# Migration to Dyalog examples (still in progress. . . )

Dr. Markos Mitsos
markos.mitsos@ergo.de

Deutsche Krankenversicherung AG DKV - ERGO, Actuarial Department

APL Germany — Berlin

ERGO
A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- "theory" explained at previous conferences

ERGO

A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- "theory" explained at previous conferences

ERGO

A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- "theory" explained at previous conferences

ERGO

A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- “theory” explained at previous conferences

ERGO

A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- "theory" explained at previous conferences

Details:

- trying to create "true" Dyalog workspaces, not APL+Win copies
- in fact combined with overhauling of old code / structures
- some examples and comparisons

ERGO

A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- "theory" explained at previous conferences

Details:

- trying to create "true" Dyalog workspaces, not APL+Win copies
- in fact combined with overhauling of old code / structures
- some examples and comparisons

ERGO

A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- "theory" explained at previous conferences

Details:

- trying to create "true" Dyalog workspaces, not APL+Win copies
- in fact combined with overhauling of old code / structures
- some examples and comparisons

ERGO

A Munich Re company

# About

Report on migration progress:

- from APL+Win to Dyalog
- ongoing for years, moving really slow
- "theory" explained at previous conferences

Details:

- trying to create "true" Dyalog workspaces, not APL+Win copies
- in fact combined with overhauling of old code / structures
- some examples and comparisons

ERGO
A Munich Re company

# Outline

1 Migration of utilities

2 Migration of applications

ERGO

A Munich Re company

# Outline

1. Migration of utilities
2. Migration of applications

ERGO
A Munich Re company

# Outline of section on utilities

In this section we outline:

- Structure: structure and error handling
- Components: basic components

ERGO
A Munich Re company

# Outline of section on utilities

In this section we outline:

- Structure structure and error handling
- Components basic components

ERGO
A Munich Re company

# Outline of section on utilities

In this section we outline:

- Structure structure and error handling
- Components basic components

ERGO

A Munich Re company

# Workspace structure and versioning

Two worlds

- structure
  - APL+Win flat (RGL rather chaotic, only necessary part of DIV)
  - Dyalog clearly structured (RGL easy to understand, whole DIV)
- versioning
  - APL+Win manually (irregular WS copies, *_Ax in DIV, ALT_* in RGL)
  - Dyalog decoupled, full SVN repository

ERGO
A Munich Re company

# Workspace structure and versioning

Two worlds

- structure
  - APL+Win flat (RGL rather chaotic, only necessary part of DIV)
  - Dyalog clearly structured (RGL easy to understand, whole DIV)
- versioning
  - APL+Win manually (irregular WS copies, *_Ax in DIV, ALT_* in RGL)
  - Dyalog decoupled, full SVN repository

ERGO
A Munich Re company

# Workspace structure and versioning

Two worlds

- structure
  - APL+Win flat (RGL rather chaotic, only necessary part of DIV)
  - Dyalog clearly structured (RGL easy to understand, whole DIV)
- versioning
  - APL+Win manually (irregular WS copies, *_Ax in DIV, ALT_* in RGL)
  - Dyalog decoupled, full SVN repository

ERGO
A Munich Re company

# Workspace structure and versioning

Two worlds

- structure
  - APL+Win flat (RGL rather chaotic, only necessary part of DIV)
  - Dyalog clearly structured (RGL easy to understand, whole DIV)
- versioning
  - APL+Win manually (irregular WS copies, *_Ax in DIV, ALT_* in RGL)
  - Dyalog decoupled, full SVN repository

ERGO

A Munich Re company

# Workspace structure and versioning

Two worlds

- structure
  - APL+Win flat (RGL rather chaotic, only necessary part of DIV)
  - Dyalog clearly structured (RGL easy to understand, whole DIV)
- versioning
  - APL+Win manually (irregular WS copies, *_Ax in DIV, ALT_* in RGL)
  - Dyalog decoupled, full SVN repository

ERGO
A Munich Re company

# Workspace structure and versioning

Two worlds

- structure
  - APL+Win flat (RGL rather chaotic, only necessary part of DIV)
  - Dyalog clearly structured (RGL easy to understand, whole DIV)
- versioning
  - APL+Win manually (irregular WS copies, *_Ax in DIV, ALT_* in RGL)
  - Dyalog decoupled, full SVN repository

ERGO
A Munich Re company

# Workspace structure and versioning

Two worlds

- structure
  - APL+Win flat (RGL rather chaotic, only necessary part of DIV)
  - Dyalog clearly structured (RGL easy to understand, whole DIV)
- versioning
  - APL+Win manually (irregular WS copies, *_Ax in DIV, ALT_* in RGL)
  - Dyalog decoupled, full SVN repository

ERGO
A Munich Re company

# Local and global objects

Great differences

- localisation
  - APL+Win very long header (GUI+results in main object RGL)
  - Dyalog multi-line header (and reduction through namespaces)
- globals
  - APL+Win flat (pick appropriate names, problems with shadowing)
  - Dyalog dedicated namespaces, centrally "registered"

ERGO
A Munich Re company

# Local and global objects

Great differences

- localisation
  - APL+Win very long header (GUI+results in main object RGL)
  - Dyalog multi-line header (and reduction through namespaces)
- globals
  - APL+Win flat (pick appropriate names, problems with shadowing)
  - Dyalog dedicated namespaces, centrally "registered"

ERGO
A Munich Re company

# Local and global objects

Great differences

- localisation
  - APL+Win very long header (GUI+results in main object RGL)
  - Dyalog multi-line header (and reduction through namespaces)
- globals
  - APL+Win flat (pick appropriate names, problems with shadowing)
  - Dyalog dedicated namespaces, centrally "registered"

ERGO
A Munich Re company

# Local and global objects

Great differences

- localisation
  - APL+Win very long header (GUI+results in main object RGL)
  - Dyalog multi-line header (and reduction through namespaces)
- globals
  - APL+Win flat (pick appropriate names, problems with shadowing)
  - Dyalog dedicated namespaces, centrally "registered"

ERGO
A Munich Re company

# Local and global objects

Great differences

- localisation
  - APL+Win very long header (GUI+results in main object RGL)
  - Dyalog multi-line header (and reduction through namespaces)
- globals
  - APL+Win flat (pick appropriate names, problems with shadowing)
  - Dyalog dedicated namespaces, centrally "registered"

ERGO
A Munich Re company

# Local and global objects

Great differences

- localisation
  - APL+Win very long header (GUI+results in main object RGL)
  - Dyalog multi-line header (and reduction through namespaces)
- globals
  - APL+Win flat (pick appropriate names, problems with shadowing)
  - Dyalog dedicated namespaces, centrally "registered"

ERGO
A Munich Re company

# Local and global objects

Great differences

- localisation
  - APL+Win very long header (GUI+results in main object RGL)
  - Dyalog multi-line header (and reduction through namespaces)
- globals
  - APL+Win flat (pick appropriate names, problems with shadowing)
  - Dyalog dedicated namespaces, centrally "registered"

ERGO
A Munich Re company

# Error code versus signalling error

New philosophy

- principle
  - APL+Win (almost) always result type + result / error code
  - Dyalog always signalling error to calling environment
- usage
  - APL+Win without distinctions
  - Dyalog anticipation of some problems, distinction and control of error "level" (easier to call)

ERGO

A Munich Re company

# Error code versus signalling error

New philosophy

- principle
  - APL+Win (almost) always result type + result / error code
  - Dyalog always signalling error to calling environment
- usage
  - APL+Win without distinctions
  - Dyalog anticipation of some problems, distinction and control of error "level" (easier to call)

ERGO
A Munich Re company

# Error code versus signalling error

New philosophy

- principle
  - APL+Win (almost) always result type + result / error code
  - Dyalog always signalling error to calling environment
- usage
  - APL+Win without distinctions
  - Dyalog anticipation of some problems, distinction and control of error "level" (easier to call)

ERGO
A Munich Re company

# Error code versus signalling error

New philosophy

- principle
  - APL+Win (almost) always result type + result / error code
  - Dyalog always signalling error to calling environment
- usage
  - APL+Win without distinctions
  - Dyalog anticipation of some problems, distinction and control of error "level" (easier to call)

ERGO
A Munich Re company

# Error code versus signalling error

New philosophy

- principle
  - APL+Win (almost) always result type + result / error code
  - Dyalog always signalling error to calling environment
- usage
  - APL+Win without distinctions
  - Dyalog anticipation of some problems, distinction and control of error "level" (easier to call)

ERGO
A Munich Re company

# Error code versus signalling error

New philosophy

- principle
  - APL+Win (almost) always result type + result / error code
  - Dyalog always signalling error to calling environment
- usage
  - APL+Win without distinctions
  - Dyalog anticipation of some problems, distinction and control of error "level" (easier to call)

ERGO
A Munich Re company

# Error code versus signalling error

New philosophy

- principle
  - APL+Win (almost) always result type + result / error code
  - Dyalog always signalling error to calling environment
- usage
  - APL+Win without distinctions
  - Dyalog anticipation of some problems, distinction and control of error "level" (easier to call)

ERGO
A Munich Re company

# Dfns and small algorithms

Dfns and Dops (for small algorithms) as new tool

- APL+Win trim written out because too small for fn
- Dyalog Dfn OK as separate object
- collect some small algorithms as utilities
- also useful inline

ERGO
A Munich Re company

# Dfns and small algorithms

Dfns and Dops (for small algorithms) as new tool

- APL+Win trim written out because too small for fn
- Dyalog Dfn OK as separate object
- collect some small algorithms as utilities
- also useful inline

ERGO
A Munich Re company

# Dfns and small algorithms

Dfns and Dops (for small algorithms) as new tool

- APL+Win trim written out because too small for fn
- Dyalog Dfn OK as separate object
- collect some small algorithms as utilities
- also useful inline

ERGO
A Munich Re company

# Dfns and small algorithms

Dfns and Dops (for small algorithms) as new tool

- APL+Win trim written out because too small for fn
- Dyalog Dfn OK as separate object
- collect some small algorithms as utilities
- also useful inline

ERGO
A Munich Re company

# Dfns and small algorithms

Dfns and Dops (for small algorithms) as new tool

- APL+Win trim written out because too small for fn
- Dyalog Dfn OK as separate object
- collect some small algorithms as utilities
- also useful inline

ERGO
A Munich Re company

# Modified assignment and COM as namespace

Clarity and uniformity of code

- building of lists or statements better readable with modified assignment
- exposition of COM objects as namespaces allows usage via APL syntax

ERGO
A Munich Re company

# Modified assignment and COM as namespace

Clarity and uniformity of code

- building of lists or statements better readable with modified assignment
- exposition of COM objects as namespaces allows usage via APL syntax

ERGO
A Munich Re company

# Modified assignment and COM as namespace

Clarity and uniformity of code

- building of lists or statements better readable with modified assignment
- exposition of COM objects as namespaces allows usage via APL syntax

ERGO
A Munich Re company

# More/other (newer) system functions

Reduction of own functions, new functionality

- existence and erasure of files (fso objects versus NEXISTS and DELETE)
- reading / writing small files (NGET and NPUT compact)
- date arithmetic (own algorithms versus DT)
- usage of regular expressions (S and R)

ERGO
A Munich Re company

# More/other (newer) system functions

Reduction of own functions, new functionality

- existence and erasure of files (fso objects versus □NEXISTS and □DELETE)
- reading / writing small files (□NGET and □NPUT compact)
- date arithmetic (own algorithms versus □DT)
- usage of regular expressions (□S and □R)

ERGO
A Munich Re company

# More/other (newer) system functions

Reduction of own functions, new functionality

- existence and erasure of files (fso objects versus □NEXISTS and □DELETE)
- reading / writing small files (□NGET and □NPUT compact)
- date arithmetic (own algorithms versus □DT)
- usage of regular expressions (□S and □R)

ERGO
A Munich Re company

# More/other (newer) system functions

Reduction of own functions, new functionality

- existence and erasure of files (fso objects versus □NEXISTS and □DELETE)
- reading / writing small files (□NGET and □NPUT compact)
- date arithmetic (own algorithms versus □DT)
- usage of regular expressions (□S and □R)

ERGO
A Munich Re company

# More/other (newer) system functions

Reduction of own functions, new functionality

- existence and erasure of files (fso objects versus □NEXISTS and □DELETE)
- reading / writing small files (□NGET and □NPUT compact)
- date arithmetic (own algorithms versus □DT)
- usage of regular expressions (□S and □R)

ERGO
A Munich Re company

# More (newer) primitives

Reduction of own functions, new functionality, clarity of code

- At @ makes code clearer and avoids necessity to assign
- Key ⌸ for structure and grouped operations
- Power ⁑ for conditional application — but also a “real” case!

ERGO

A Munich Re company

# More (newer) primitives

Reduction of own functions, new functionality, clarity of code

- At @ makes code clearer and avoids necessity to assign
- Key ⌸ for structure and grouped operations
- Power ⁑ for conditional application — but also a “real” case!

ERGO
A Munich Re company

# More (newer) primitives

Reduction of own functions, new functionality, clarity of code

- At @ makes code clearer and avoids necessity to assign
- Key ⌸ for structure and grouped operations
- Power ⁑ for conditional application — but also a “real” case!

ERGO

A Munich Re company

# More (newer) primitives

Reduction of own functions, new functionality, clarity of code

- At @ makes code clearer and avoids necessity to assign
- Key ⌸ for structure and grouped operations
- Power ⁑ for conditional application — but also a “real” case!

ERGO

A Munich Re company

# Outline of section on applications

In this section we outline:

- Interactive interactive elements
- Enhancement enhancement through function separation
- Passing data passing data as parameters and globals

ERGO
A Munich Re company

# Outline of section on applications

In this section we outline:

- Interactive interactive elements
- Enhancement enhancement through function separation
- Passing data passing data as parameters and globals

ERGO
A Munich Re company

# Outline of section on applications

In this section we outline:

- **Interactive** interactive elements
- **Enhancement** enhancement through function separation
- **Passing data** passing data as parameters and globals

ERGO

A Munich Re company

# Outline of section on applications

In this section we outline:

- **Interactive** interactive elements
- **Enhancement** enhancement through function separation
- **Passing data** passing data as parameters and globals

ERGO
A Munich Re company

# Main and parameter GUI

Schematic interaction with user

- APL+Win main GUIs similar, Dyalog schematic
- APL+Win parameter GUIs based on pages, Dyalog on subforms
- multiple Grids allowed

ERGO
A Munich Re company

# Main and parameter GUI

Schematic interaction with user

- APL+Win main GUIs similar, Dyalog schematic
- APL+Win parameter GUIs based on pages, Dyalog on subforms
- multiple Grids allowed

ERGO
A Munich Re company

# Main and parameter GUI

Schematic interaction with user

- APL+Win main GUIs similar, Dyalog schematic
- APL+Win parameter GUIs based on pages, Dyalog on subforms
- multiple Grids allowed

ERGO
A Munich Re company

# Main and parameter GUI

Schematic interaction with user

- APL+Win main GUIs similar, Dyalog schematic
- APL+Win parameter GUIs based on pages, Dyalog on subforms
- multiple Grids allowed

ERGO
A Munich Re company

# Running protocol

## Visible protocol

- APL+Win Class as variant of Windows Form
- APL+Win Instance residing “somewhere”, passed as name of Windows object
- Dyalog Class proper class containing Form
- Dyalog Instance proper namespace, passed as reference, “saved” as global reference
- added methods for timestamping message and reacting to decision

ERGO
A Munich Re company

# Running protocol

Visible protocol

- APL+Win Class as variant of Windows Form
- APL+Win Instance residing “somewhere”, passed as name of Windows object
- Dyalog Class proper class containing Form
- Dyalog Instance proper namespace, passed as reference, “saved” as global reference
- added methods for timestamping message and reacting to decision

ERGO
A Munich Re company

# Running protocol

Visible protocol

- APL+Win Class as variant of Windows Form
- APL+Win Instance residing “somewhere”, passed as name of Windows object
- Dyalog Class proper class containing Form
- Dyalog Instance proper namespace, passed as reference, “saved” as global reference
- added methods for timestamping message and reacting to decision

ERGO
A Munich Re company

# Running protocol

Visible protocol

- APL+Win Class as variant of Windows Form
- APL+Win Instance residing “somewhere”, passed as name of Windows object
- Dyalog Class proper class containing Form
- Dyalog Instance proper namespace, passed as reference, “saved” as global reference
- added methods for timestamping message and reacting to decision

ERGO
A Munich Re company

# Running protocol

Visible protocol

- APL+Win Class as variant of Windows Form
- APL+Win Instance residing “somewhere”, passed as name of Windows object
- Dyalog Class proper class containing Form
- Dyalog Instance proper namespace, passed as reference, “saved” as global reference
- added methods for timestamping message and reacting to decision

ERGO
A Munich Re company

# Running protocol

Visible protocol

- APL+Win Class as variant of Windows Form
- APL+Win Instance residing “somewhere”, passed as name of Windows object
- Dyalog Class proper class containing Form
- Dyalog Instance proper namespace, passed as reference, “saved” as global reference
- added methods for timestamping message and reacting to decision

ERGO
A Munich Re company

# Usage of DB2 Utilities

separating Code for a Utility from that one

- APL+Win one function for unloading tables
- code partly redundant with function for cross-loading data
- code partly generic batch job / DB2 Utility code
- Dyalog generic part as utility (DIV), redundant part as separate function

ERGO
A Munich Re company

# Usage of DB2 Utilities

separating Code for a Utility from that one

- APL+Win one function for unloading tables
- code partly redundant with function for cross-loading data
- code partly generic batch job / DB2 Utility code
- Dyalog generic part as utility (DIV), redundant part as separate function

ERGO
A Munich Re company

# Usage of DB2 Utilities

separating Code for a Utility from that one

- APL+Win one function for unloading tables
- code partly redundant with function for cross-loading data
- code partly generic batch job / DB2 Utility code
- Dyalog generic part as utility (DIV), redundant part as separate function

ERGO
A Munich Re company

# Usage of DB2 Utilities

separating Code for a Utility from that one

- APL+Win one function for unloading tables
- code partly redundant with function for cross-loading data
- code partly generic batch job / DB2 Utility code
- Dyalog generic part as utility (DIV), redundant part as separate function

ERGO
A Munich Re company

# Usage of DB2 Utilities

separating Code for a Utility from that one

- APL+Win one function for unloading tables
- code partly redundant with function for cross-loading data
- code partly generic batch job / DB2 Utility code
- Dyalog generic part as utility (DIV), redundant part as separate function

ERGO
A Munich Re company

# “Small” namespaces

Usage of “small” namespaces to group arguments / parameters

- APL+Win high number of needed parameters lead to different and/or confusing calls
- Dyalog aggregation of parameters with similar function facilitated uniform, better readable call
- **regst** for “regular steering”, **sondst** for specials like debugging, **hinw** for logging of problems
- for clarity pruned copies of such namespaces in “leafs”
- chaos in higher-up functions avoided

ERGO

A Munich Re company

# “Small” namespaces

Usage of “small” namespaces to group arguments / parameters

- APL+Win high number of needed parameters lead to different and/or confusing calls
- Dyalog aggregation of parameters with similar function facilitated uniform, better readable call
- **regst** for “regular steering”, **sondst** for specials like debugging, **hinw** for logging of problems
- for clarity pruned copies of such namespaces in “leafs”
- chaos in higher-up functions avoided

ERGO
A Munich Re company

# “Small” namespaces

Usage of “small” namespaces to group arguments / parameters

- APL+Win high number of needed parameters lead to different and/or confusing calls
- Dyalog aggregation of parameters with similar function facilitated uniform, better readable call
- **regst** for “regular steering”, **sondst** for specials like debugging, **hinw** for logging of problems
- for clarity pruned copies of such namespaces in “leafs”
- chaos in higher-up functions avoided

ERGO

A Munich Re company

# “Small” namespaces

Usage of “small” namespaces to group arguments / parameters

- APL+Win high number of needed parameters lead to different and/or confusing calls
- Dyalog aggregation of parameters with similar function facilitated uniform, better readable call
- **regst** for “regular steering”, **sondst** for specials like debugging, **hinw** for logging of problems
- for clarity pruned copies of such namespaces in “leafs”
- chaos in higher-up functions avoided

ERGO

A Munich Re company

# “Small” namespaces

Usage of “small” namespaces to group arguments / parameters

- APL+Win high number of needed parameters lead to different and/or confusing calls
- Dyalog aggregation of parameters with similar function facilitated uniform, better readable call
- **regst** for “regular steering”, **sondst** for specials like debugging, **hinw** for logging of problems
- for clarity pruned copies of such namespaces in “leafs”
- chaos in higher-up functions avoided

ERGO

A Munich Re company

# “Small” namespaces

Usage of “small” namespaces to group arguments / parameters

- APL+Win high number of needed parameters lead to different and/or confusing calls
- Dyalog aggregation of parameters with similar function facilitated uniform, better readable call
- **regst** for “regular steering”, **sondst** for specials like debugging, **hinw** for logging of problems
- for clarity pruned copies of such namespaces in “leafs”
- chaos in higher-up functions avoided

ERGO

A Munich Re company

# Local, semi-global and global

Very high number of variables with significant amount of data

- APL+Win actuarial data could practically not passed on as parameters
- ugly, semi-globals solution used, localisation in main function, "transparency" in dependent ones
- additionally data "saved" as true globals at end of function
- Dyalog actuarial data is namespace, passed as reference
- dependent functions make copy, modify and return it, calling one uses modification
- global copy if requested

ERGO
A Munich Re company

# Local, semi-global and global

Very high number of variables with significant amount of data

- APL+Win actuarial data could practically not passed on as parameters
- ugly, semi-globals solution used, localisation in main function, "transparency" in dependent ones
- additionally data "saved" as true globals at end of function
- Dyalog actuarial data is namespace, passed as reference
- dependent functions make copy, modify and return it, calling one uses modification
- global copy if requested

ERGO
A Munich Re company

# Local, semi-global and global

Very high number of variables with significant amount of data

- APL+Win actuarial data could practically not passed on as parameters
- ugly, semi-globals solution used, localisation in main function, "transparency" in dependent ones
- additionally data "saved" as true globals at end of function
- Dyalog actuarial data is namespace, passed as reference
- dependent functions make copy, modify and return it, calling one uses modification
- global copy if requested

ERGO

A Munich Re company

# Local, semi-global and global

Very high number of variables with significant amount of data

- APL+Win actuarial data could practically not passed on as parameters
- ugly, semi-globals solution used, localisation in main function, “transparency” in dependent ones
- additionally data “saved” as true globals at end of function
- Dyalog actuarial data is namespace, passed as reference
- dependent functions make copy, modify and return it, calling one uses modification
- global copy if requested

ERGO

A Munich Re company

# Local, semi-global and global

Very high number of variables with significant amount of data

- APL+Win actuarial data could practically not passed on as parameters
- ugly, semi-globals solution used, localisation in main function, "transparency" in dependent ones
- additionally data "saved" as true globals at end of function
- Dyalog actuarial data is namespace, passed as reference
- dependent functions make copy, modify and return it, calling one uses modification
- global copy if requested

ERGO

A Munich Re company

# Local, semi-global and global

Very high number of variables with significant amount of data

- APL+Win actuarial data could practically not passed on as parameters
- ugly, semi-globals solution used, localisation in main function, "transparency" in dependent ones
- additionally data "saved" as true globals at end of function
- Dyalog actuarial data is namespace, passed as reference
- dependent functions make copy, modify and return it, calling one uses modification
- global copy if requested

ERGO

A Munich Re company

# Local, semi-global and global

Very high number of variables with significant amount of data

- APL+Win actuarial data could practically not passed on as parameters
- ugly, semi-globals solution used, localisation in main function, "transparency" in dependent ones
- additionally data "saved" as true globals at end of function
- Dyalog actuarial data is namespace, passed as reference
- dependent functions make copy, modify and return it, calling one uses modification
- global copy if requested

ERGO

A Munich Re company

# Conclusion

Future:

- infrastructure almost done
- simulations proper on the horizon
- still long-time project. . .

ERGO

A Munich Re company

# Conclusion

Future:

- infrastructure almost done
- simulations proper on the horizon
- still long-time project. . .

ERGO

A Munich Re company

# Conclusion

Future:

- infrastructure almost done
- simulations proper on the horizon
- still long-time project. . .

ERGO

A Munich Re company

# Conclusion

Future:

- infrastructure almost done
- simulations proper on the horizon
- still long-time project...

ERGO

A Munich Re company

# Conclusion

Future:

- infrastructure almost done
- simulations proper on the horizon
- still long-time project...

◂ begin

ERGO

A Munich Re company