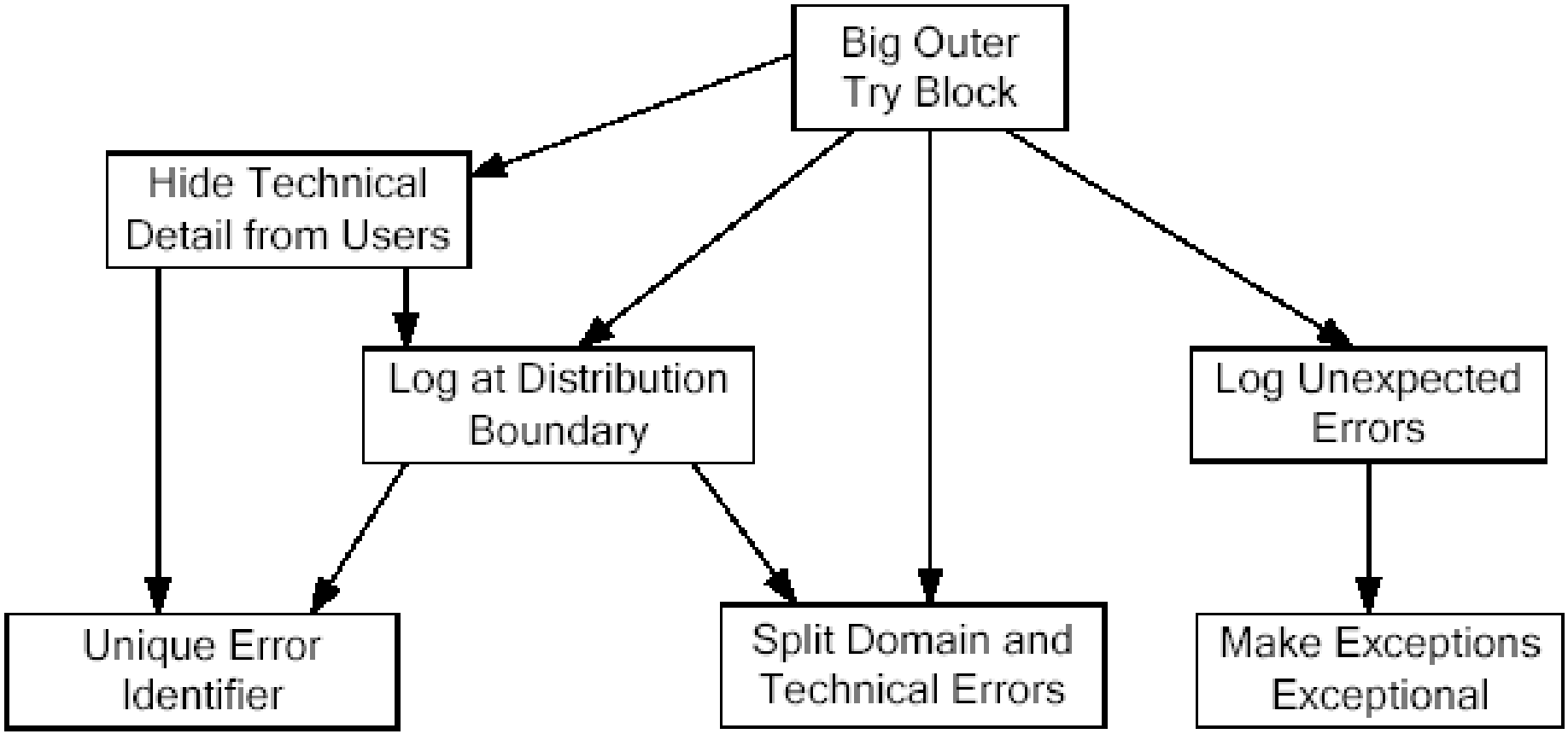
Error Handling

OOA/OOD

xuyingxiao@126.com

Error Handling Patterns

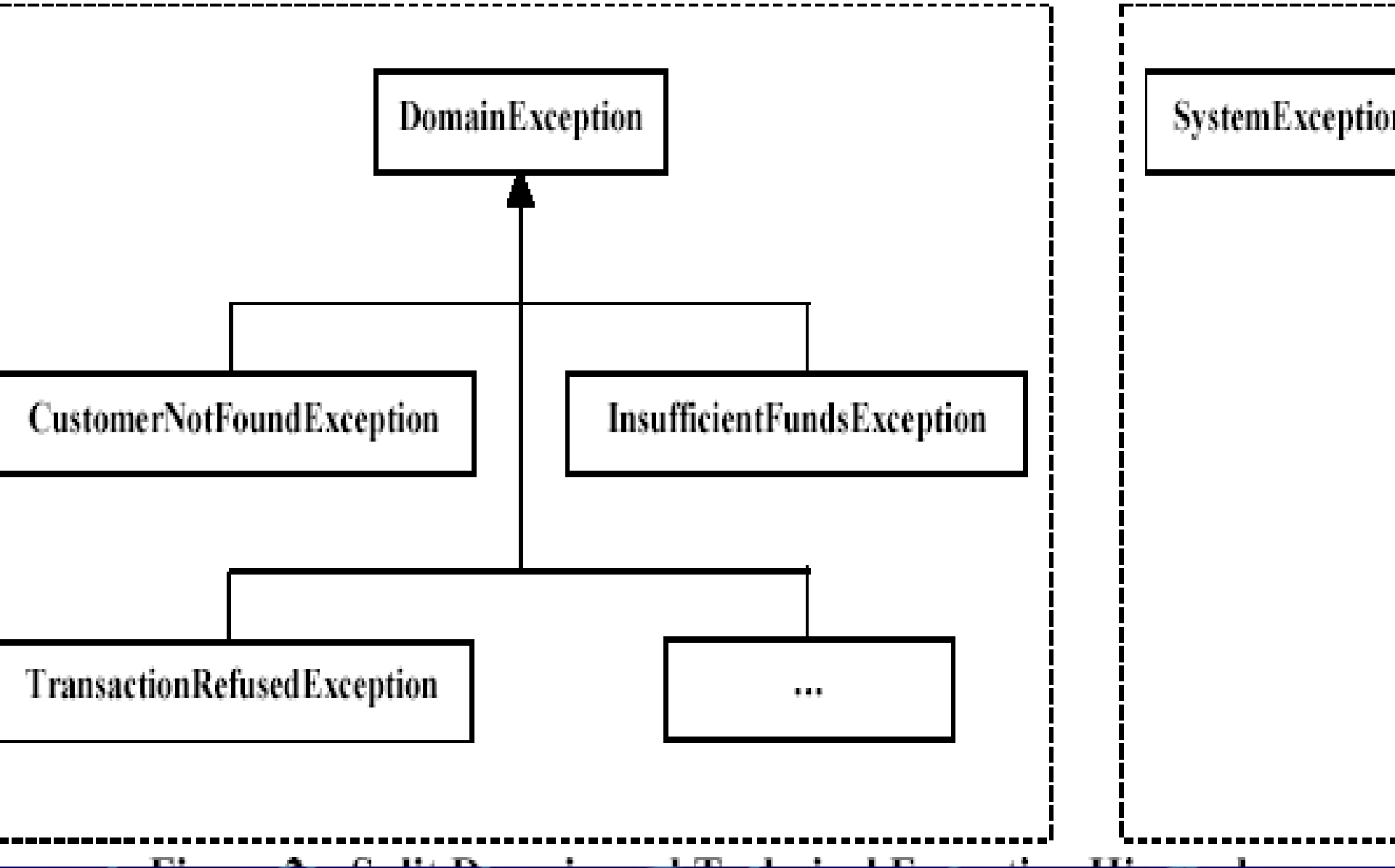


Split Domain and Technical Errors

- “domain errors”, are due to errors in the business logic or business processing (e.g. wrong type of customer for insurance policy).
- “technical errors”, are caused by problems in the underlying platform (e.g. could not connect to database) or by unexpected faults (e.g. divide by zero).

- Handling technical errors in domain code makes this code more obscure and difficult to maintain.

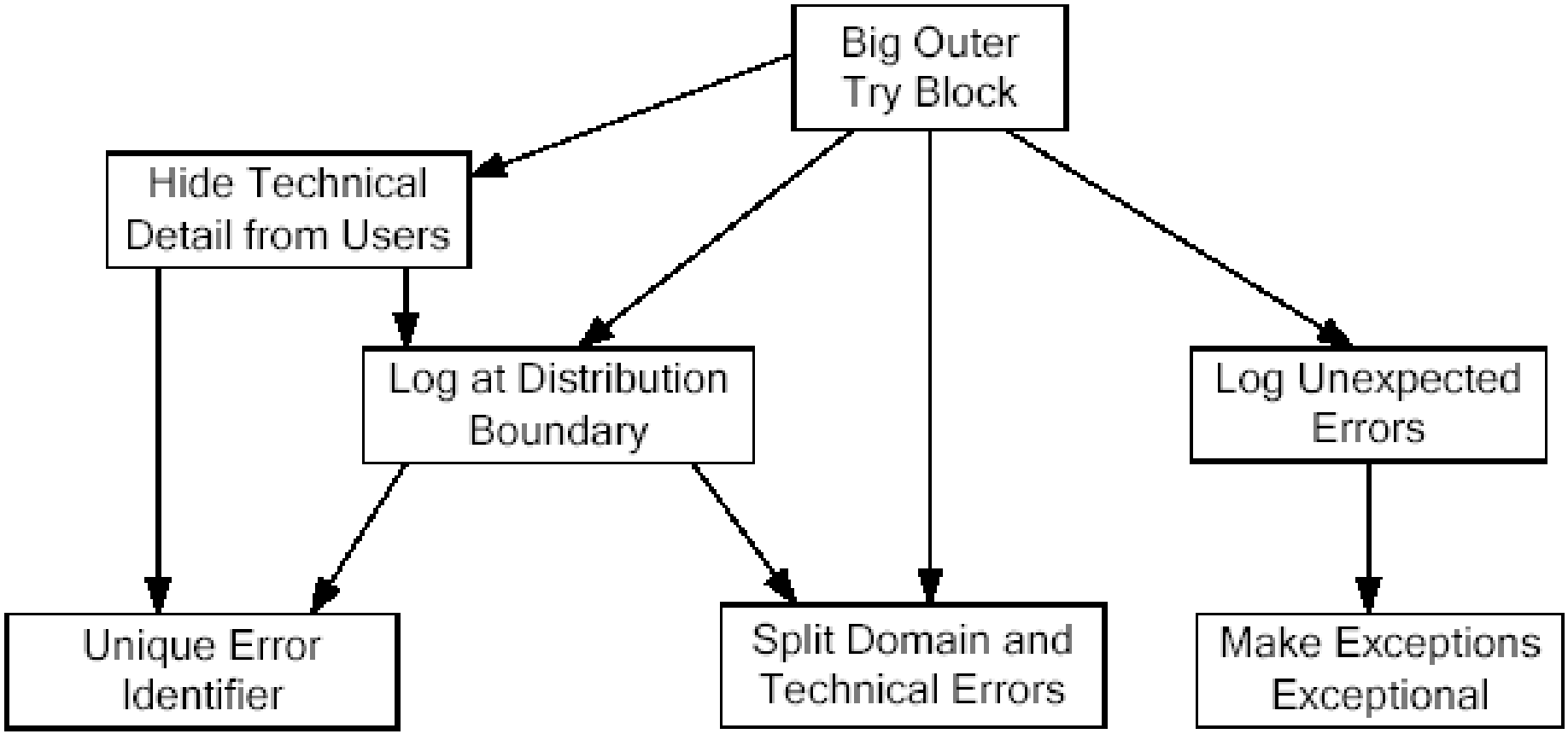
- ***Solution***
- Split domain and technical error handling. Create separate exception/error hierarchies and handle at different points and in different ways as appropriate.


DomainException
SystemException
CustomerNotFoundException
InsufficientFundsException
TransactionRefusedException
...

- public class DomainException extends Exception
- {
- ...
- }
- Public class InsufficientFundsException extends …Exception
- {
- ...
- }
- public class SystemException extends Exception
- {
- ...
- }

```
public float withdrawFunds(float amount)
throws InsufficientFundsException, SystemException
{
        try
        {
//              Domain code that could generate various errors
//              both technical and domain
        }
        catch (DomainException ex)
        {
                throw ex;
        }
        catch (Exception ex)
        {
                throw new SystemException(ex);
        }
}
```

Error Handling Patterns



Big Outer Try Block

- Errors will propagate right to the edge of the system and will appear to “crash” the application if not handled at that point.
- Users are mostly on remote sites and will not do much to report errors

- ***Solution***

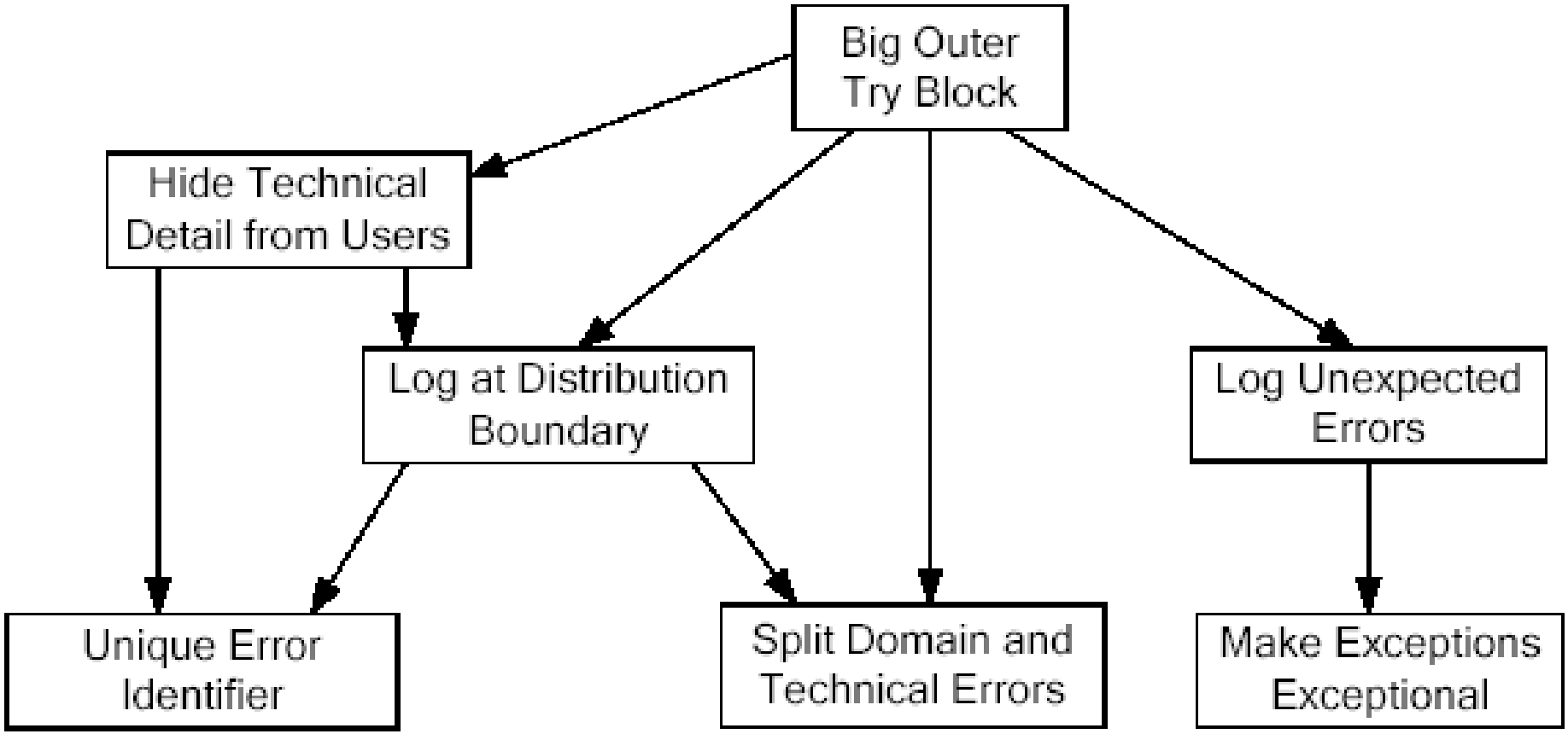
- Implement a *Big Outer Try Block* at the "edge" of the system to catch and handle errors that cannot be handled by other tiers of the system. The error handling in the block can report errors in a consistent way at a level of detail appropriate to the user constituency.

```
public class ApplicationMain {
        ...
        public static void main(String[] args) {
                try{
                        ApplicationMain m = new ApplicationMain() ;
                        m.initialize() ;
                        m.execute() ;
                        m.terminate() ;
                }
                catch(AppDomainException de) {
                        //  Domain exceptions shouldn't get to this level as
                        // they should be handled in the user interface. If
                        // they get here, report the text to the user and
                        // log them in a local log file

                }
                catch(AppTechnicalException te) {
                        // Technical exceptions here are probably user interface
                        // problems. Display a generic apology and log to a
                        // local log file

                }
                catch(Throwable t) {
                        // Other exception objects must be internal errors
                        // that could not be caught and handled elsewhere.
                        // Display a generic apology and log to a local log file
                }
        }
```

Error Handling Strategies

	Expected	*Unexpected*
Domain	• Handle in the application code • Display details to the user • Don't log the error	• Throw an exception • Display details to the user • Log the error
Technical	• Handle in the application code • Don't display details to the user • Don't log the error	• Throw an exception • Don't display details to the user • Log the error

Error Handling Patterns

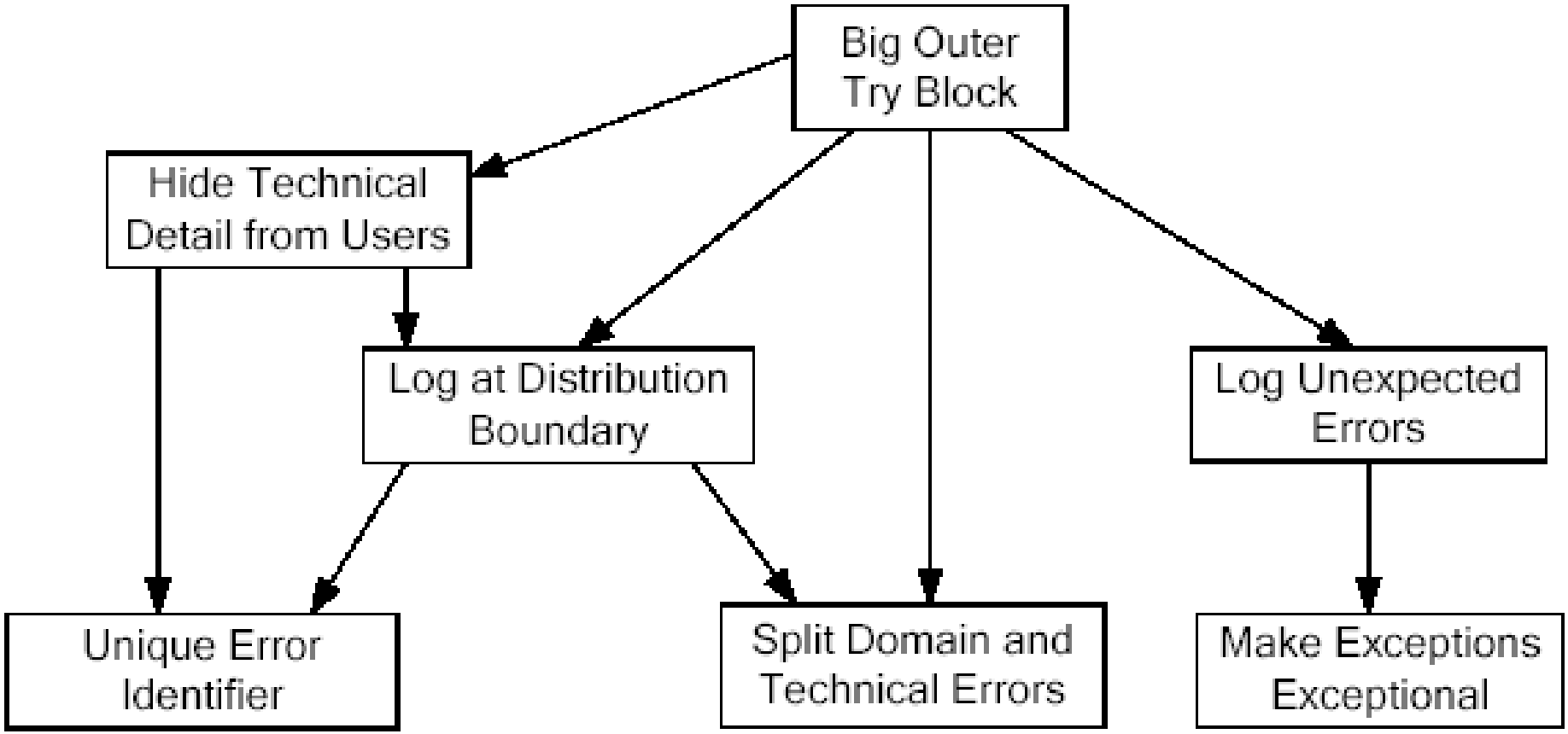


Log at Distribution Boundary

- The details of technical errors rarely make sense outside a particular, specialized, environment where specialists with appropriate knowledge can address them.
- Multi-tier systems, particularly those that use a number of distinct technologies in different tiers.

- ***Solution***
- When technical errors occur, log them on the system where they occur passing a simpler generic SystemError back to the caller for reporting at the end-user interface.
- The generic error lets calling code know that there has been a problem so that they can handle it but reduces the amount of system-specific information that needs to be passed back through the distribution boundary.

Error Handling Patterns



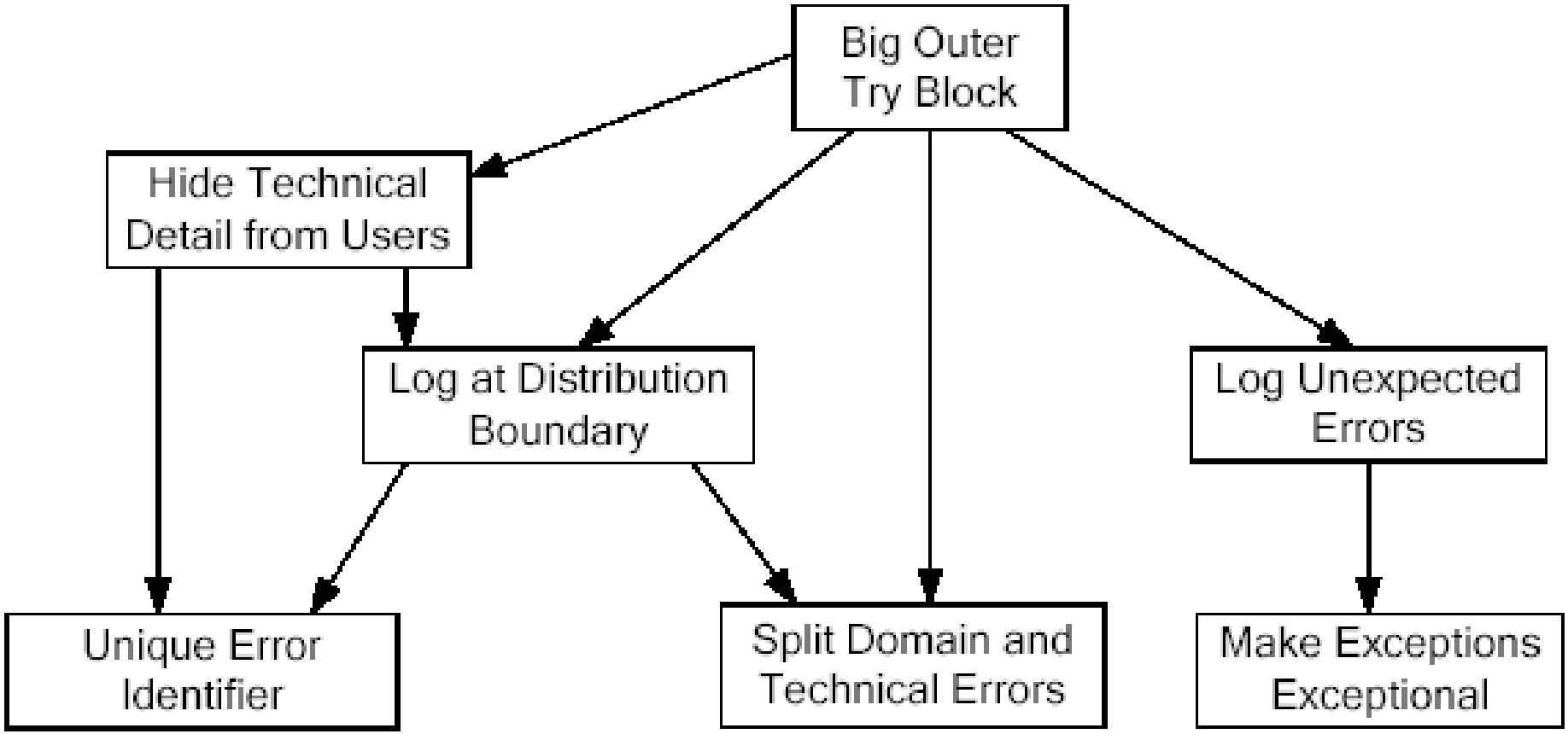
Unique Error Identifier

- If an error on one tier in a distributed system causes knock-on errors on other tiers you get a distorted view of the number of errors in the system and their origin.

- ***Solution***
- Generate a Unique Identifier when the original error occurs and propagate this back to the caller.
- Always include the Unique Identifier with any error log information so that multiple log entries from the same cause can be associated and the underlying error can be correctly identified.

- ***Implementation***
  - uniqueness of the error identifier
    - space
    - time
    - integrity
  - consistency with which it is used in the logs.

Error Handling Patterns



Hide Technical Error Detail from Users

- The technical details of errors that occur are typically of no interest to the end-users of a system.
- If exposed to such users, this error information may cause unnecessary concern and support overhead.

Solution

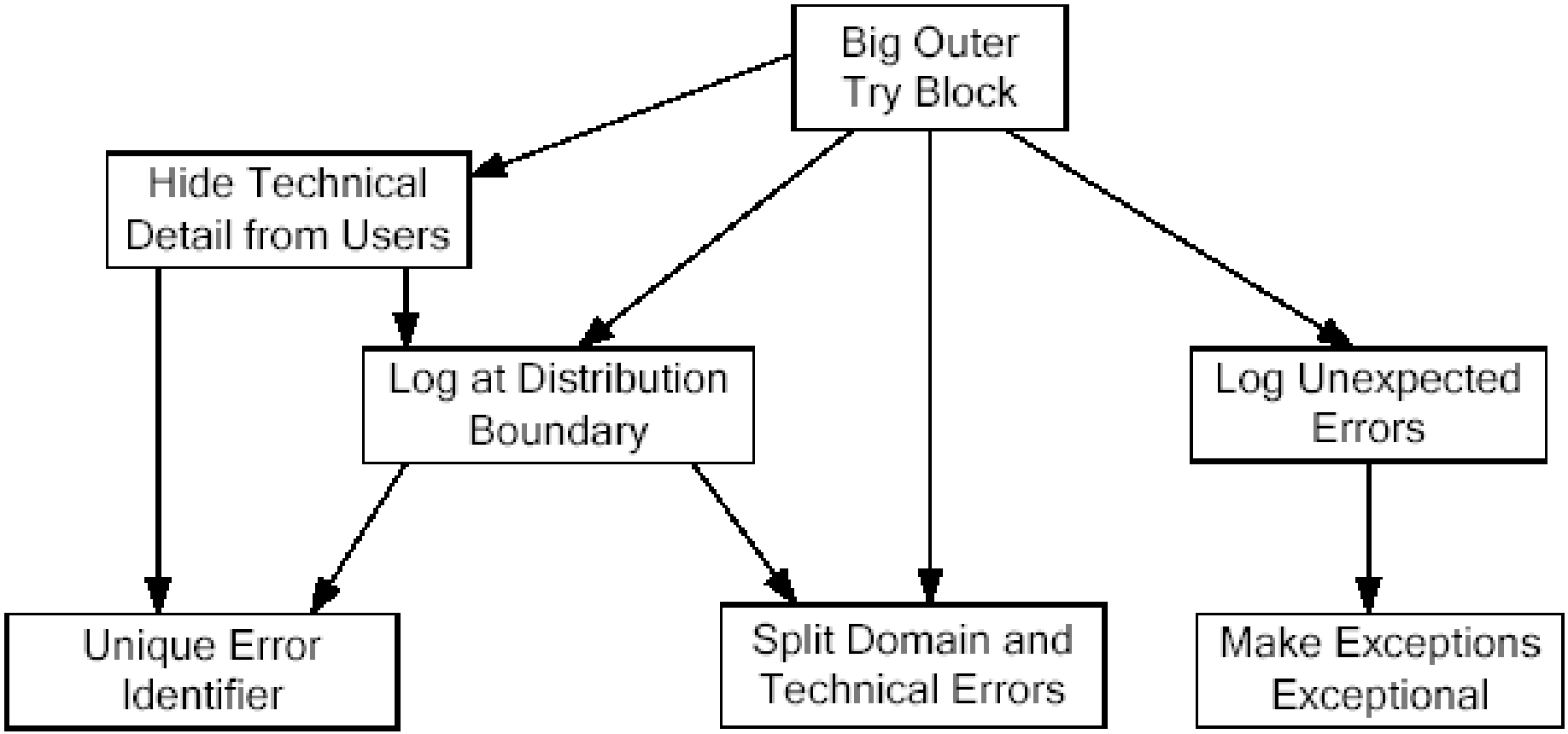
- Implement a standard mechanism for reporting unexpected technical errors to end-users.
- The mechanism can report all errors in a consistent way at a level of detail appropriate to the different user constituencies who need to be informed about the error.

Implementation

- void notifyTechnicalError(Throwable t) ;
- perform two key tasks:

- Consequence

Error Handling Patterns



Log Unexpected Errors

- If routine error conditions are logged, this makes real errors requiring operator intervention difficult to spot.

Solution

- Implement separate error handling mechanisms for expected and unexpected errors.
- Error conditions that are expected to arise in the course of normal domain processing should not be logged but handled in the code or by the user. Hence, any logged error should be viewed as requiring investigation.

Implementation

- use two distinct error handling approaches for expected and unexpected errors:
  - Log unexpected errors according to the other patterns
  - Do not log expected errors,

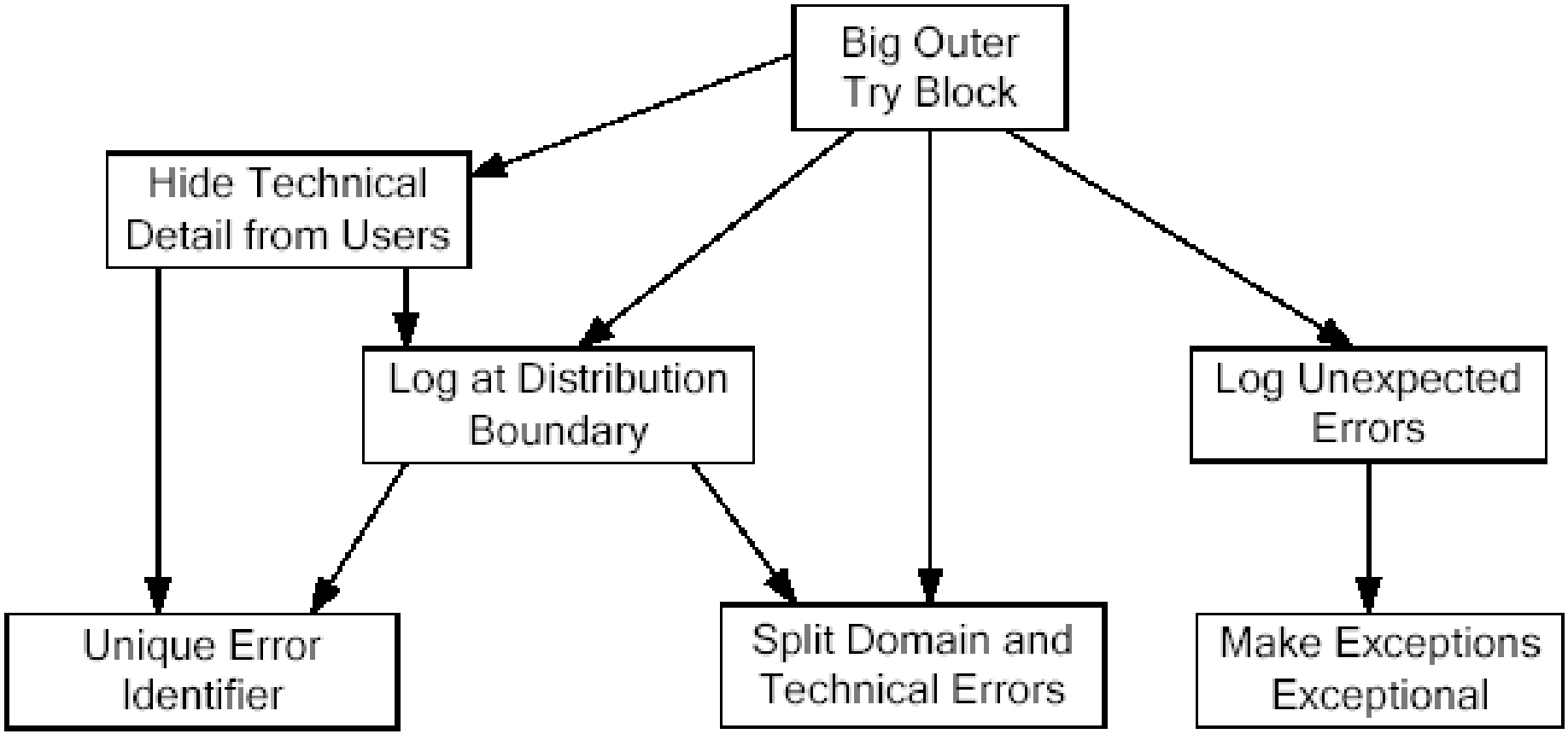
- could not connect to database
- no such product code

- One variation on this approach is
  - log different types of error message to different places.——application event log and security event log
- a user failing to authenticate
- large numbers of failed searches at a search engine site

- A second variation
  - log different types of error message in one location but to mark each log message with one or more attributes that allow a set of filters to be created to provide the ability to extract various subsets of the log content on demand to support different uses (such as error monitoring versus usability analysis).

- *Consequences*

Error Handling Patterns



Make Exceptions Exceptional

- A number of languages include exception handling facilities and these are powerful additions to the error handling toolkit available to programmers.
- However, if exceptions are used to indicate expected error conditions occurring, then calling code becomes much more difficult to understand.

Solution

- Indicate expected domain errors by means of return codes.
- Only use exceptions to indicate runtime problems such as underlying platform errors or configuration/data errors.

Implementation

- Conditions that will occur routinely in standard algorithms
  - be handled as part of the standard business logic in the system.
- Conditions that will only occur due to unexpected errors
  - be handled by a combination of logging and exiting the current code block via an exception path.