Extended Exception Mechanisms for Contingencies

Thorsten van Ellen, BTC AG

Agenda

1. Terms
2. The Problem and Properties of Contingencies
3. Objectives and Proposed Solution
4. Related Work
5. Future Work

BTC

Discovery of contingencies

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- Implement `allocateMemory` when no more main memory is available?
- Don't terminate? Return invalid memory address? Terminate immediately?

BTC

Discovery of contingencies

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- Implement **`allocateMemory`** when no more main memory is available?
- Don't terminate? Return invalid memory address? Terminate immediately?
- Or communicate the **`OutOfMemory`**-Situation?
- **`OutOfMemory`**: availableMemory < requiredMemory

- Fullfilled, but not specified: behaviour undefined
- Specified in callee, excluded in caller: specification contradiction
- Specified and nothing done: caller postconditions violated

- Specify (postcondition) and „signal“ somehow
- Defined and known before runtime
- Unsatisfactory for continuation, handle!

BTC

Definition Contingency

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- An Error is a situation, the conditions of which contradict the specification.
- Error = specification violation = undefined conditions

- Definition Contingency:
  A Contingency is a situation, that is
  - described within the specification of a module, and
  - represents a module result
  - where the task or function, which calling modules depend on, was not performed.
- Usually perceived as errors/specification violation of the caller, but are not the same because conditions defined
- Indicating could or should not fulfill its usual work, work refusal
- Specified, but unsatisfactory
- Potential appearance known in advance

.

Agenda

1. Terms
2. The Problem and Properties of Contingencies
3. Objectives and Proposed Solution
4. Related Work
5. Future Work

Current recommendations don't work!

Contingencies:

- Happen in the real world
- Disrupt services, ignoring leads to violations
- Conditions exactly known, should be handled normal
- Handle as part of normal code. Should not be regarded as error.
- How should they be communicated and handled?

BTC

Current recommendations don't work!

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Contingencies:

- Happen in the real world
- Disrupt services, ignoring leads to violations
- Conditions exactly known, should be handled normal
- Handle as part of normal code. Should not be regarded as error.
- How should they be communicated and handled?

Common recommendations:

- Exceptions only for errors, e.g., [Meyer88]
- Declare (special values), e.g., [Bloch03] and redeclare, e.g. [Cristian95]
- Abstract implementation details, e.g., [Bloch03]

Where is the problem?

- Contingencies are omnipresent, accumulate, are implementation dependent and must not be abstracted

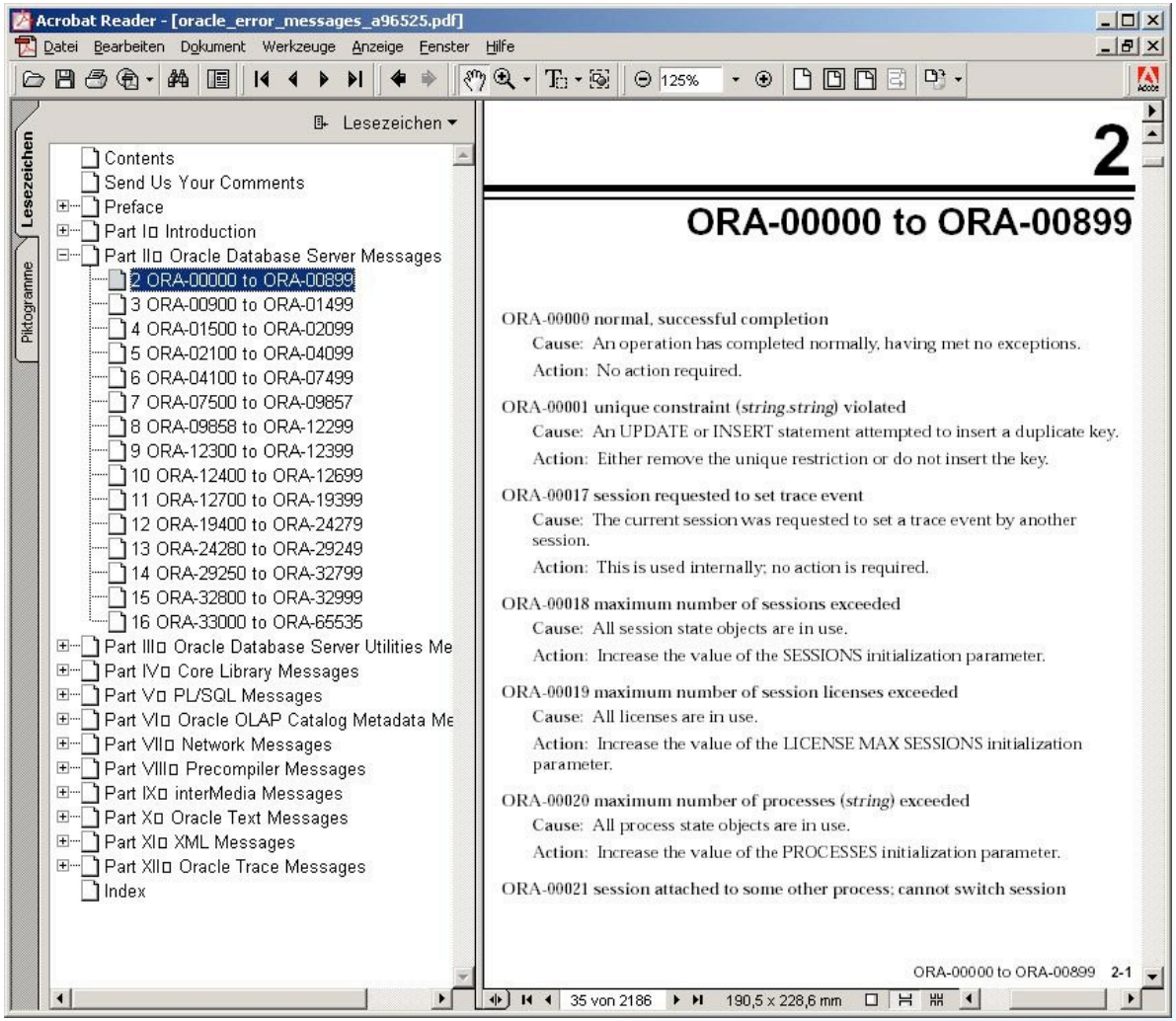
Only few? What can happen, when calling ...

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- OpenFile?
  FileNotFound, DirectoryNotFound, DriveNotFound, FileReadOnly, DirectoryReadOnly, DriveReadOnly (HW-Switch), FileNameInvalid, DirectoryNameInvalid, DriveNameInvalid, FileLocked, DriveLocked, MediaNotInserted, MediaNotFormatted, DiskFull, NoAvailableFileHandles, EndOfFile, NetworkDisconnected, QuotaOverflow, DiskEjected, DiskIOError, ...

- ExecuteSQLStatement?
  ColumnNotFound, TableNotFound, SchemaNotFound, DatabaseNotFound, InvalidColumnName, InvalidTableName, InvalidSchemaName, InvalidDatabase, RecordLocked, TableLocked, TableSpaceFull, ...

Systematic problem: contingencies are omnipresent

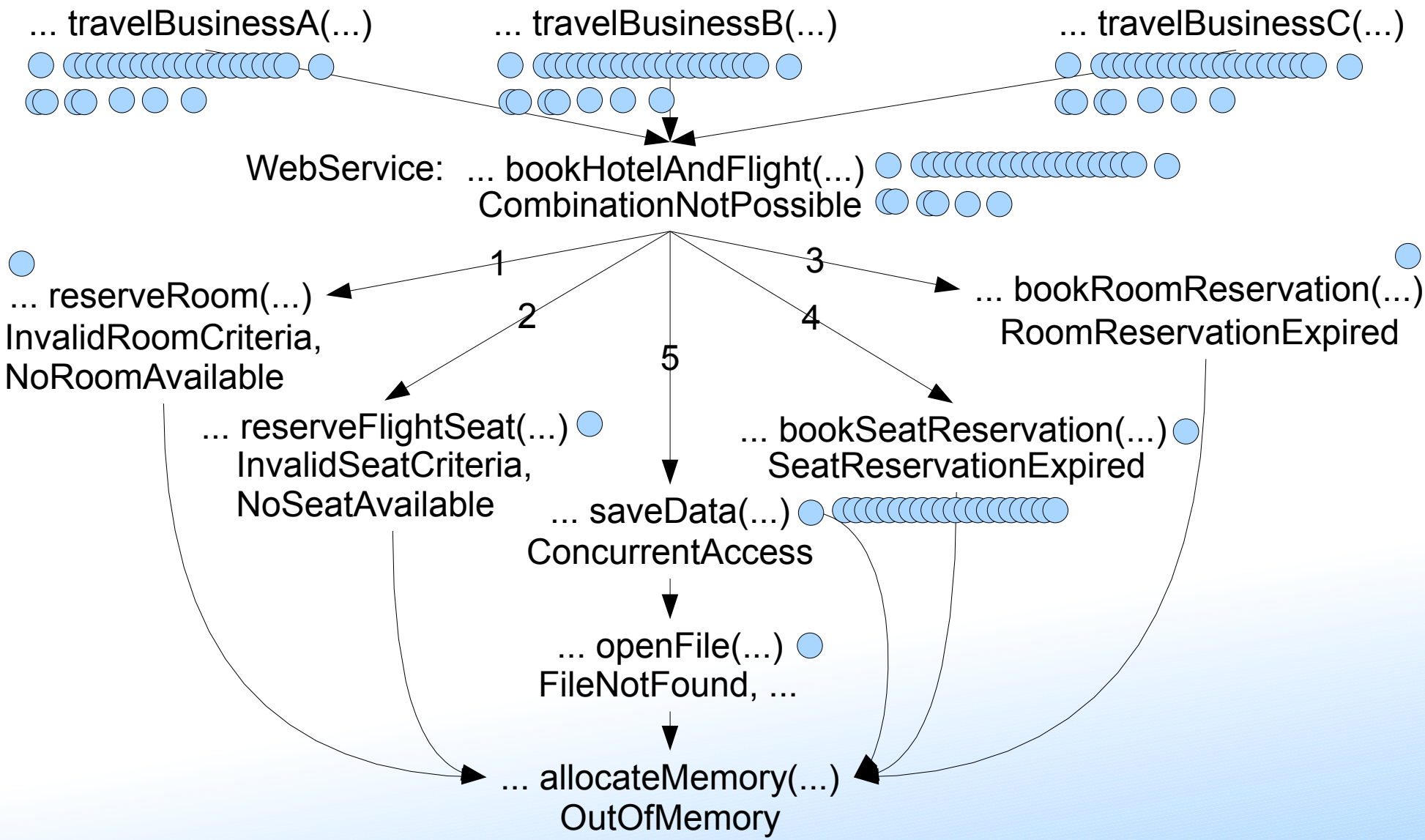


http://download.oracle.com/docs/cd/B10501_01/server.920/a96525.pdf

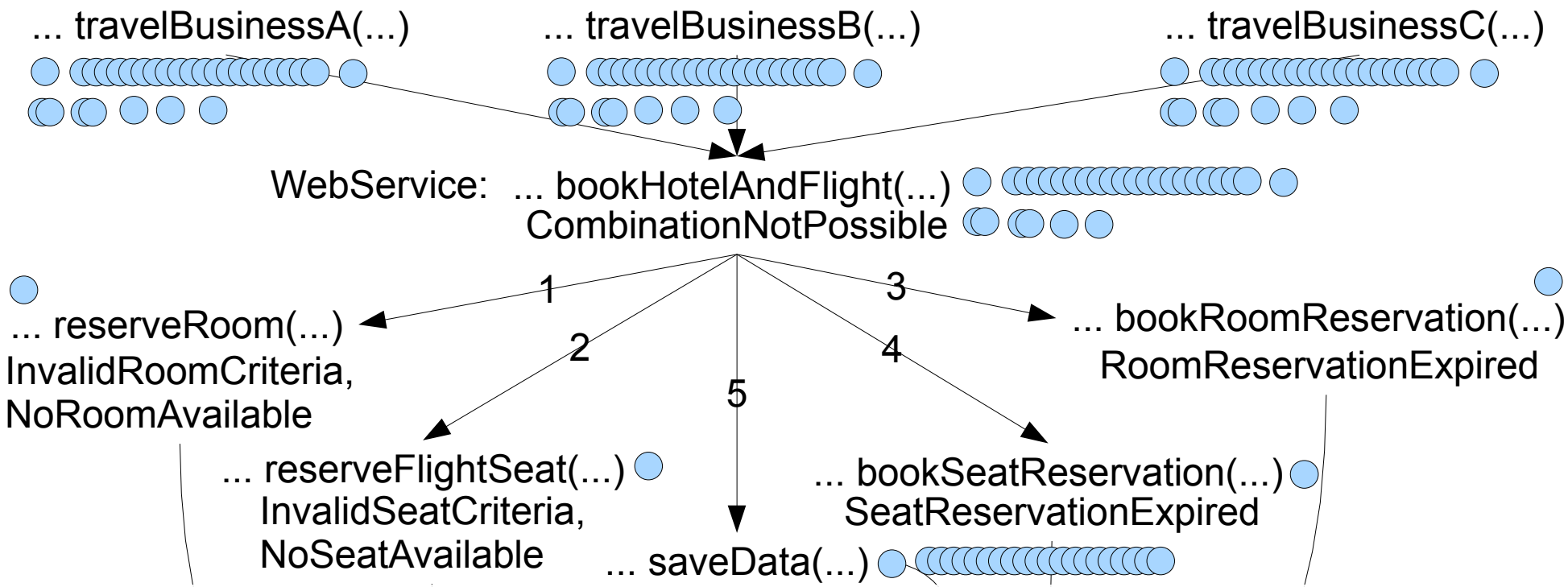
- Substantial amounts are contingencies cleanly intercepted and documented at development time

Systematic problem: contingencies accumulate

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

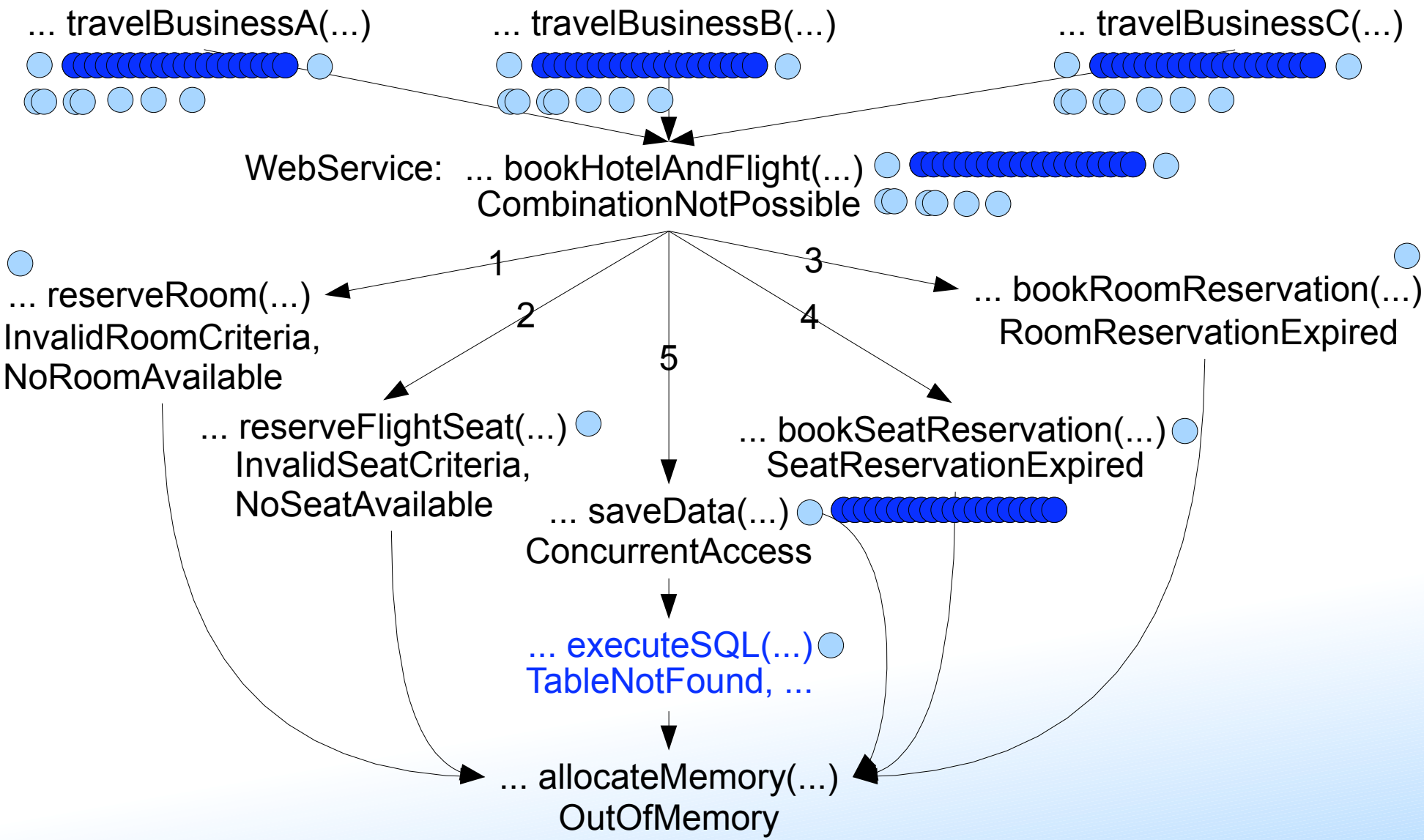


Systematic problem: contingencies accumulate

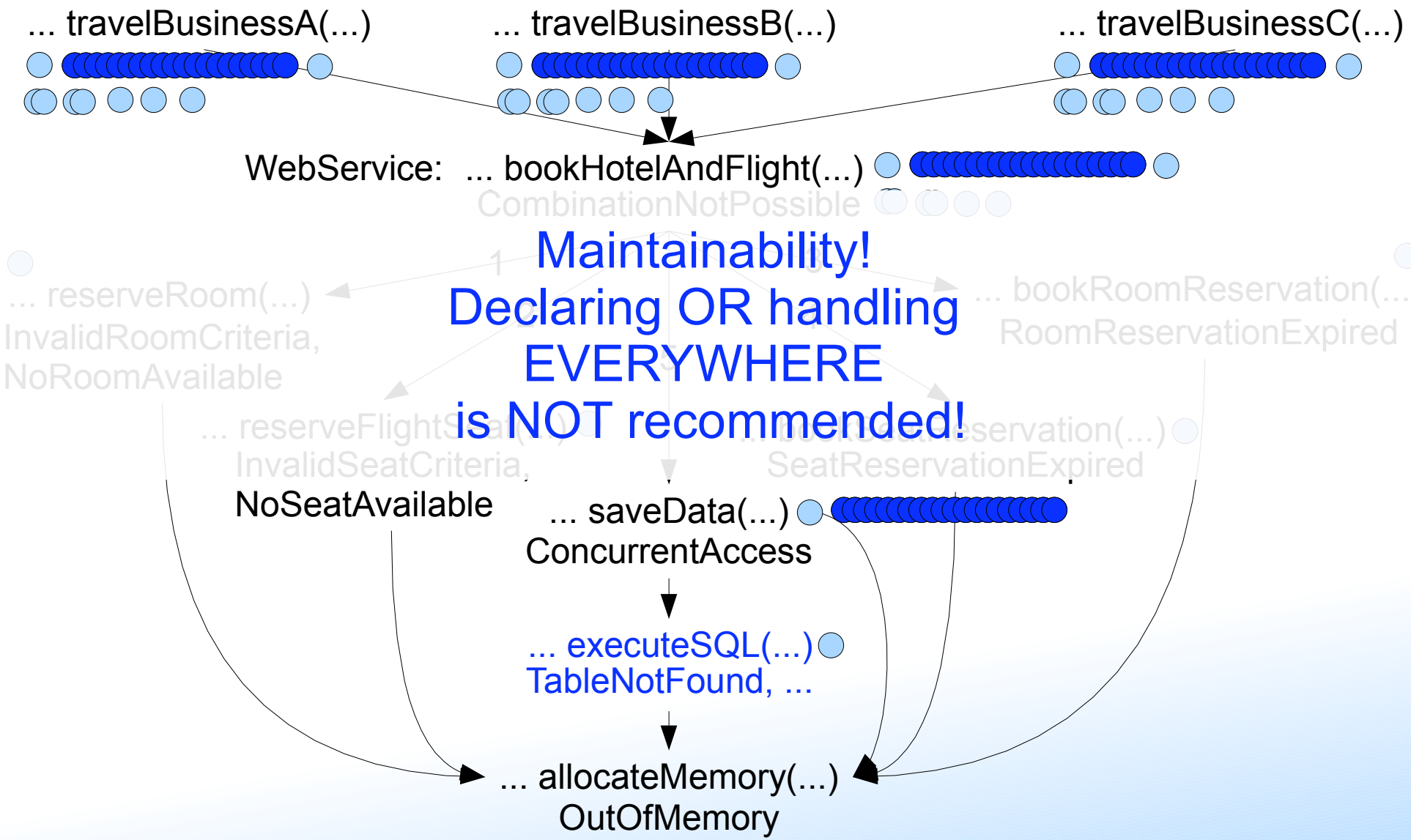


Redundancy!
Declaring OR handling
EVERYWHERE
is NOT recommended!

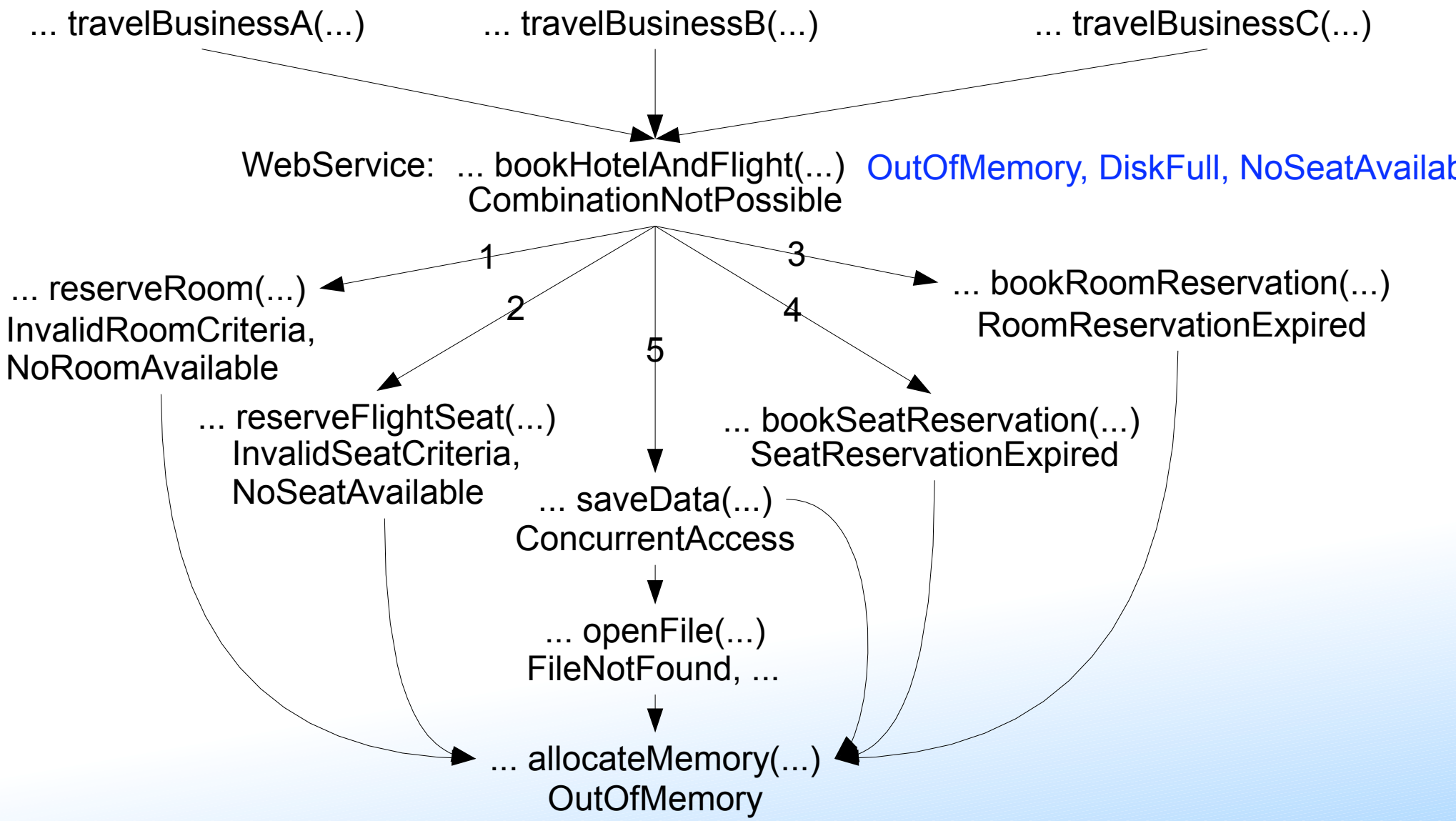
Systematic problem: implementation dependency



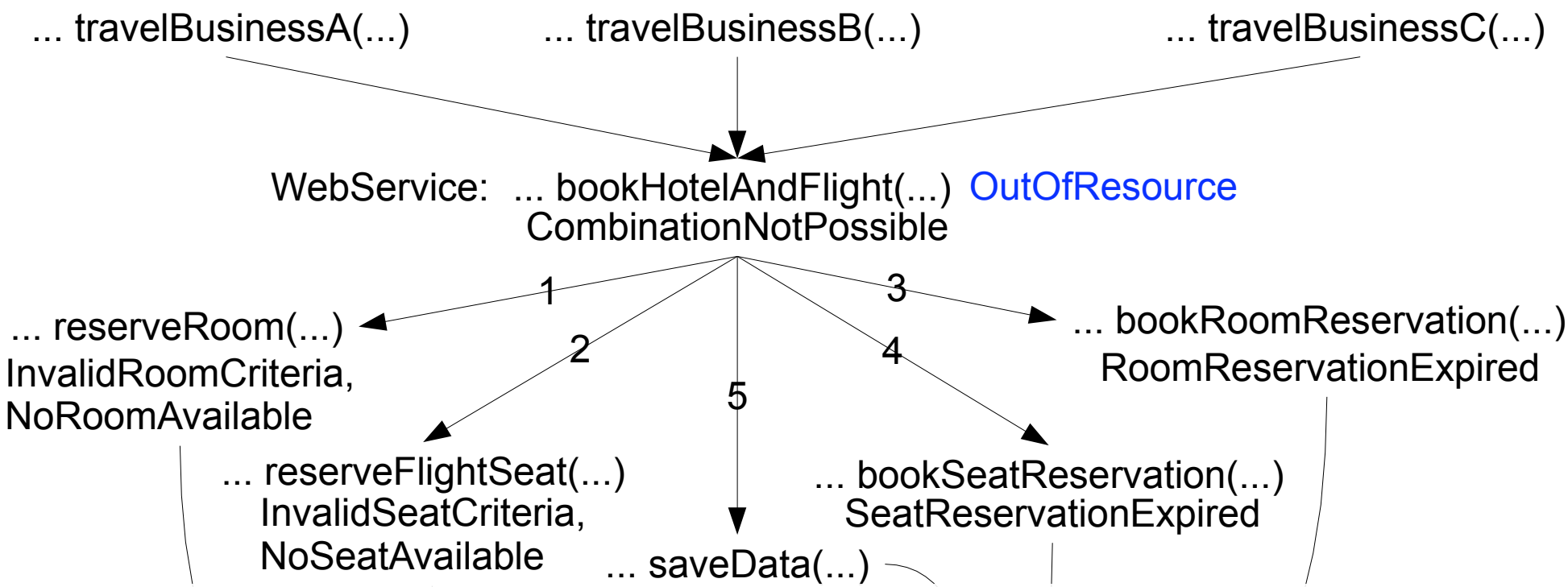
Systematic problem: implementation dependency



Systematic problem: abstraction of implementation is no solution



Systematic problem: abstraction of implementation is no solution



Solution for OutOfMemory: swapping on disk
Solution for DiskFull: select other disk, ...
Solution for OutOfResource?
Abstracting NOT recommended! React specifically!

Agenda

1. Terms
2. The Problem and Properties of Contingencies
3. Objectives and Proposed Solution
4. Related Work
5. Future Work

BTC

Objectives

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- Distinguish contingencies
- Simple communication between the call levels
- Determine automatically at development time
- Get complete documentation of all levels
- Enable forward recovery (ascertaining, repairing, continuing)
  - Even for physical side effects
  - Avoid complications of conventional interfaces
- Enable overriding any handling by outer context with broader knowledge and component access
- Keep information hiding as far as possible

BTC

Overview of solution

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Deficiencies of conventional exception mechanisms

- Ascertaining
- Repairing
- Continuing
- Overriding

Proposed Solution

- Ascertaining interactively
- 3-fold separation (coined by Common Lisp [Pitman90])
- Reversed search order

Handle them, but which?

- Ascertain to handle
- Do you know the contingencies of your last program?
- Can you enumerate them completely?
  E.g., where can DiskFull happen? Why Not?

BTC

Handle them, but which?

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- Ascertain to handle
- Do you know the contingencies of your last program?
- Can you enumerate them completely?
  E.g., where can DiskFull happen? Why Not?

- Conventionally, cannot be ascertained before runtime
- Repair possibilities cannot be applied purposefully before runtime.
- Instead manually in interface, automatic ascertaining
- To be ascertainable contingencies must be marked in the source

Repair by access to implementation details

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Example:

- „DiskFull“ while saving data on disk,
- Regardless, where the problem occurred, raise exception multiple levels
- Let user select alternative path,
- Repair work refusal in lower level, where the problem occurred

BTC

Cooperation and repair of levels

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

```
void main(String[] args) { // Within higher layer with GUI-access
      try {
         doTasks(args);
      } catch (HardDiskFull) {
         String alt = AskForAlternativePath.execute().result();
         // set/repair currenPath to alternative, but how/where?
      }
}

void saveEditedData(Data edited) throws HardDiskFull { // No GUI
      if (getFree(currentPath) < edited.size()) {

            throw new HardDiskFull();



      } // ... writing data on disk
}
```

BTC

Cooperation and repair of levels

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

```
void main(String[] args) { // Within higher layer with GUI-access
      try {
          doTasks(args);
      } catch (HardDiskFull) {
          String alt = AskForAlternativePath.execute().result();
          solve selectAlternativePath(alt);
      }
}

void saveEditedData(Data edited) throws HardDiskFull { // No GUI
      if (getFree(currentPath) < edited.size()) {
          try {
              throw new HardDiskFull();
          } offer selectAlternativePath(String alternative) {
              currentPath = alternative;
          }

      } // ... writing data on disk
}
```

BTC

Continuing after solution of contingency?

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- Example:
  - Loop over some files
  - Work refusal occurs
  - Good solution exists, but not redundant
  - Raise exception multiple levels
  - Leave loop and unwind stack
  - Solve contingency
- And then? Continuation! But how?
- Which line of the try-block? Try for every line?
- Transactions? For physical side effects?
- Compensation? For ignition of the 2nd propulsion stage?
- Publish internal partial state[Miller97]? Sacrifice information hiding? Interface complications everywhere?
- Additional effort and cost before and at runtime: kind of damage!

BTC

Continuing after solution of contingency?

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- Example:
  - Loop over some files
  - Work refusal occurs
  - Good solution exists, but not redundant
  - Raise exception multiple levels
  - Leave loop and unwind stack
  - Solve contingency
- And then? Continuation! But how?
- Which line of the try-block? Try for every line?
- Transactions? For physical side effects?
- Compensation? Programmer must provide code
- Publish internal partial state? Violation of information hiding? Interface complications everywhere?
- Additional effort and cost before and at runtime: kind of damage!

The simplest and most efficient solution(!):
do NOT unwind the stack and
resume at a suitable place:
(Special) resumption!

BTC

Override handling?

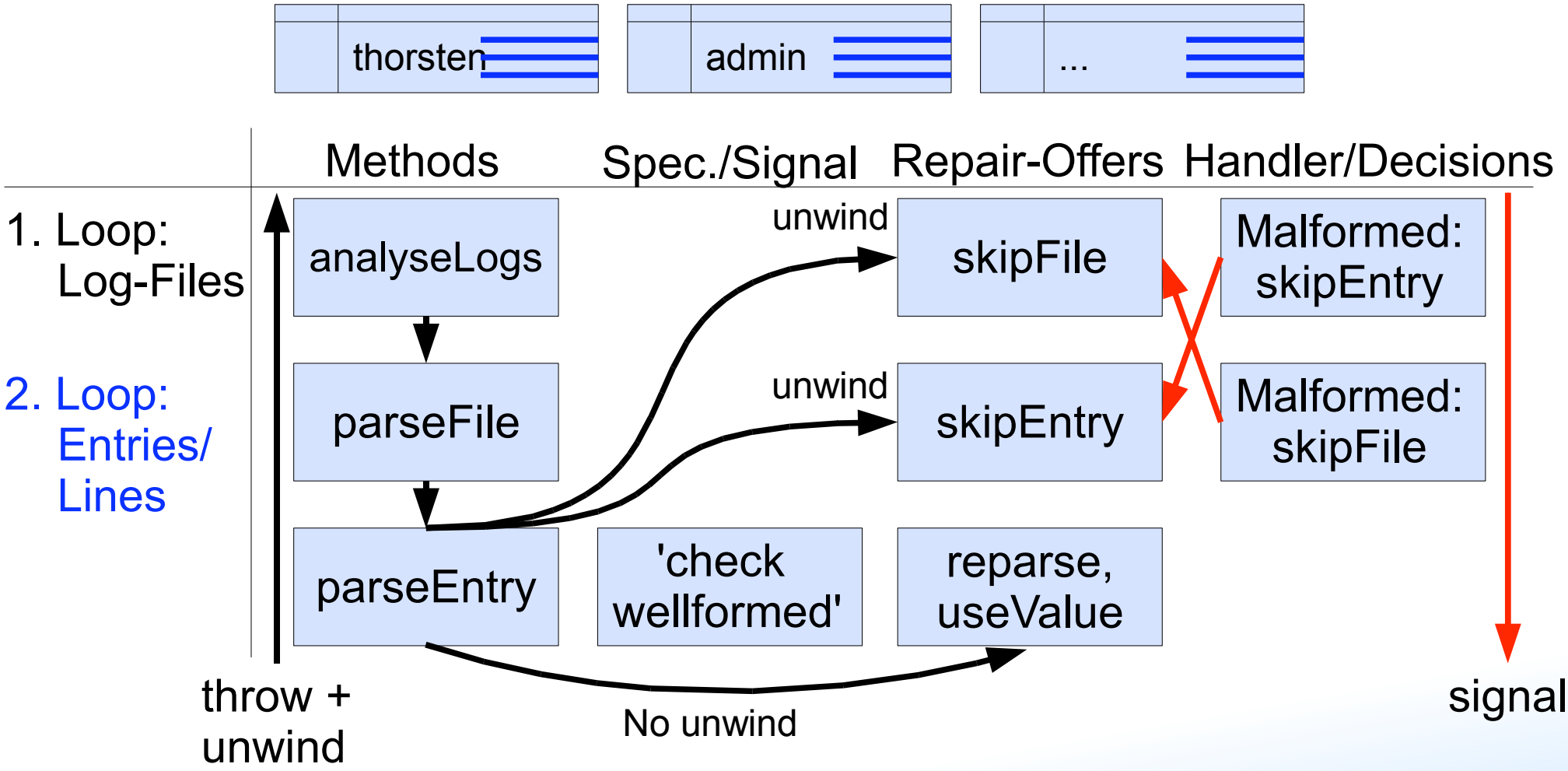
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

```
public void print(Object document, int fromPage) {
      try {
         // print ...
      } catch (PaperJamDetected jamDetected) {
         // Default problem handling: cancel and logging
         logger.error("Paper jam occured: aborting print.");
      }
}
public void printAdvanced (Object document, int fromPage) {
      try {
         print(document, fromPage);
      } catch (PaperJamDetected jamDetected) {
         // Advanced problem handling: automatically repair
         advancedPaperEmitter.removeJam();
         printAdvanced(document, jamDetected.atPage());
      }
}
```

Not possible!

- Handling can never be overridden, neither for third party nor for own code!

Example for 3-fold separation: read log-files



Lower do not catch anything away from higher
higher can change lower
(“Polymorphism” along the stack)

Example: Idea partly Peter Seibel, Practical Common LISP http://www.gigamonkeys.com/book/beyond-exception-handling-conditions-and-restarts.html

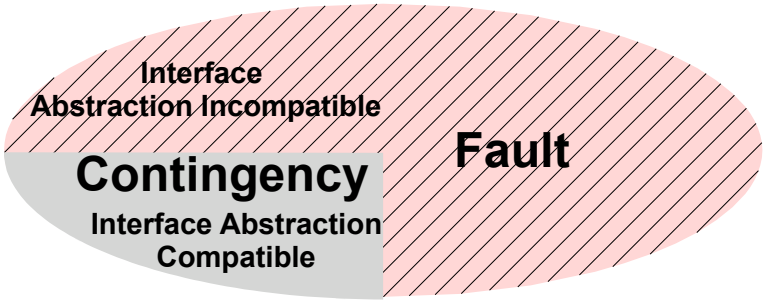
Agenda

1. Terms
2. The Problem and Properties of Contingencies
3. Objectives and Proposed Solution
4. Related Work
5. Future Work

Related Work

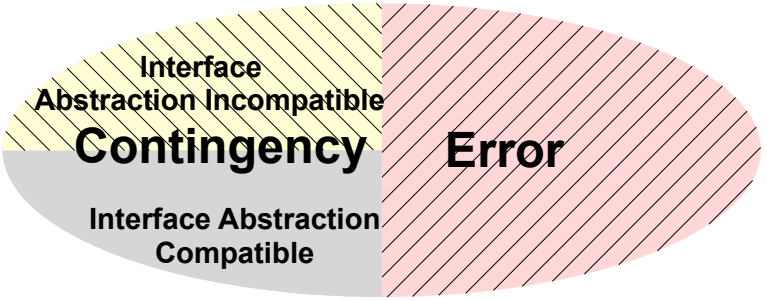
Ruzek's Contingencies [Ruzek07]

„expressed in terms of intended purpose not in terms of implementation“



Our Contingencies

„work refusals, defined conditions, regardless of level“



Domain Partition [Cristian95]:
Relation of results to its prediction by the specification for the input

Standard Domain	Anticipated Exception Domain
Failure Domain	Unanticipated Domain („Specification Failure“)

Related Languages and Mechanisms

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Indispensable features:

1. Ascertain and interactively choose (exists nowhere, yet)
2. Dynamic search top-down (exists nowhere, yet)
3. Resumption at arbitrary level
4. Repair of afflicted implementation details
5. Multiple „offer“ at the same level
6. Exception Safety

At:	Supported:					
	1.	2.	3.	4.	5.	6.
Extended Exception Mechanisms	X	X	X	X	X	
Common Lisp [Pitman90]:			X	X	X	
Smalltalk [Meyer88]:				X		
Closures:				X	X	
Callbacks [Gruler05]:					X	

Agenda

1. Terms
2. The Problem and Properties of Contingencies
3. Objectives and Proposed Solution
4. Related Work
5. Future Work

BTC

Future Work

Institut für Informatik
Christian-Albrechts-Universität zu Kiel

- Formalization: set based, similar to [Cristian95]
- Open Source Implementation for Java: extension of
  - Syntax
  - Compiler
  - Standard lib
- Participants are welcome!
- Empirical research with student software practical to explore
- Proposal for the Java Standard (JSR):
  - Expert group according to JCP
  - Extension of Specification
  - Extension of Test suite
- Participants are welcome!

Summary

- Treat contingencies systematically
- New recommendations required
  - Don't declare implementation details, use exceptions
  - Don't abstract and treat specifically
- Extended mechanisms required (3-fold separation)
  - Ascertain interactively at development time
  - Don't unwind immediately
  - Repair and continue
  - Override inappropriate handling
- New mechanisms maybe be of great help for safety critical applications

References

- A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput., 1(1):11–33, 2004.
- J. Bloch. Effective Java Programming Language Guide. Mountain View, CA, USA,Sun Microsystems, Inc., 2001
- F. Cristian. Exception handling and tolerance of software faults. In M. R. Lyu, editor, Software Fault Tolerance, pages 81-107. John Wiley & sons, 1995.
- A. Gruler and C. Heinlein. Exception handling with resumption: Design and implementation in Java. In PLC, pages 165-171, 2005.
- B. H. Liskov and A. Snyder. Exception handling in CLU. IEEE Trans. Softw. Eng., 5(6):546-558, 1979.
- B. Meyer. Object-Oriented Software Construction. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
- R. Miller and A. Tripathi. Issues with Exception Handling in Object-Oriented Systems. In LNCS 1241, pages 85-103, 1997.
- K. M. Pitman. Exceptional situations in Lisp. In Proceedings for the First European Conference on the Practical Application of Lisp (EUROPAL'90), Cambridge, UK, 1990.
- B. Ruzek. Effective java exceptions. dev2dev.bea.com, January 2007. http://www.oracle.com/technology/pub/articles/dev2arch/2006/11/effective-exceptions.html.