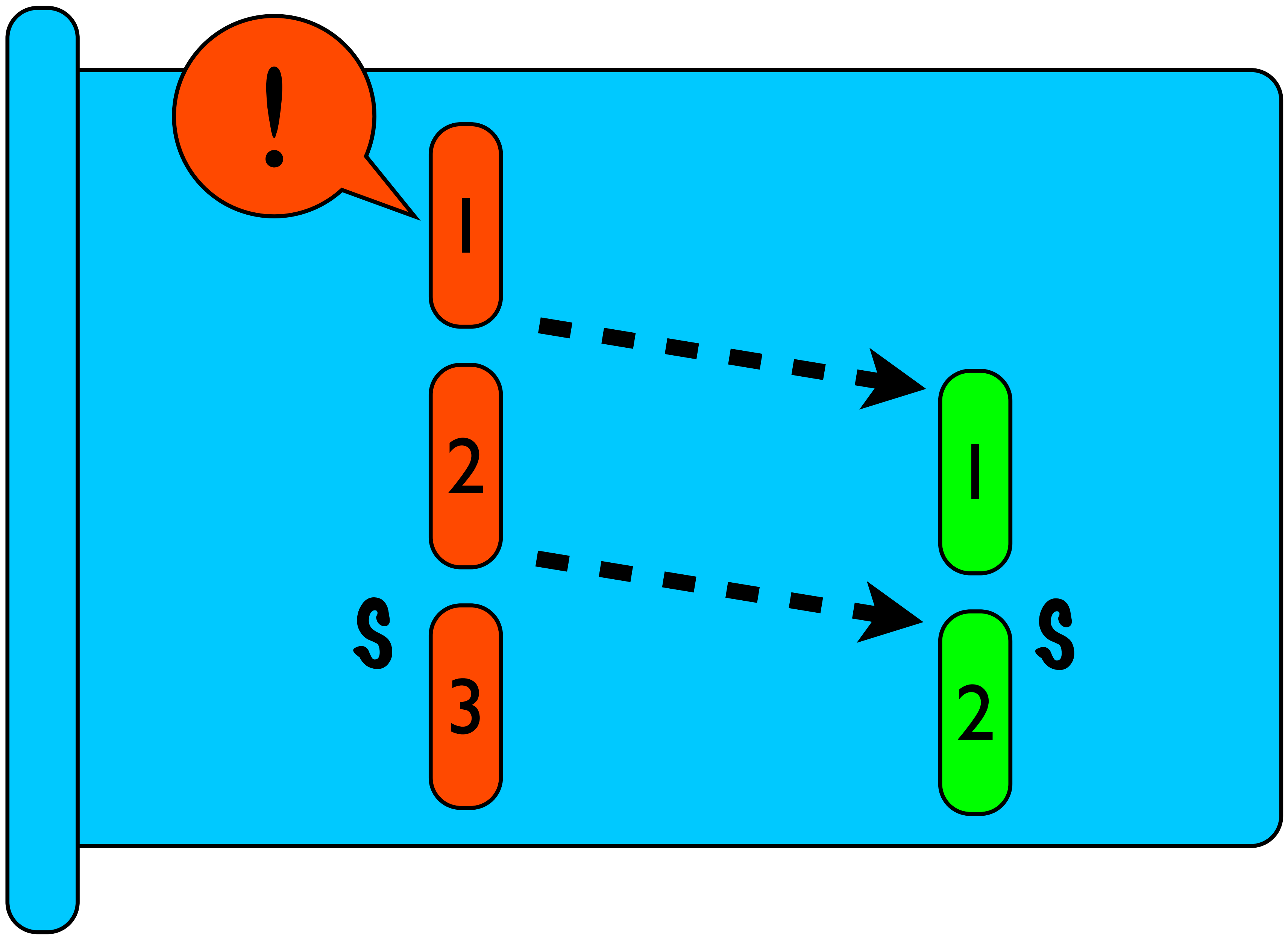


Successors



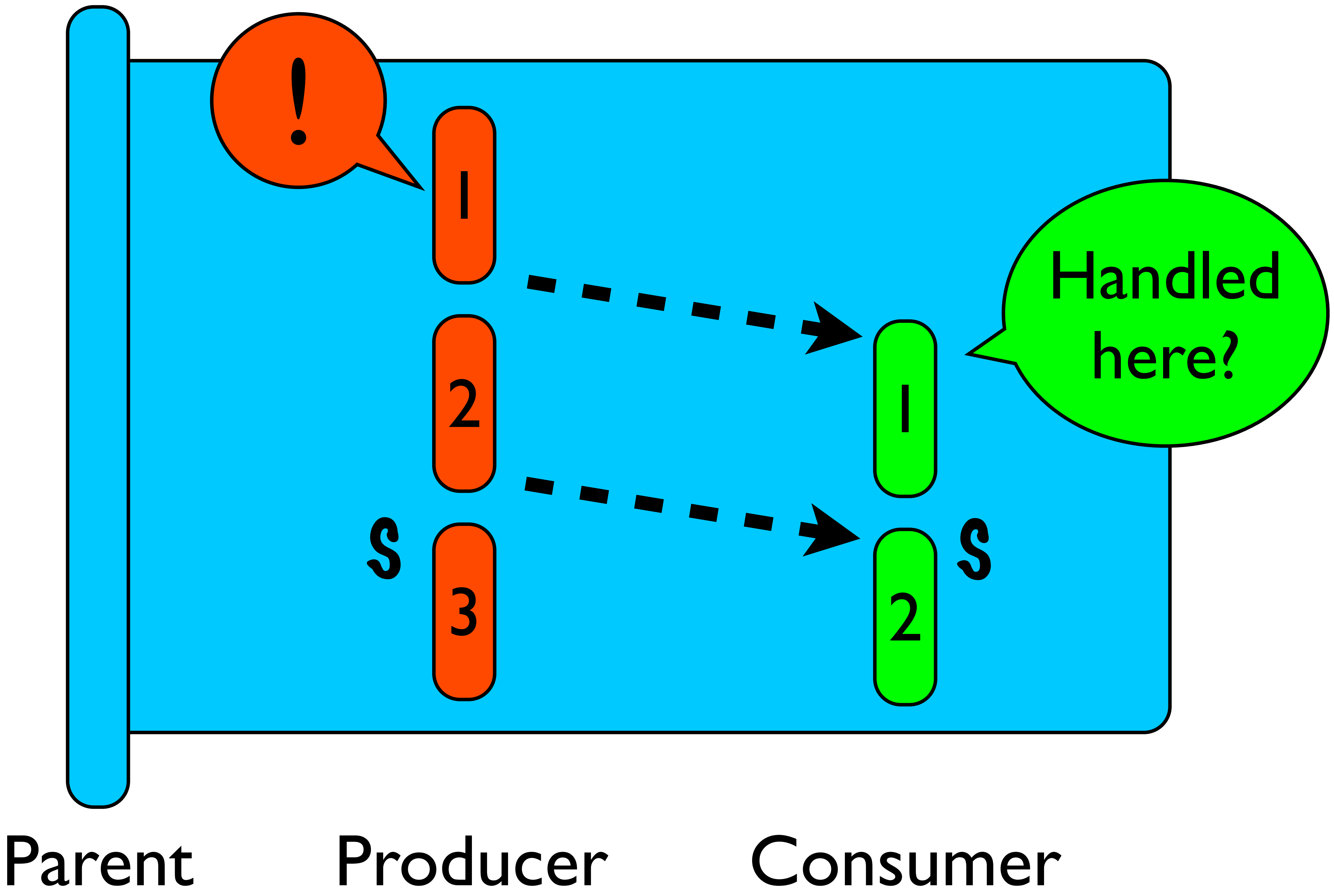
Parent Producer Consumer

Successors

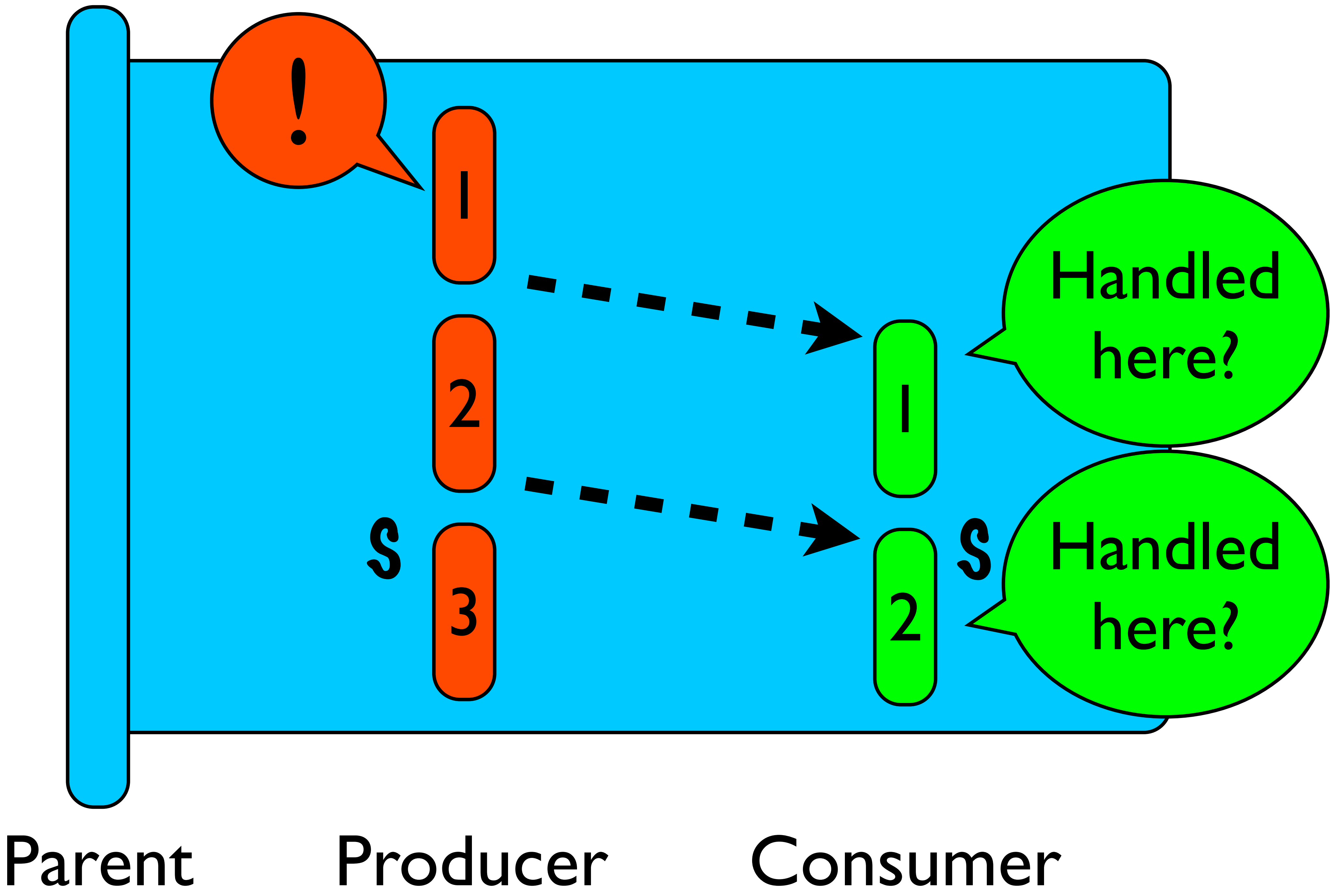


Parent Producer Consumer

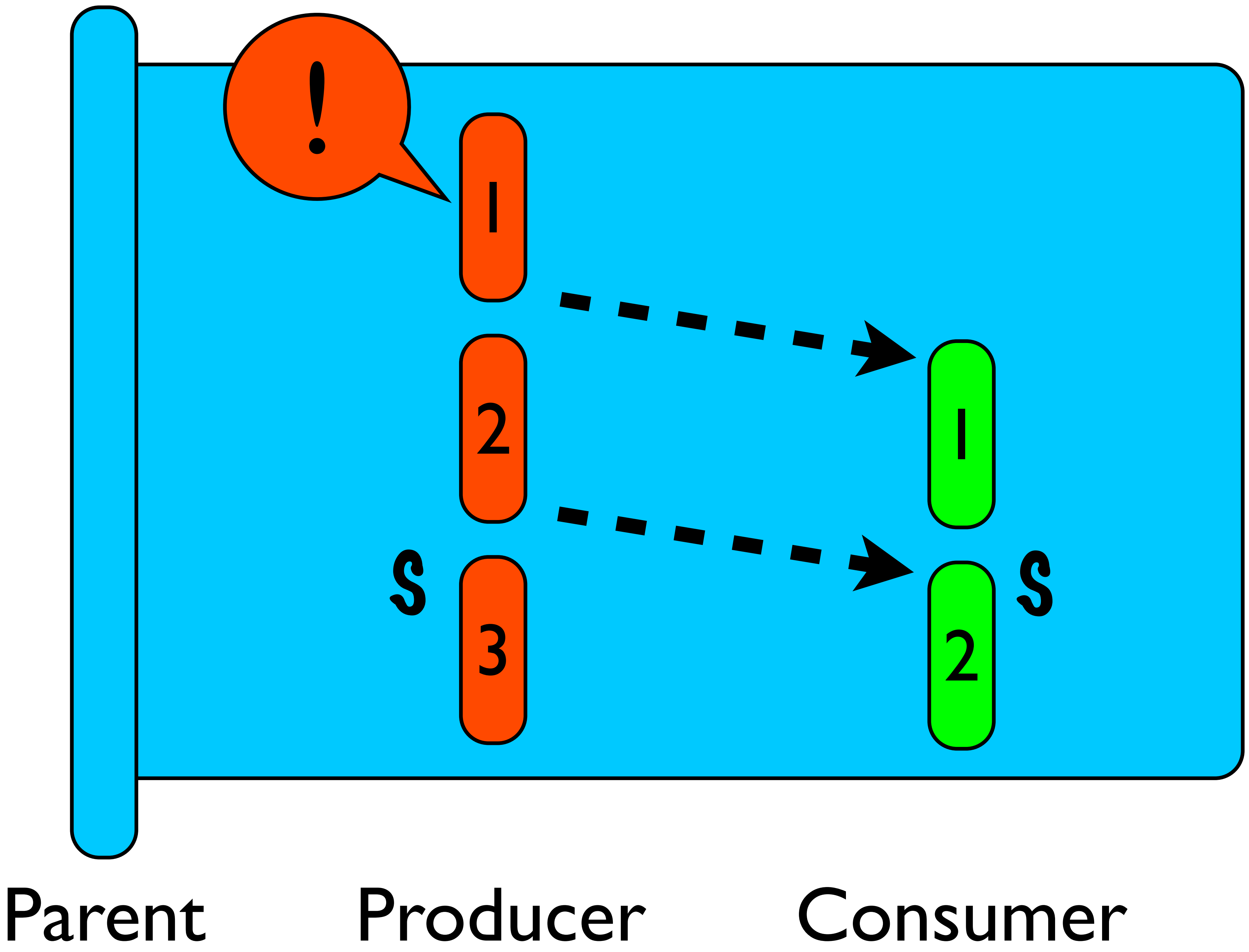
Successors



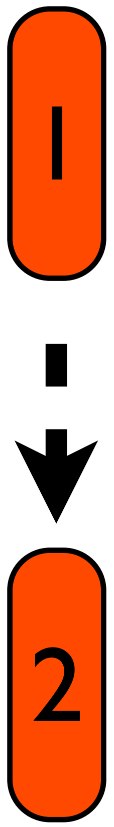
Successors


!
1
2
S
3
1
S
2
Handled here?
Handled here?
Parent
Producer
Consumer

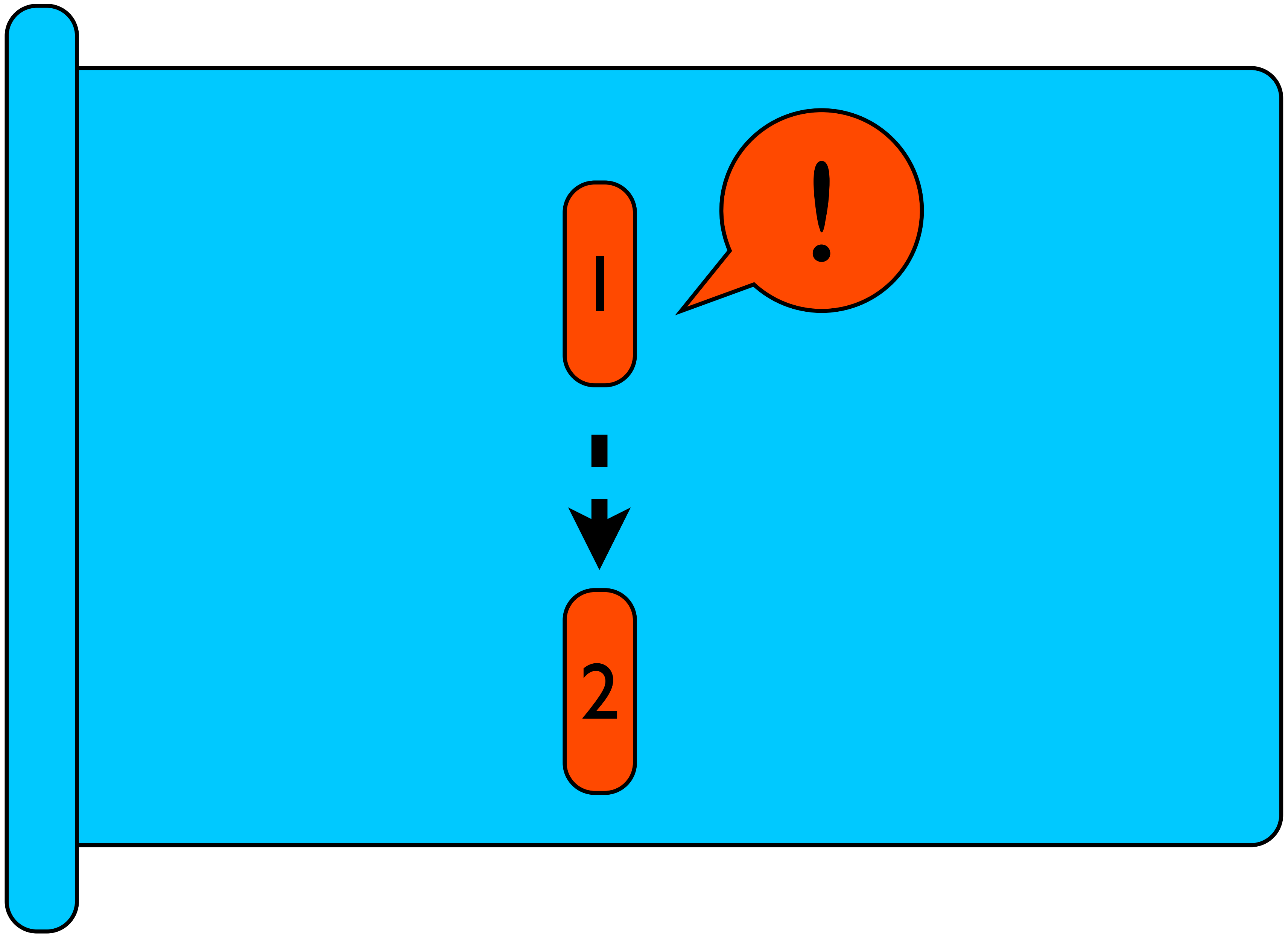
Successors



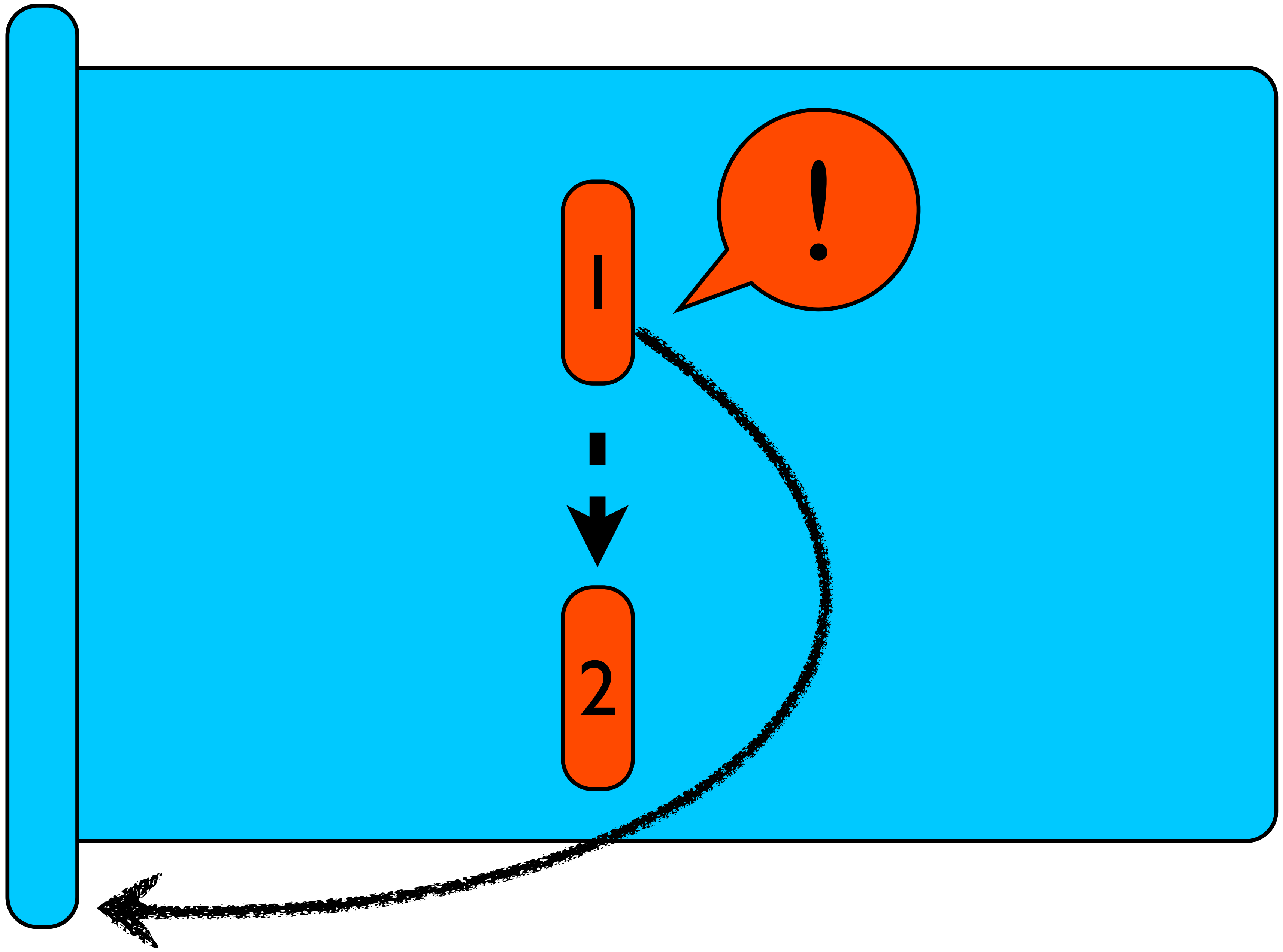
Wrapping



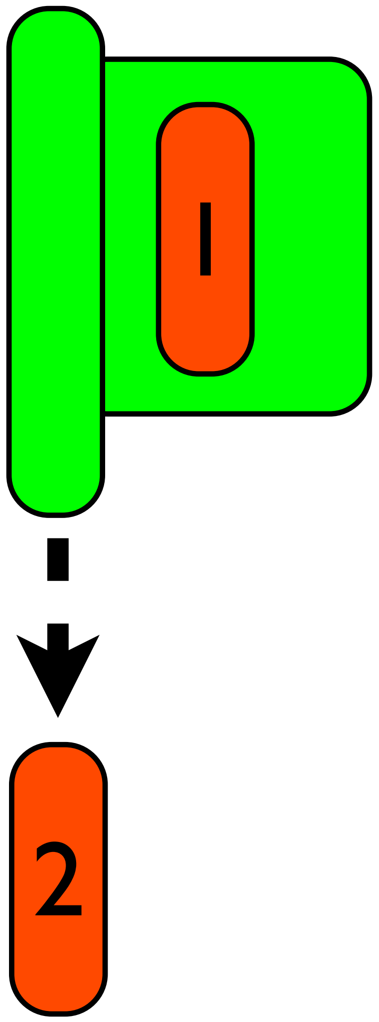
Wrapping



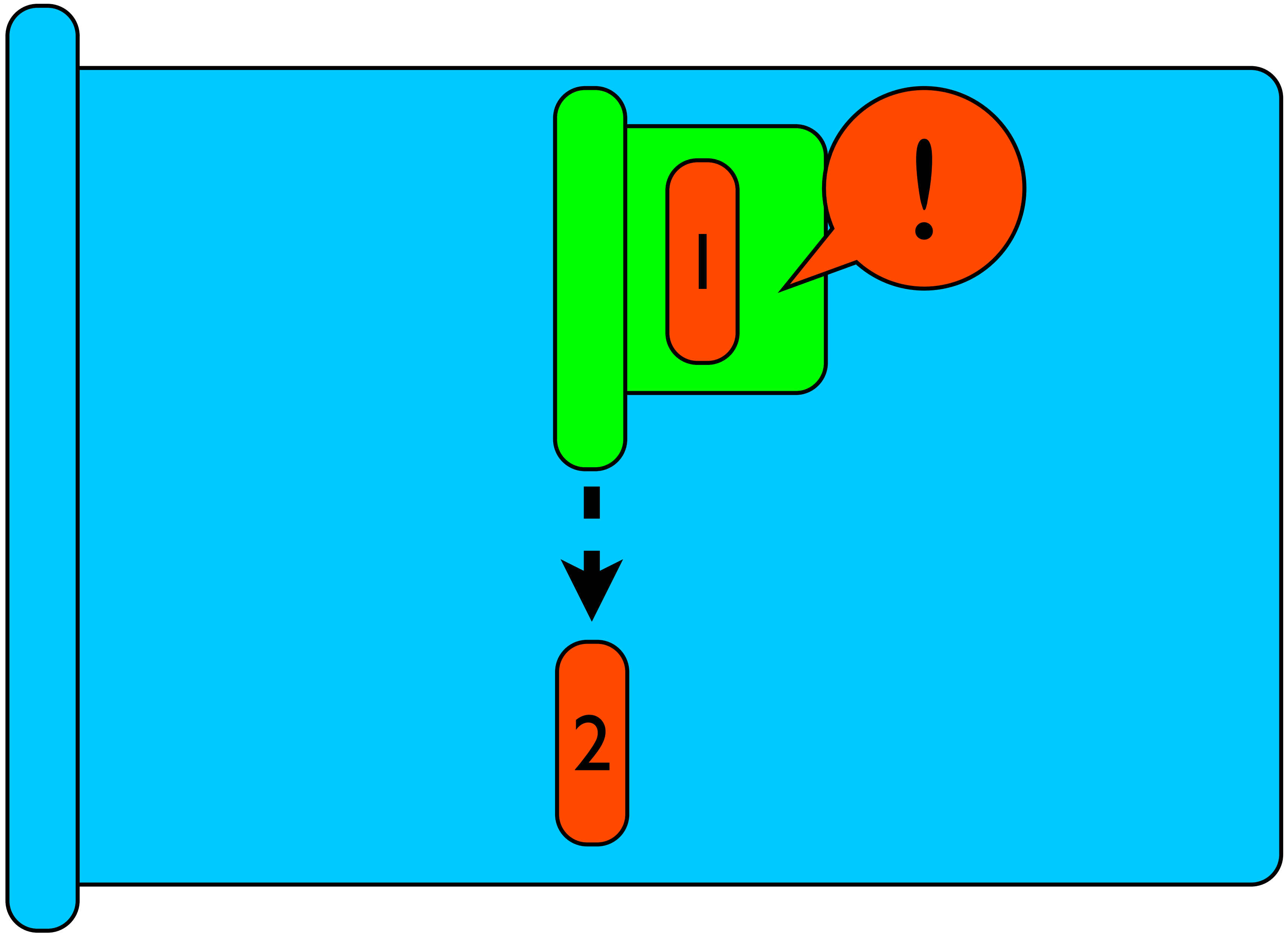
Wrapping



Wrapping



Wrapping



Wrapping