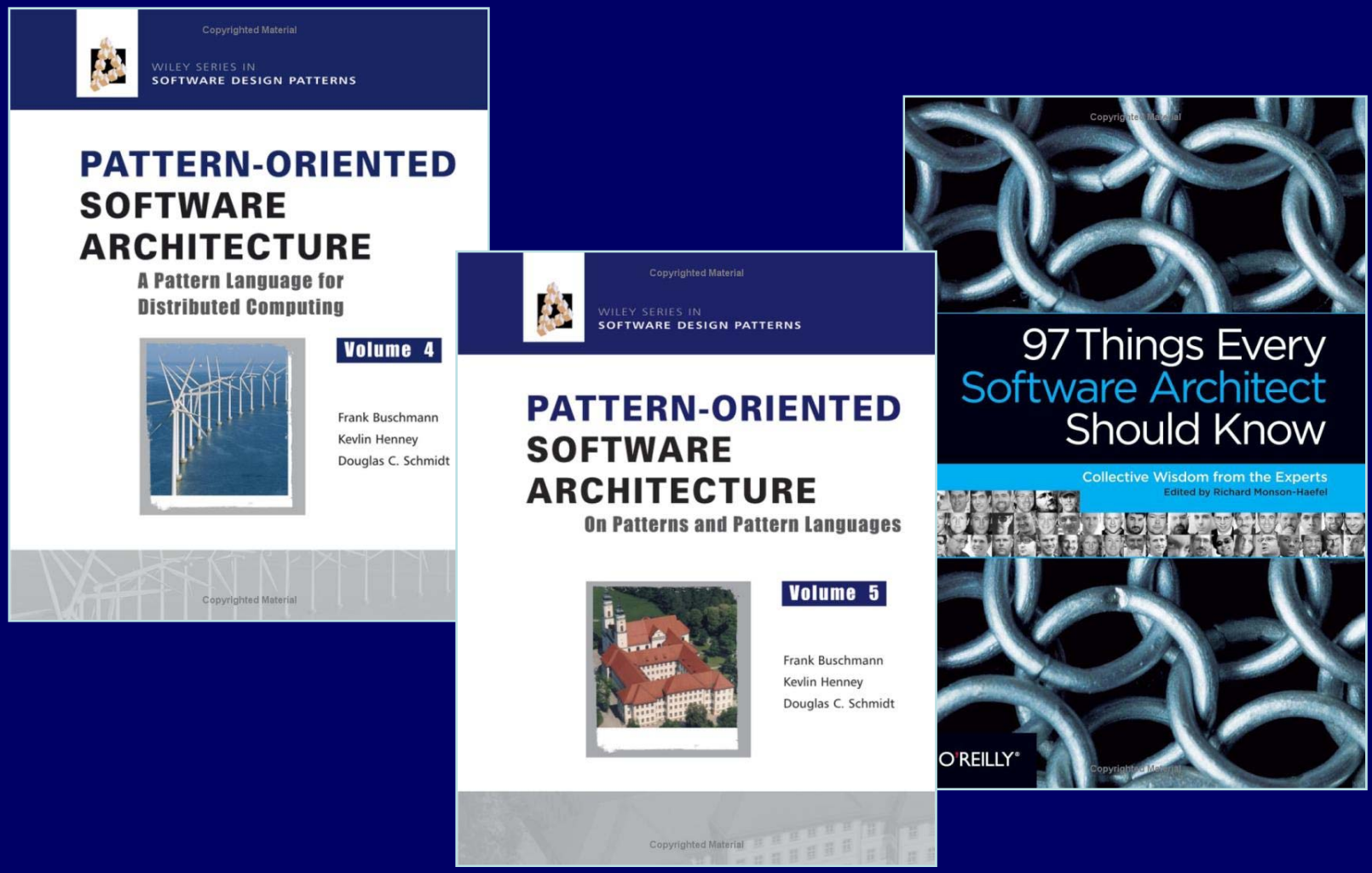
# Five Considerations for Software Architects

**Kevlin Henney**

***kevlin@curbralan.com***


Copyrighted Material
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
Copyrighted Material
Copyrighted Material
WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
Copyrighted Material
97 Things Every Software Architect Should Know
Collective Wisdom from the Experts
Edited by Richard Monson-Haefel
O'REILLY®


Economy
Visibility
Spacing
Symmetry
Emergence
QCONNECT

# Economy


There's nothing long-winded about
"Liberté, égalité, fraternité".
Maurice Saatchi

*Continuing existence or cessation of existence: those are the scenarios. Is it more empowering mentally to work towards an accommodation of the downsizings and negative outcomes of adversarial circumstance, or would it be a greater enhancement of the bottom line to move forwards to a challenge to our current difficulties, and, by making a commitment to opposition, to effect their demise?*

**Tom Burton, *Long Words Bother Me***

*To be, or not to be: that is the question:*
*Whether 'tis nobler in the mind to suffer*
*The slings and arrows of outrageous fortune,*
*Or to take arms against a sea of troubles,*
*And by opposing end them?*

**William Shakespeare, *Hamlet***

```
#ifndef BOOST_LEXICAL_CAST_INCLUDED
#define BOOST_LEXICAL_CAST_INCLUDED

// Boost lexical_cast.hpp header  ------------------------------------------------//
//
// See http://www.boost.org for most recent version including documentation.
//
// what:  lexical_cast custom keyword cast
// who:   contributed by Kevlin Henney and enhanced by Terje Slettebø
// when:  November 2000, March 2003
//
#include <typeinfo>
#include <string>
#include <boost/config.hpp>
#include <boost/limits.hpp>
#include <boost/static_assert.hpp>

#ifdef BOOST_NO_STRINGSTREAM
#include <strstream>
#else
#include <sstream>
#endif

#if defined(BOOST_NO_TEMPLATE_PARTIAL_SPECIALIZATION) || \
    defined(BOOST_BCB_PARTIAL_SPECIALIZATION_BUG)
#include <boost/mpl/if.hpp>
#include <boost/type_traits/same_traits.hpp>
#endif

namespace boost
{
    // exception used to indicate runtime lexical_cast failure
    class bad_lexical_cast : public std::bad_cast
    {
    public:
        virtual ~bad_lexical_cast() throw()
        {
        }
    };

    namespace detail
    {
        template<typename Target, typename Source>
        class no_lexical_conversion : public bad_lexical_cast
        {
        public:
            no_lexical_conversion()
              : _description(
                    std::string() + "bad lexical cast: " +
                    "source type value could not be interpreted as target, Target=" +
                    typeid(Target).name() + ", Source=" + typeid(Source).name())
            {
            }
            virtual ~no_lexical_conversion() throw()
            {
            }
            virtual const char *what() const throw()
            {
                return _description.c_str();
            }
        private:
            const std::string _description; // static initialization fails on MSVC6
        };
    }

namespace detail
{
    template<typename CharType>
    struct stringstream
    {
        #if defined(BOOST_NO_STRINGSTREAM)
        typedef std::strstream type;
        #else
        typedef std::basic_stringstream<CharType> type;
        #endif
    };

    template<typename Target, typename Source, typename Interpreter>
    void setup_interpreter(Interpreter &interpreter)
    {
        interpreter.unsetf(std::ios_base::skipws);

        if(std::numeric_limits<Target>::is_specialized)
            interpreter.precision(std::numeric_limits<Target>::digits10 + 1);
        else if(std::numeric_limits<Source>::is_specialized)
            interpreter.precision(std::numeric_limits<Source>::digits10 + 1);
    }

template<typename Target, typename Source, typename CharType>
struct lexical_cast_base
{
    static Target do_cast(Source arg)
    {
        typename stringstream<CharType>::type interpreter;

        setup_interpreter<Target, Source>(interpreter);

        Target result;

        if(!(interpreter << arg) || !(interpreter >> result) ||
           !(interpreter >> std::ws).eof())
            throw detail::no_lexical_conversion<Target, Source>();

        return result;
    }
};

////////////////////////////////////////////////////////////////////////////////
// any_to_any_base
////////////////////////////////////////////////////////////////////////////////

template<typename Target, typename Source, typename CharType>
struct any_to_any_base
{
    static Target do_cast(Source arg)
    {
        #ifndef BOOST_NO_STRINGSTREAM
        typename stringstream<CharType>::type interpreter;

        if(!(interpreter << arg))
            throw detail::no_lexical_conversion<Target, Source>();
#else
```

```
    typename stringstream<CharType>::type interpreter(arg);
#endif

        setup_interpreter<Target, Source>(interpreter);

        Target result;

        if(!(interpreter >> result) || !(interpreter >> std::ws).eof())
            throw detail::no_lexical_conversion<Target, Source>();

        return result;
    }
};

////////////////////////////////////////////////////////////////////////////////
// any_to_string_base
////////////////////////////////////////////////////////////////////////////////

template<typename Target, typename Source, typename CharType>
struct any_to_string_base
{
    static Target do_cast(Source arg)
    {
        typename stringstream<CharType>::type interpreter;
        setup_interpreter<Target, Source>(interpreter);
        if(!(interpreter << arg))
            throw detail::no_lexical_conversion<Target, Source>();
        return interpreter.str();
    }
};

template<typename Target, typename Source>
struct string_to_char_base
{
    static Target do_cast(Source arg)
    {
        if(arg.length()!=1)
            throw detail::no_lexical_conversion<Target, Source>();
        return arg[0];
    }
};

template<typename Target, typename Source>
struct pointer_to_char_to_char_base
{
    static Target do_cast(Source arg)
    {
        if(arg[0] == Target() || arg[1] != Target())
            throw detail::no_lexical_conversion<Target, Source>();
        return arg[0];
    }
};

template<typename Target, typename Source>
struct char_to_string_base
{
    static Target do_cast(Source arg)
    {
        return Target(1, arg);
    }
};

template<typename Target, typename Source>
struct direct_cast_base
{
    static Target do_cast(Source arg)
    {
        return arg;
    }
};

#if !defined(BOOST_NO_TEMPLATE_PARTIAL_SPECIALIZATION) && \
!defined(BOOST_BCB_PARTIAL_SPECIALIZATION_BUG)

template<typename Target, typename Source>
inline Target lexical_cast_impl(Target *, Source arg)
{
return lexical_cast_base<Target, Source, char>::do_cast(arg);
}

template<typename Target, typename Source>
void lexical_cast_impl(Target **,Source)
{
    BOOST_STATIC_ASSERT((Target *) 0); // Pointer as target type not supported
}

inline char lexical_cast_impl(char *, char *arg)
{
    return pointer_to_char_to_char_base<char, char *>::do_cast(arg);
}

inline char lexical_cast_impl(char *, const char *arg)
{
    return pointer_to_char_to_char_base<char, const char *>::do_cast(arg);
}

template<typename Source>
inline wchar_t lexical_cast_impl(wchar_t *, Source arg)
{
    return lexical_cast_base<wchar_t, Source, wchar_t>::do_cast(arg);
}

inline wchar_t lexical_cast_impl(wchar_t *, wchar_t *arg)
{
    return pointer_to_char_to_char_base<wchar_t, wchar_t *>::do_cast(arg);
}

inline wchar_t lexical_cast_impl(wchar_t *, const wchar_t *arg)
{
    return pointer_to_char_to_char_base<wchar_t, const wchar_t *>::do_cast(arg);
}

inline wchar_t lexical_cast_impl(wchar_t *, const std::wstring &arg)
{
return string_to_char_base<wchar_t, const std::wstring &>::do_cast(arg);
}

template<typename CharType, typename CharTraits, typename Allocator>
inline CharType
    lexical_cast_impl(CharType *, const std::basic_string<CharType, CharTraits, Allocator> &arg)
{
    return string_to_char_base<CharType, const std::basic_string<CharType, CharTraits, Allocator>
```

```
&>::do_cast(arg);
}

template<typename CharType, typename CharTraits, typename Allocator, typename Source>
inline std::basic_string<CharType, CharTraits, Allocator>
    lexical_cast_impl(std::basic_string<CharType, CharTraits, Allocator> *, Source arg)
{
    return any_to_string_base<std::basic_string<CharType, CharTraits, Allocator>, Source,
CharType>::do_cast(arg);
}

template<typename CharType, typename CharTraits, typename Allocator>
inline std::basic_string<CharType, CharTraits, Allocator>
    lexical_cast_impl(std::basic_string<CharType, CharTraits, Allocator> *, CharType arg)
{
    return char_to_string_base<std::basic_string<CharType, CharTraits, Allocator>, CharType>::do_cast(arg);
}

template<typename CharType, typename CharTraits, typename Allocator>
inline std::basic_string<CharType, CharTraits, Allocator>
    lexical_cast_impl(std::basic_string<CharType, CharTraits, Allocator> *, CharType *arg)
{
    return direct_cast_base<std::basic_string<CharType, CharTraits, Allocator>, CharType *>::do_cast(arg);
}

template<typename CharType, typename CharTraits, typename Allocator>
inline std::basic_string<CharType, CharTraits, Allocator>
    lexical_cast_impl(std::basic_string<CharType, CharTraits, Allocator> *, const CharType *arg)
{
    return direct_cast_base<std::basic_string<CharType, CharTraits, Allocator>, const CharType
*>::do_cast(arg);
}

template<typename Target>
inline Target lexical_cast_impl(Target *, char *arg)
{
    return string_to_any_base<Target, char *, char>::do_cast(arg);
}

template<typename Target>
inline Target lexical_cast_impl(Target *, const char *arg)
{
    return string_to_any_base<Target, const char *, char>::do_cast(arg);
}

template<typename Target>
inline Target lexical_cast_impl(Target *, wchar_t *arg)
{
    return lexical_cast_base<Target, wchar_t, wchar_t>::do_cast(arg);
}

template<typename Target>
inline Target lexical_cast_impl(Target *, const wchar_t *arg)
{
    return string_to_any_base<Target, const wchar_t *, wchar_t>::do_cast(arg);
}

template<typename Target, typename CharType, typename CharTraits, typename Allocator>
inline Target lexical_cast_impl(Target *, const std::basic_string<CharType, CharTraits, Allocator> &arg)
{
    return string_to_any_base<Target,
        const std::basic_string<CharType, CharTraits, Allocator> &, CharType>::do_cast(arg);
}

template<typename Type>
inline Type lexical_cast_impl(Type *, const Type &arg)
{
    return direct_cast_base<Type, const Type &>::do_cast(arg);
}

inline std::string lexical_cast_impl(std::string *, char *arg)
{
    return direct_cast_base<std::string, char *>::do_cast(arg);
}

inline std::string lexical_cast_impl(std::string *, const char *arg)
{
    return direct_cast_base<std::string, const char *>::do_cast(arg);
}

inline std::string lexical_cast_impl(std::string *, const std::string &arg)
{
    return direct_cast_base<std::string, const std::string &>::do_cast(arg);
}

inline std::wstring lexical_cast_impl(std::wstring *, wchar_t *arg)
{
    return direct_cast_base<std::wstring, wchar_t *>::do_cast(arg);
}

inline std::wstring lexical_cast_impl(std::wstring *, const wchar_t *arg)
{
    return direct_cast_base<std::wstring, const wchar_t *>::do_cast(arg);
}

inline std::wstring lexical_cast_impl(std::wstring *, const std::wstring &arg)
{
    return direct_cast_base<std::wstring, const std::wstring &>::do_cast(arg);
}

inline wchar_t lexical_cast_impl(wchar_t *, wchar_t arg)
{
    return direct_cast_base<wchar_t, wchar_t>::do_cast(arg);
}

inline std::wstring lexical_cast_impl(std::wstring *, wchar_t arg)
{
    return char_to_string_base<wchar_t, wchar_t>::do_cast(arg);
}

#else

// Simulated partial specialisation
// select_base
////////////////////////////////////////////////////////////////////////////////
```

```
    // mpl::apply_if doesn't work well for MSVC 6 here, and neither does
    // inheriting from a metafunction
        template<class Target, class Source>
        struct select_base
        {
          typedef typename mpl::if_<is_same<Target, char>,                  // Target==char?
            typename mpl::if_<is_same<Source, char>,                        // Source==char?
              direct_cast_base<Target, Source>,
              typename mpl::if_<is_same<Source, char *>,                    // Source==char *?
                direct_cast_base<Target, Source>,
                typename mpl::if_<is_same<Source, const char *>,            // Source==const char *?
                  pointer_to_char_to_char_base<Target, Source>,
                  typename mpl::if_<is_same<Source, std::string>,           // Source==std::string?
                    string_to_char_base<Target, const Source &>,
                    lexical_cast_base<Target, Source, char>
                  >::type
                >::type
              >::type
            >::type,
            typename mpl::if_<is_same<Target, std::string>,                 // Target==std::string?
              typename mpl::if_<is_same<Source, char>,                      // Source==char?
                char_to_string_base<Target, Source>,
                typename mpl::if_<is_same<Source, char *>,                  // Source==char *?
                  direct_cast_base<Target, Source>,
                  typename mpl::if_<is_same<Source, const char *>,          // Source==const char *?
                    direct_cast_base<Target, Source>,
                    typename mpl::if_<is_same<Source, std::string>,         // Source==std::string?
                      direct_cast_base<Target, const Source &>,
                      any_to_string_base<Target, Source, char>
                    >::type
                  >::type
                >::type
              >::type,
              typename mpl::if_<is_same<Target, wchar_t>,                   // Target==wchar_t?
                typename mpl::if_<is_same<Source, wchar_t>,                 // Source==wchar_t?
                  direct_cast_base<Target, Source>,
                  typename mpl::if_<is_same<Source, wchar_t *>,             // Source==wchar_t *?
                    pointer_to_char_to_char_base<Target, Source>,
                    typename mpl::if_<is_same<Source, const wchar_t *>,     // Source==const wchar_t *?
                      direct_cast_base<Target, Source>,
                      typename mpl::if_<is_same<Source, std::wstring>,      // Source==std::wstring?
                        string_to_char_base<Target, const Source &>,
                        lexical_cast_base<Target, Source, wchar_t>
                      >::type
                    >::type
                  >::type
                >::type,
                typename mpl::if_<is_same<Target, std::wstring>,            // Target==std::wstring?
                  typename mpl::if_<is_same<Source, wchar_t>,               // Source==wchar_t?
                    char_to_string_base<Target, Source>,
                    typename mpl::if_<is_same<Source, wchar_t *>,           // Source==wchar_t *?
                      direct_cast_base<Target, Source>,
                      typename mpl::if_<is_same<Source, const wchar_t *>,   // Source==const wchar_t *?
                        direct_cast_base<Target, Source>,
                        typename mpl::if_<is_same<Source, std::wstring>,    // Source==std::wstring?
                          direct_cast_base<Target, const Source &>,
                          any_to_string_base<Target, Source, wchar_t>
                        >::type
                      >::type
                    >::type
                  >::type,
                  typename mpl::if_<is_same<Target, Source>,                                    // Target==Source?
                    direct_cast_base<Target, const Source &>,
                    typename mpl::if_<is_same<Source, char *>,                                  // Source==char *?
                      string_to_any_base<Target, Source, char>,
                      typename mpl::if_<is_same<Source, const char *>,                          //
Source==const
char *?
                        string_to_any_base<Target, Source, char>,
                        typename mpl::if_<is_same<Source, std::string>,                         //
Source==std::string?
                          string_to_any_base<Target, const Source &, char>,
                          typename mpl::if_<is_same<Source, wchar_t>,                           // Source==wchar_t?
                            lexical_cast_base<Target, Source, wchar_t>,
                            typename mpl::if_<is_same<Source, const wchar_t *>,                 // Source==const
wchar_t *?
                              lexical_cast_base<Target, Source, wchar_t>,
                              typename mpl::if_<is_same<Source, std::wstring>,                  //
Source==std::wstring?
                                lexical_cast_base<Target, const Source &, wchar_t>,
                                lexical_cast_base<Target, Source, char>
                              >::type
                            >::type
                          >::type
                        >::type
                      >::type
                    >::type
                  >::type
                >::type
              >::type
            >::type
          >::type type;
        };

        template<typename Target, typename Source>
        inline Target lexical_cast_impl(Target *, Source arg)
        {
            return select_base<Target, Source>::type::do_cast(arg);
        }
    #endif
}

template<typename Target, typename Source>
inline Target lexical_cast(const Source &arg)
{
    return detail::lexical_cast_impl(static_cast<Target *>(0), arg);
}
}

// Copyright Kevlin Henney, 2000-2003, Terje Slettebø, 2003. All rights reserved.
//
// Permission to use, copy, modify, and distribute this software for any
// purpose is hereby granted without fee, provided that this copyright and
// permissions notice appear in all copies and derivatives.
//
// This software is provided "as is" without express or implied warranty.
#endif
```

```
#ifndef BOOST_LEXICAL_CAST_INCLUDED
#define BOOST_LEXICAL_CAST_INCLUDED

// Boost lexical_cast.hpp header  ------------------------------------------//
//
// See http://www.boost.org for most recent version including documentation.
// See end of this header for rights and permissions.
//
// what:  lexical_cast custom keyword cast
// who:   contributed by Kevlin Henney,
//        enhanced with contributions from Terje Slettebø,
//        with additional fixes and suggestions from Gennaro Prota,
//        Beman Dawes, Dave Abrahams, Daryle Walker, Peter Dimov,
//        and other Boosters
// when:  November 2000, March 2003

#include <string>
#include <typeinfo>
#include <boost/config.hpp>
#include <boost/limits.hpp>
#include <boost/type_traits/is_pointer.hpp>

#ifdef BOOST_NO_STRINGSTREAM
#include <strstream>
#else
#include <sstream>
#endif

#if defined(BOOST_NO_STRINGSTREAM) || \
    defined(BOOST_NO_STD_WSTRING) || \
    defined(BOOST_NO_CWCHAR) || \
    defined(BOOST_NO_CWCTYPE)
#define DISABLE_WIDE_CHAR_SUPPORT
#endif

#ifdef BOOST_NO_INTRINSIC_WCHAR_T
#include <cwchar>
#endif

namespace boost
{
    // exception used to indicate runtime lexical_cast failure
    class bad_lexical_cast : public std::bad_cast
    {
    public:
        virtual ~bad_lexical_cast() throw()
        {
        }
    };

    namespace detail // actual underlying concrete exception type
    {
        template<typename Target, typename Source>
        class no_lexical_conversion : public bad_lexical_cast
        {
        public:
            no_lexical_conversion()
              : description(
                    std::string() + "bad lexical cast: " +
                    "source type value could not be interpreted as target, Target=" +
                    typeid(Target).name() + ", Source=" + typeid(Source).name())
            {
            }
            virtual ~no_lexical_conversion() throw()
            {
            }
            virtual const char *what() const throw()
            {
                return description.c_str();
            }
        private:
            const std::string description; // static initialization fails on MSVC6
        };
    }

    namespace detail // selectors for choosing stream character type
    {
        template<typename Type>
        struct stream_char
        {
            typedef char type;
        };

        #ifndef DISABLE_WIDE_CHAR_SUPPORT
        template<>
        struct stream_char<wchar_t>
        {
            typedef wchar_t type;
        };

        template<>
        struct stream_char<wchar_t *>
        {
            typedef wchar_t type;
        };

        template<>
        struct stream_char<const wchar_t *>
        {
            typedef wchar_t type;
        };

    template<>
    struct stream_char<std::wstring>
        {
            typedef wchar_t type;
        };
        #endif

        template<typename TargetChar, typename SourceChar>
        struct widest_char
        {
            typedef TargetChar type;
        };

        template<>
        struct widest_char<char, wchar_t>
        {
            typedef wchar_t type;
        };
    }

    namespace detail // stream wrapper for handling lexical conversions
    {
        template<typename Target, typename Source>
        class lexical_stream
        {
        public:
            lexical_stream()
            {
                stream.unsetf(std::ios::skipws);

                if(std::numeric_limits<Target>::is_specialized)
                    stream.precision(std::numeric_limits<Target>::digits10 + 1);
                else if(std::numeric_limits<Source>::is_specialized)
                    stream.precision(std::numeric_limits<Source>::digits10 + 1);
            }
            ~lexical_stream()
            {
                #if defined(BOOST_NO_STRINGSTREAM)
                stream.freeze(false);
                #endif
            }
            bool operator<<(const Source &input)
            {
                return stream << input;
            }
            template<typename InputStreamable>
            bool operator>>(InputStreamable &output)
            {
                return !is_pointer<InputStreamable>::value &&
                       stream >> output &&
                       (stream >> std::ws).eof();
            }
            bool operator>>(std::string &output)
            {
                #if defined(BOOST_NO_STRINGSTREAM)
                stream << '\0';
                #endif
                output = stream.str();
                return true;
            }
            #ifndef DISABLE_WIDE_CHAR_SUPPORT
            bool operator>>(std::wstring &output)
            {
                output = stream.str();
                return true;
            }
            #endif
        private:
            typedef typename widest_char<
                    typename stream_char<Target>::type,
                    typename stream_char<Source>::type>::type char_type;

            #if defined(BOOST_NO_STRINGSTREAM)
            std::strstream stream;
            #elif defined(BOOST_NO_STD_LOCALE)
            std::stringstream stream;
            #else
            std::basic_stringstream<char_type> stream;
            #endif
        };
    }

    template<typename Target, typename Source>
    Target lexical_cast(Source arg)
    {
        detail::lexical_stream<Target, Source> interpreter;
        Target result;

        if(!(interpreter << arg && interpreter >> result))
            throw detail::no_lexical_conversion<Target, Source>();
        return result;
    }
}

// Copyright Kevlin Henney, 2000-2003. All rights reserved.
//
// Permission to use, copy, modify, and distribute this software for any
// purpose is hereby granted without fee, provided that this copyright and
// permissions notice appear in all copies and derivatives.
//
// This software is provided "as is" without express or implied warranty.

#undef DISABLE_WIDE_CHAR_SUPPORT
#endif
```

*The difference between a good and a poor architect is that the poor architect succumbs to every temptation and the good one resists it.*

**Ludwig Wittgenstein**

```
interface Iterator
{
    boolean set_to_first_element();
    boolean set_to_next_element();
    boolean set_to_next_nth_element(in unsigned long n) raises(…);
    boolean retrieve_element(out any element) raises(…);
    boolean retrieve_element_set_to_next(out any element, out boolean more) raises(…);
    boolean retrieve_next_n_elements(in unsigned long n, out AnySequence result, out boolean more) raises(…);
    boolean not_equal_retrieve_element_set_to_next(in Iterator test, out any element) raises(…);
    void remove_element() raises(…);
    boolean remove_element_set_to_next() raises(…);
    boolean remove_next_n_elements(in unsigned long n, out unsigned long actual_number) raises(…);
    boolean not_equal_remove_element_set_to_next(in Iterator test) raises(…);
    void replace_element(in any element) raises(…);
    boolean replace_element_set_to_next(in any element) raises(…);
    boolean replace_next_n_elements(in AnySequence elements, out unsigned long actual_number) raises(…);
    boolean not_equal_replace_element_set_to_next(in Iterator test, in any element) raises(…);
    boolean add_element_set_iterator(in any element) raises(…);
    boolean add_n_elements_set_iterator(in AnySequence elements, out unsigned long actual_number) raises(…);
    void invalidate();
    boolean is_valid();
    boolean is_in_between();
    boolean is_for(in Collection collector);
    boolean is_const();
    boolean is_equal(in Iterator test) raises(…);
    Iterator clone();
    void assign(in Iterator from_where) raises(…);
    void destroy();
};
```

```
interface BindingIterator
{
    boolean next_one(out Binding result);
    boolean next_n(in unsigned long how_many, out BindingList result);
    void destroy();
};
```

*That which is overdesigned, too highly specific, anticipates outcome; the anticipation of outcome guarantees, if not failure, the absence of grace.*

William Gibson, *All Tomorrow's Parties*

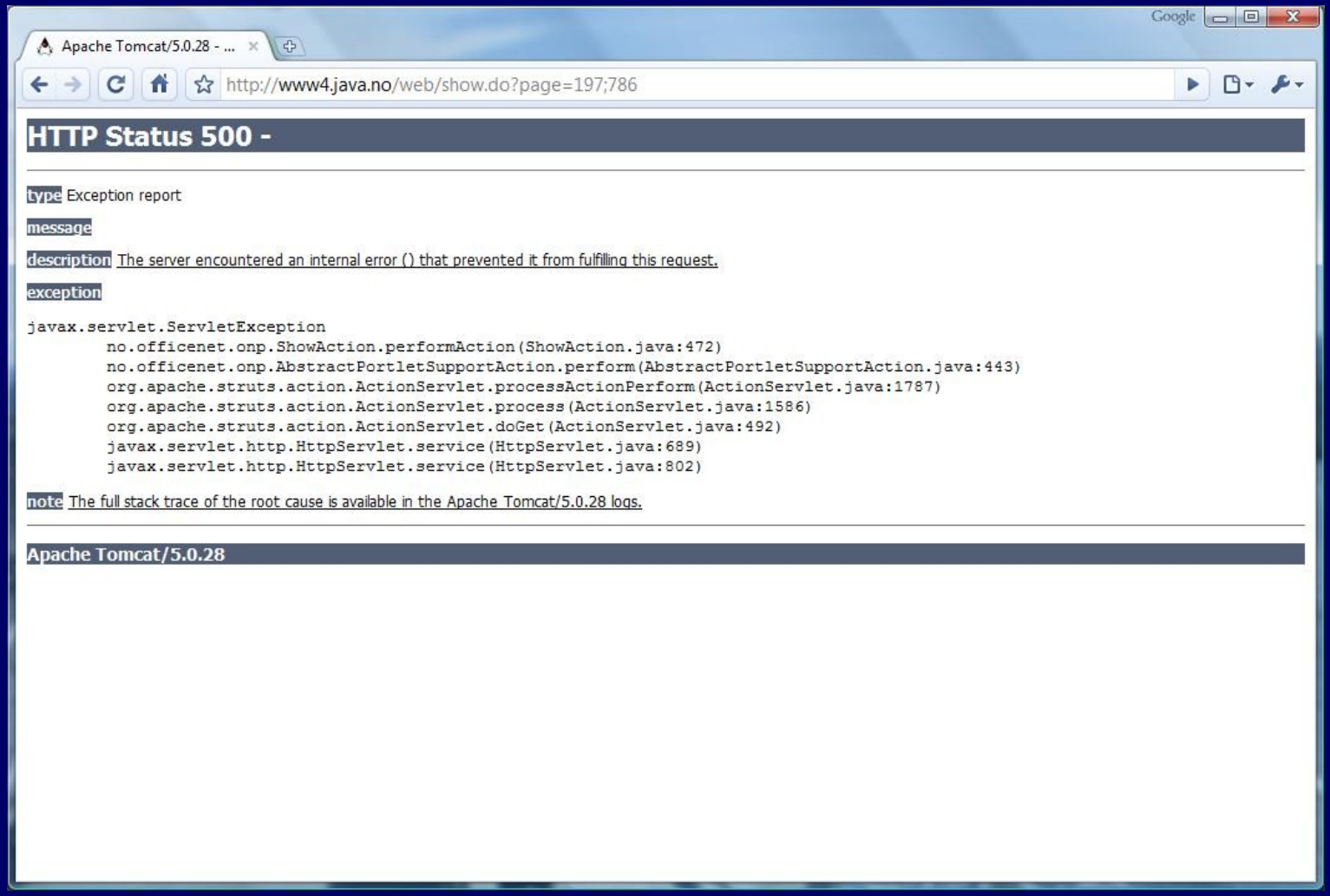
There have always been fairly severe size constraints on the Unix operating system and its software. Given the partially antagonistic desires for reasonable efficiency and expressive power, the size constraint has encouraged not only economy but a certain elegance of design.

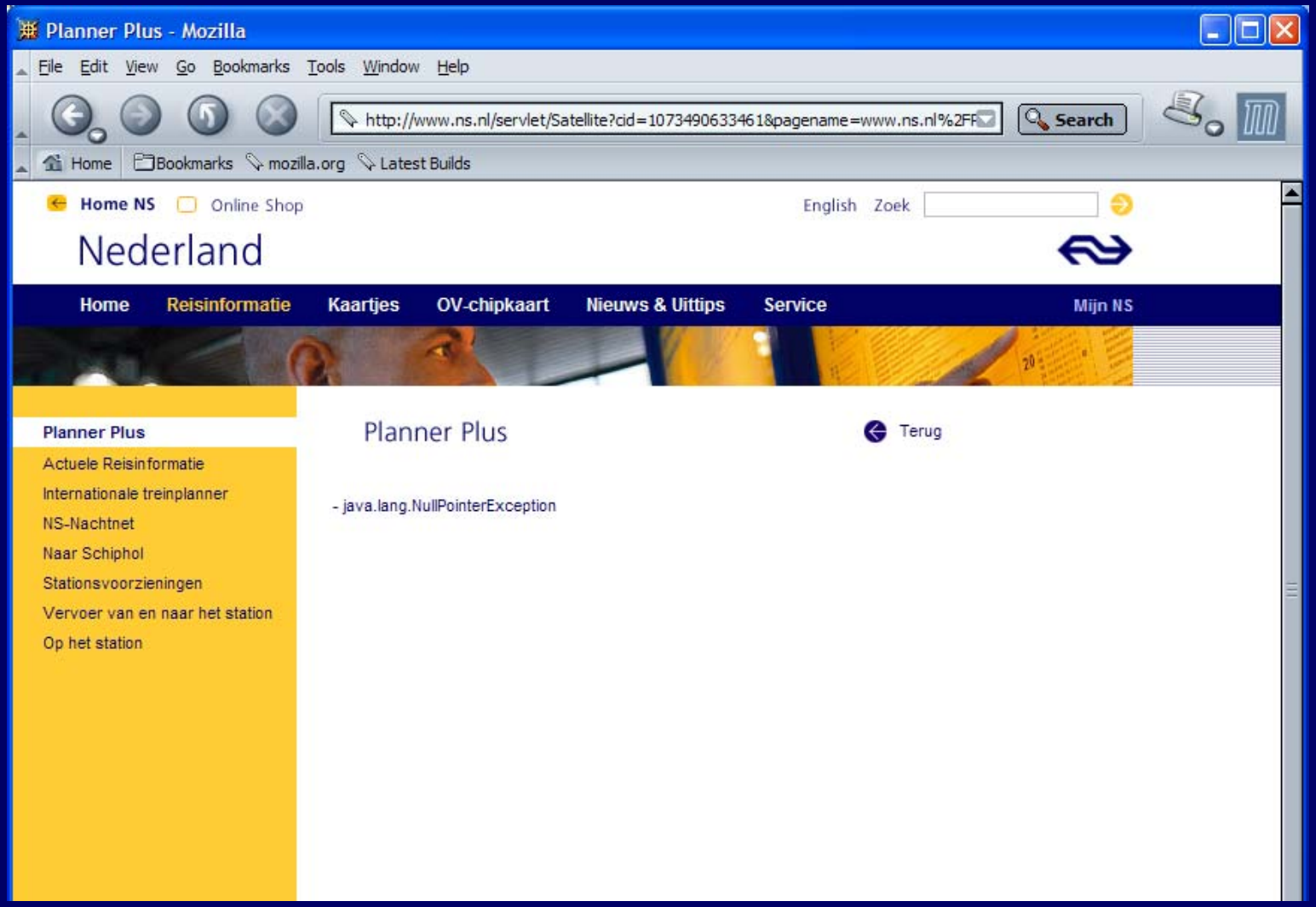
*Dennis Ritchie and Ken Thompson*

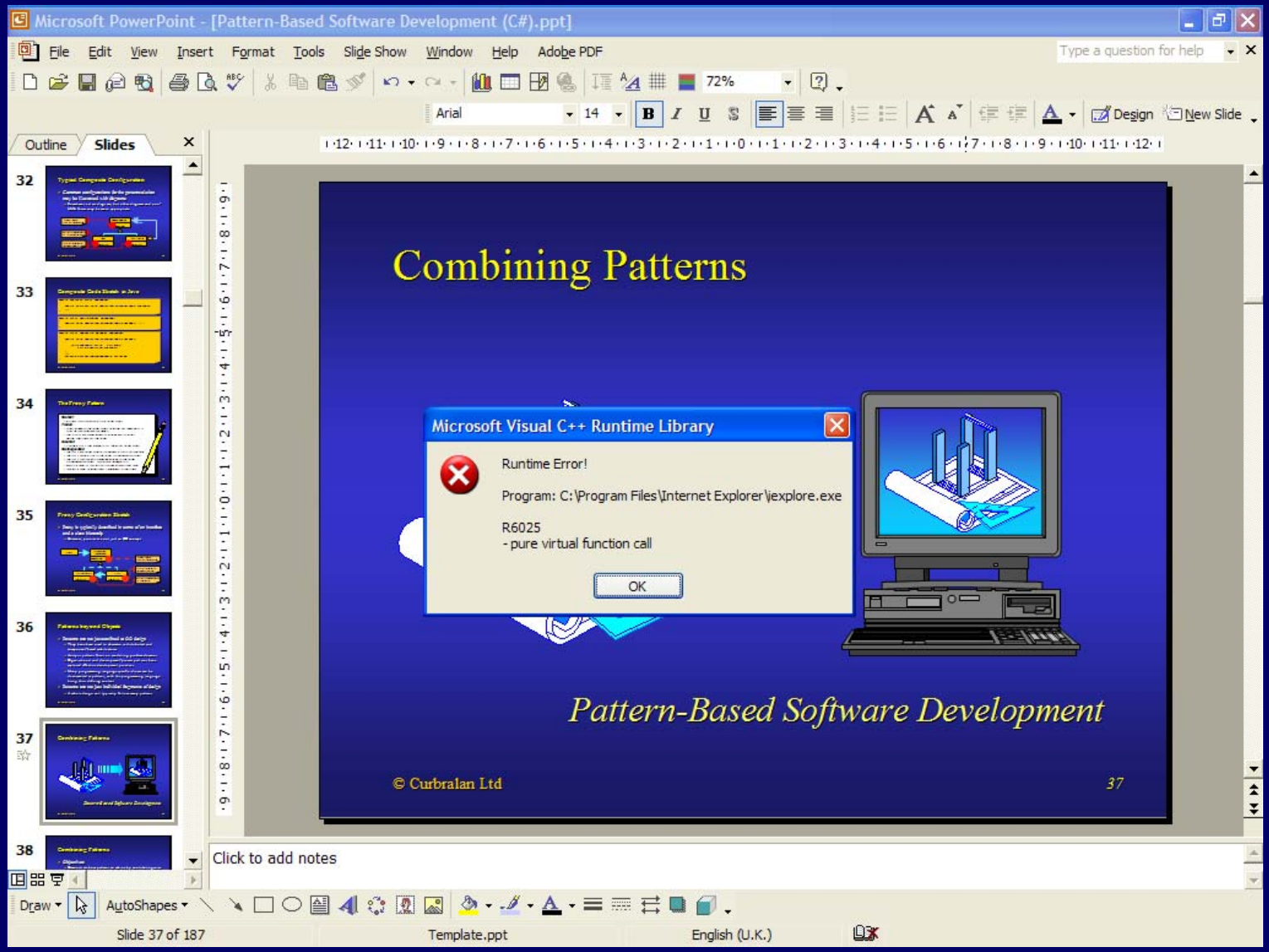
# Visibility

*The cause is hidden.*
*The effect is visible to all.*

**Ovid**


Google
Apache Tomcat/5.0.28 - ...
http://www4.java.no/web/show.do?page=197;786
HTTP Status 500 -
type Exception report
message
description The server encountered an internal error () that prevented it from fulfilling this request.
exception
javax.servlet.ServletException
no.officenet.onp.ShowAction.performAction(ShowAction.java:472)
no.officenet.onp.AbstractPortletSupportAction.perform(AbstractPortletSupportAction.java:443)
org.apache.struts.action.ActionServlet.processActionPerform(ActionServlet.java:1787)
org.apache.struts.action.ActionServlet.process(ActionServlet.java:1586)
org.apache.struts.action.ActionServlet.doGet(ActionServlet.java:492)
javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
note The full stack trace of the root cause is available in the Apache Tomcat/5.0.28 logs.
Apache Tomcat/5.0.28

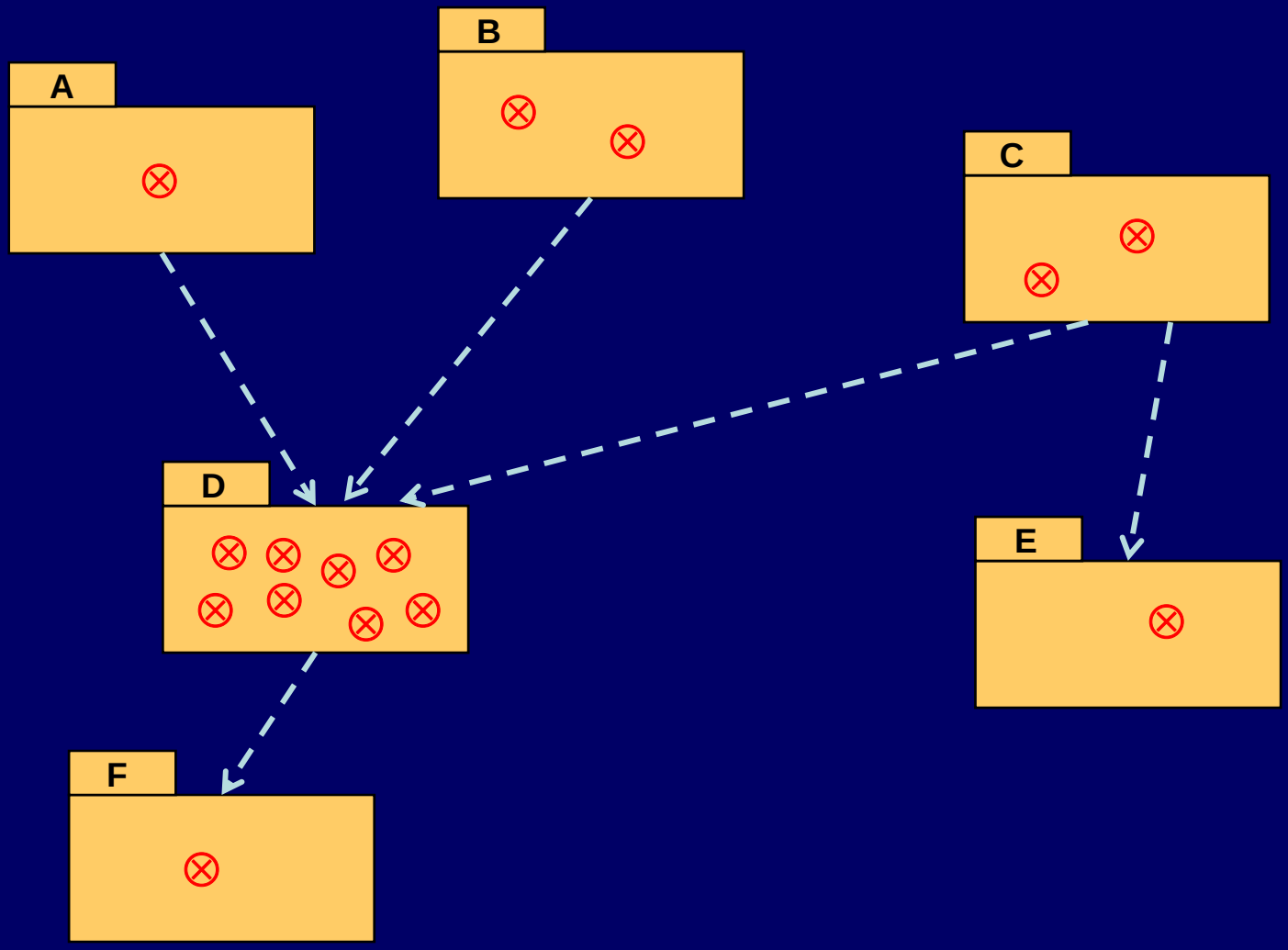

Planner Plus - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://www.ns.nl/servlet/Satellite?cid=1073490633461&pagename=www.ns.nl%2FP
Search
Home Bookmarks mozilla.org Latest Builds
Home NS
Online Shop
English
Zoek
Nederland
Home
Reisinformatie
Kaartjes
OV-chipkaart
Nieuws & Uittips
Service
Mijn NS
Planner Plus
Actuele Reisinformatie
Internationale treinplanner
NS-Nachtnet
Naar Schiphol
Stationsvoorzieningen
Vervoer van en naar het station
Op het station
Planner Plus
Terug
- java.lang.NullPointerException

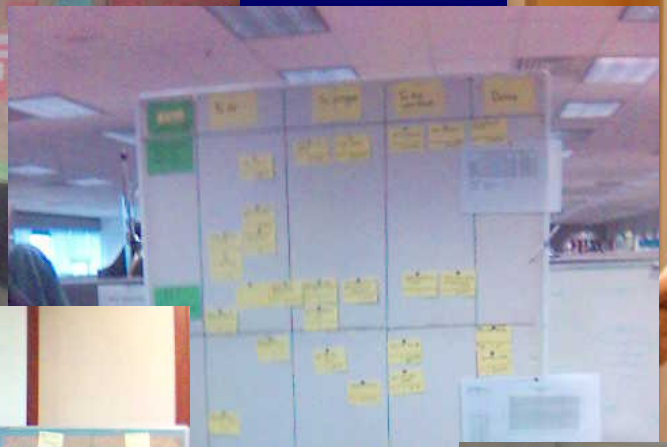

Microsoft PowerPoint - [Pattern-Based Software Development (C#).ppt]
File Edit View Insert Format Tools Slide Show Window Help Adobe PDF
Type a question for help
72%
Arial 14
Design New Slide
Outline Slides
32
33
34
35
36
37
38
Combining Patterns
Microsoft Visual C++ Runtime Library
Runtime Error!
Program: C:\Program Files\Internet Explorer\iexplore.exe
R6025
- pure virtual function call
OK
Pattern-Based Software Development
© Curbralan Ltd
37
Click to add notes
Draw AutoShapes
Slide 37 of 187
Template.ppt
English (U.K.)

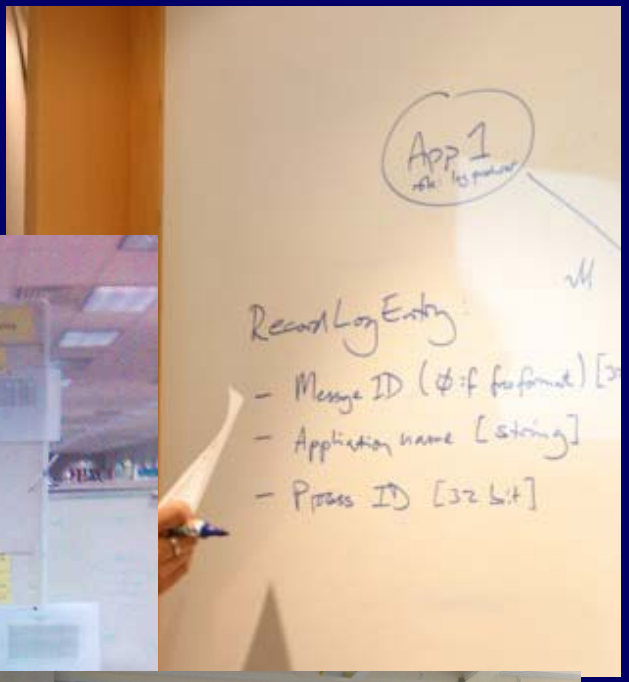

Using autodetected IRQ (11) to improve performance.
ifcust (PC/TCP Class 1 packet driver - DIX Ethernet) init
5 free packets of length 160, 5 free packets of length 1
The kernel is using asynchronous sends
The Resident Module occupies 0 bytes of conventional mem

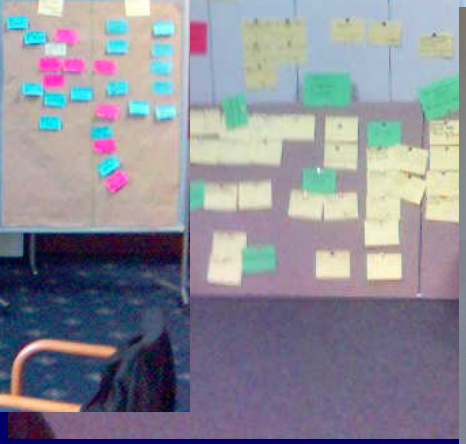
*Test Early.*
*Test Often.*
*Test Automatically.*

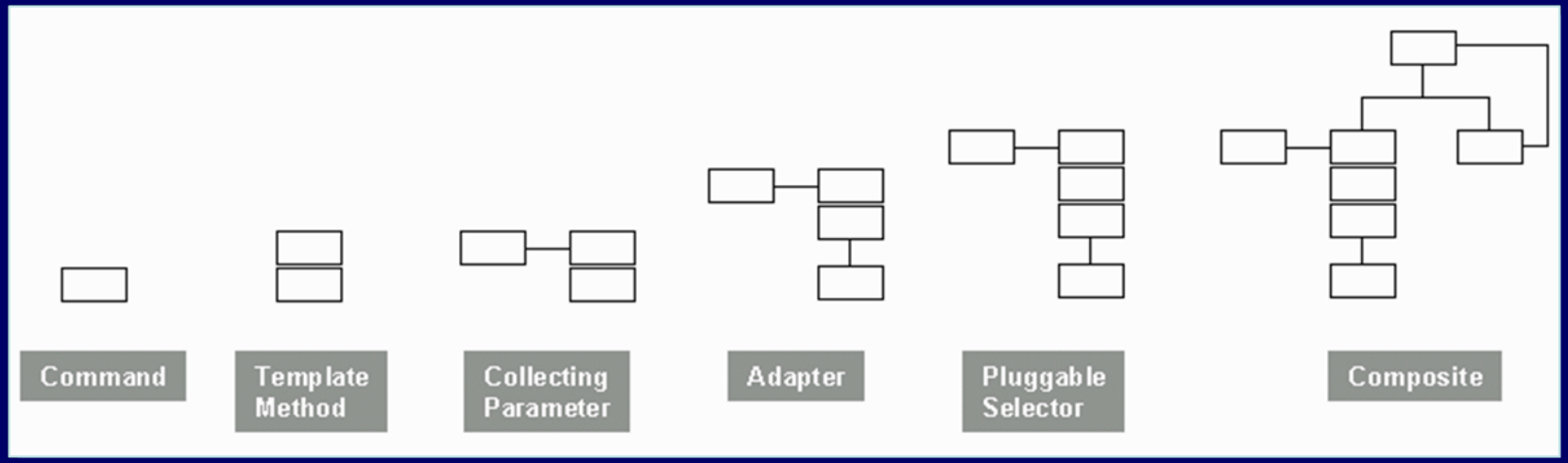
*Andrew Hunt and David Thomas*
*The Pragmatic Programmer*

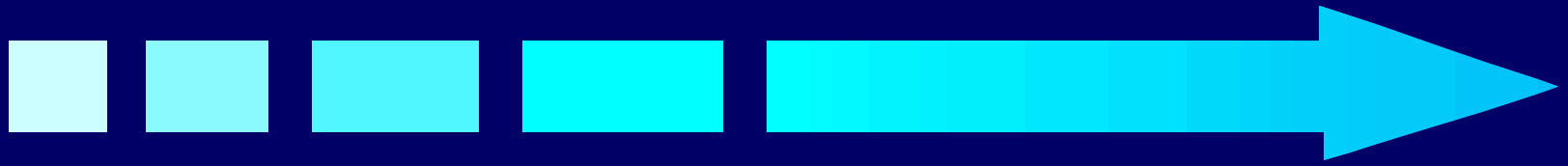

A
B
C
D
E
F




App 1
Record Log Entry
- Message ID
- Application name [string]
- Process ID [32 bit]

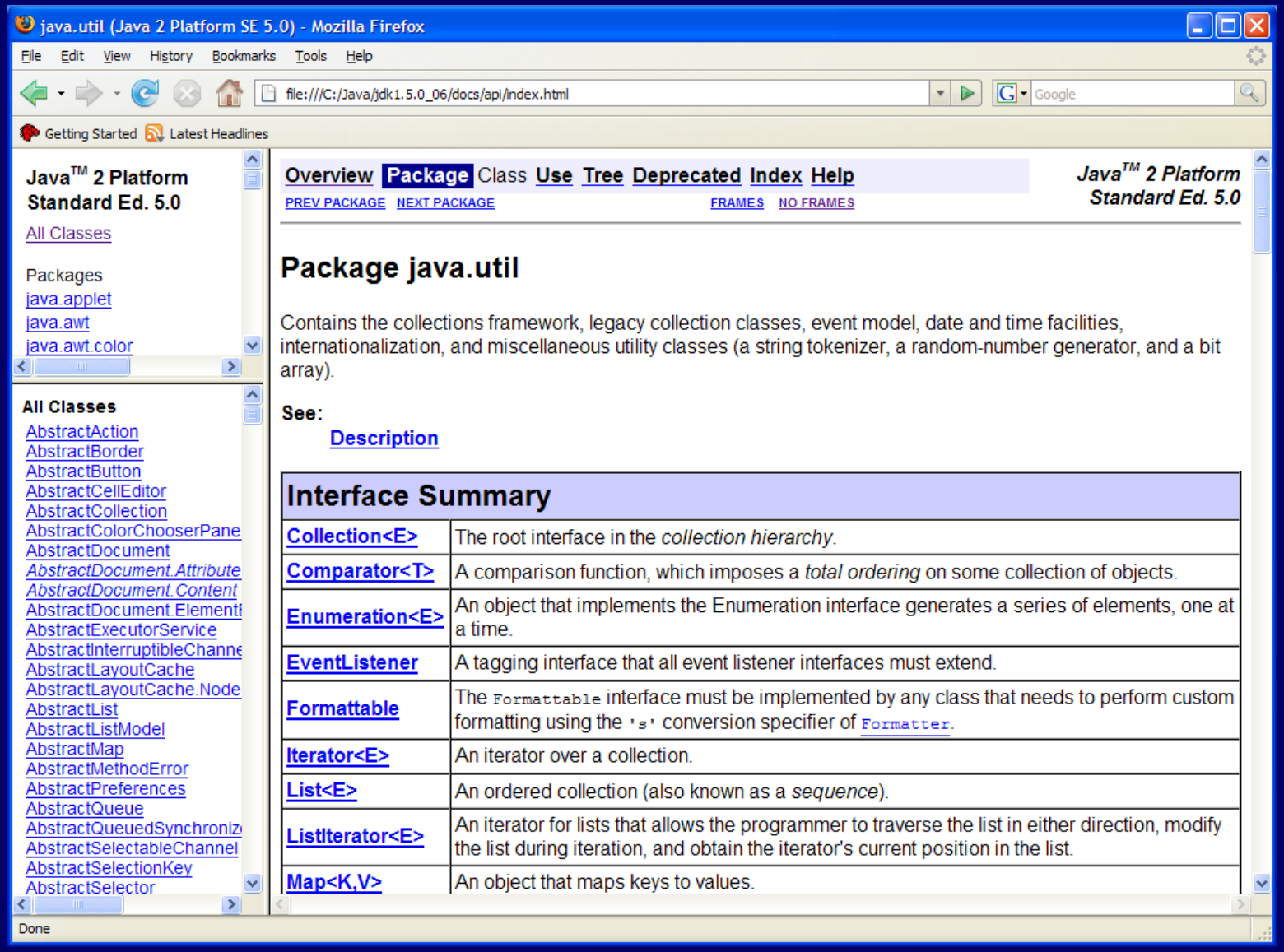


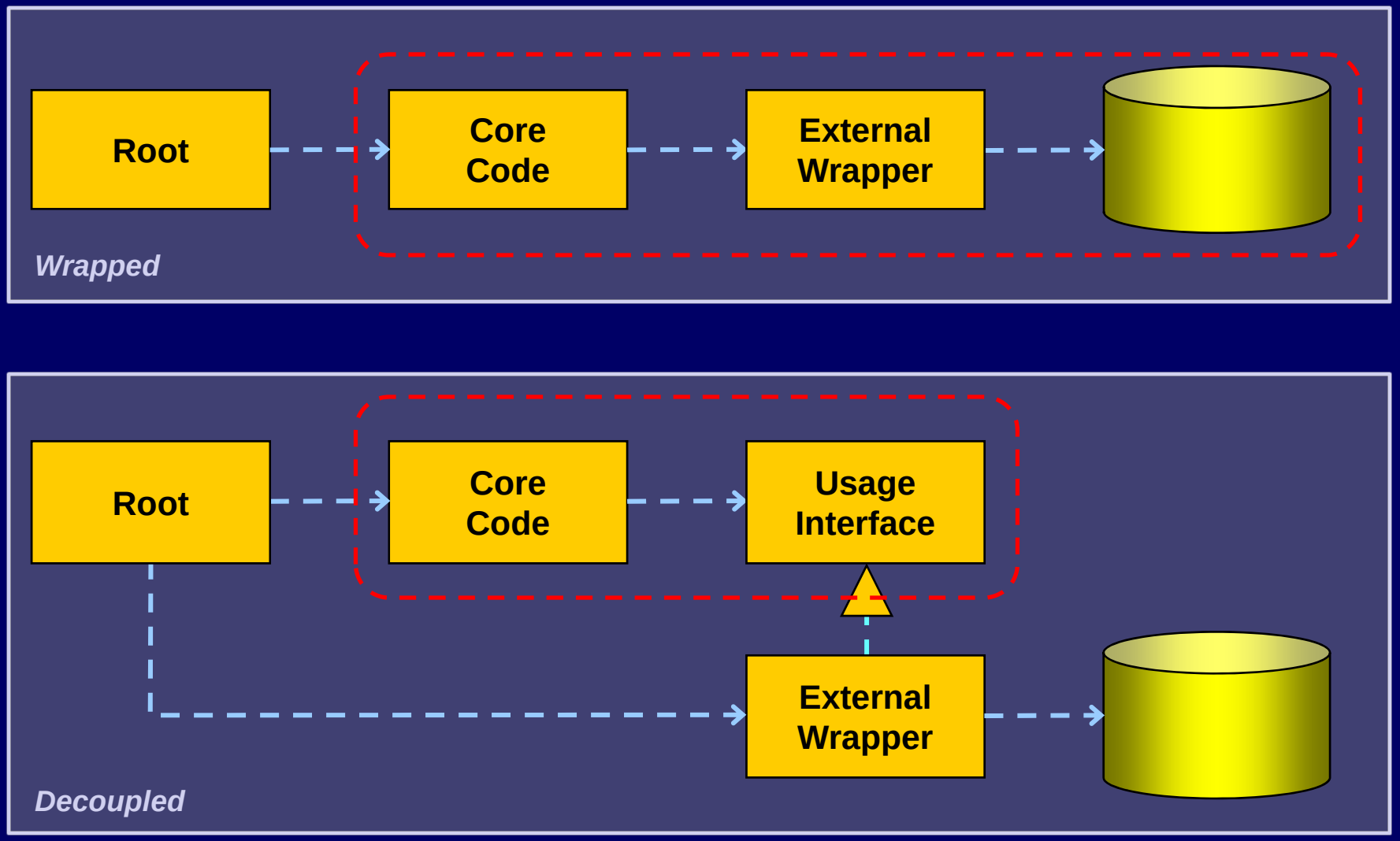


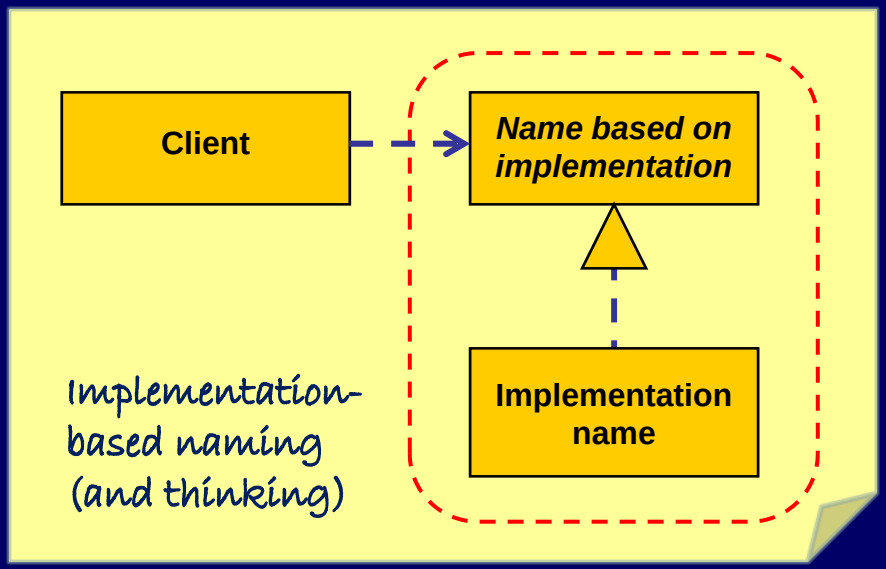



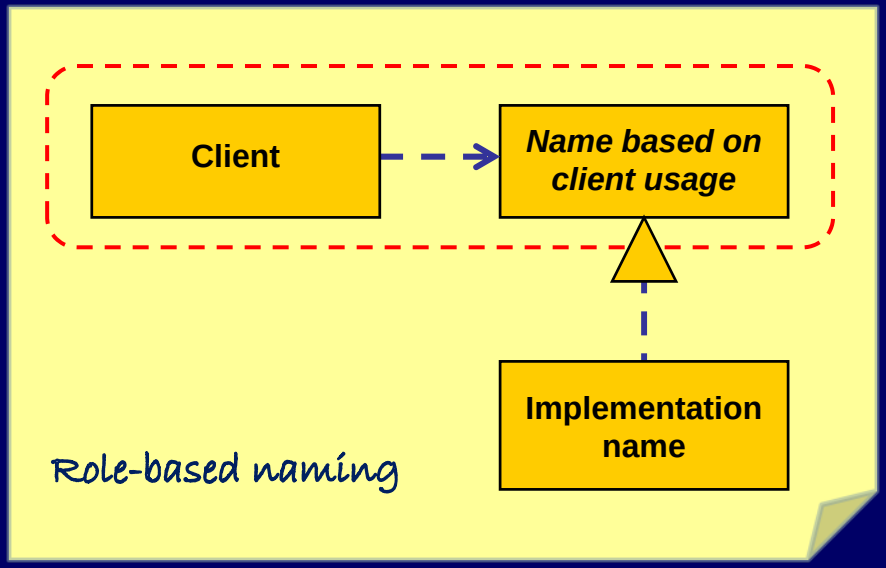
**Patterns help to mark out and make design decisions visible and communicable. A pattern focuses on context and forces that relate to a design situation. Pattern sequences highlight incremental growth, making progress visible.**

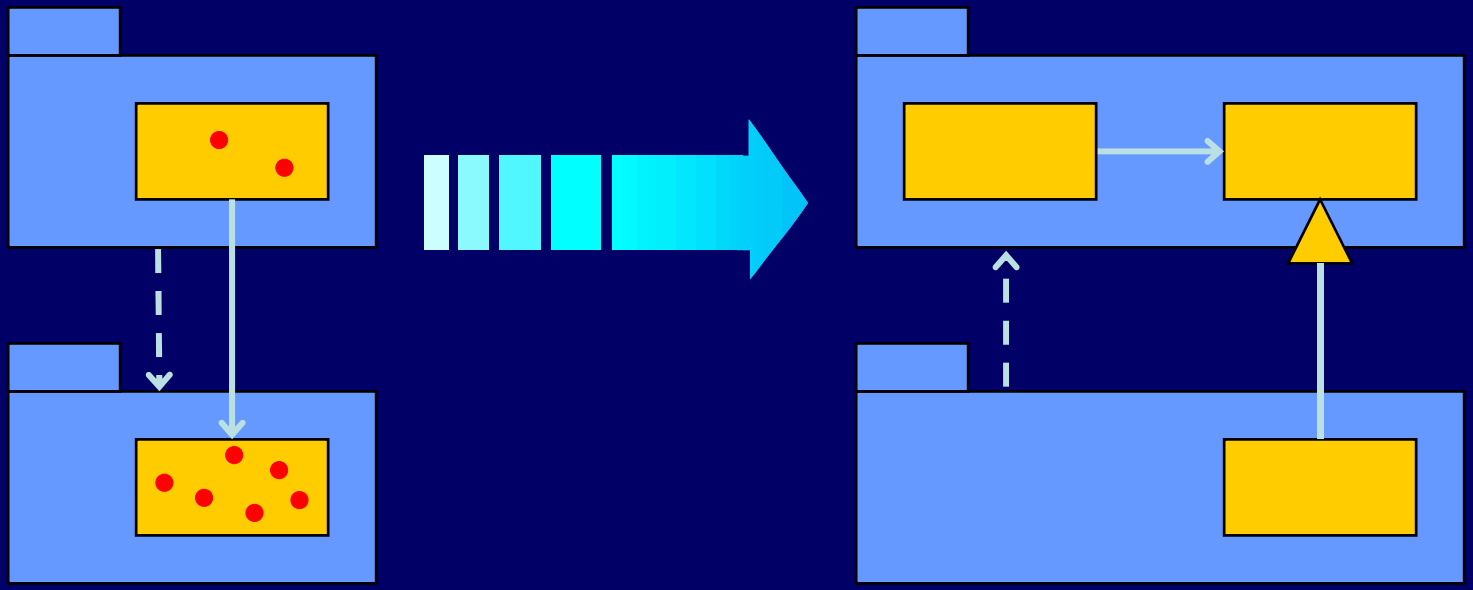
# Spacing


java.util (Java 2 Platform SE 5.0) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Java/jdk1.5.0_06/docs/api/index.html
Google
Getting Started
Latest Headlines
Java™ 2 Platform Standard Ed. 5.0
All Classes
Packages
java.applet
java.awt
java.awt.color
All Classes
AbstractAction
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPane
AbstractDocument
AbstractDocument.Attribute
AbstractDocument.Content
AbstractDocument.Element
AbstractExecutorService
AbstractInterruptibleChanne
AbstractLayoutCache
AbstractLayoutCache.Node
AbstractList
AbstractListModel
AbstractMap
AbstractMethodError
AbstractPreferences
AbstractQueue
AbstractQueuedSynchroniz
AbstractSelectableChannel
AbstractSelectionKey
AbstractSelector
Overview Package Class Use Tree Deprecated Index Help
PREV PACKAGE NEXT PACKAGE
FRAMES NO FRAMES
Java™ 2 Platform Standard Ed. 5.0
Package java.util
Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).
See:
Description
Interface Summary
Collection<E> The root interface in the collection hierarchy.
Comparator<T> A comparison function, which imposes a total ordering on some collection of objects.
Enumeration<E> An object that implements the Enumeration interface generates a series of elements, one at a time.
EventListener A tagging interface that all event listener interfaces must extend.
Formattable The Formattable interface must be implemented by any class that needs to perform custom formatting using the 's' conversion specifier of Formatter.
Iterator<E> An iterator over a collection.
List<E> An ordered collection (also known as a sequence).
ListIterator<E> An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.
Map<K,V> An object that maps keys to values.
Done


Root
Core Code
External Wrapper
Wrapped
Root
Core Code
Usage Interface
External Wrapper
Decoupled


Client
Name based on implementation
Implementation name
Implementation-based naming (and thinking)


Client
Name based on client usage
Implementation name
Role-based naming



Consider a number of possible change scenarios and mark affected components.

Dependency inversion allows a design's dependencies to be reversed, loosened and manipulated at will, which means that dependencies can be aligned with known or anticipated stability.

# Symmetry

***symmetry*** *the quality of being made up of exactly similar parts facing each other or around an axis.*

- *correct or pleasing proportion of the parts of a thing.*
- *similarity of exact correspondence between different things.*

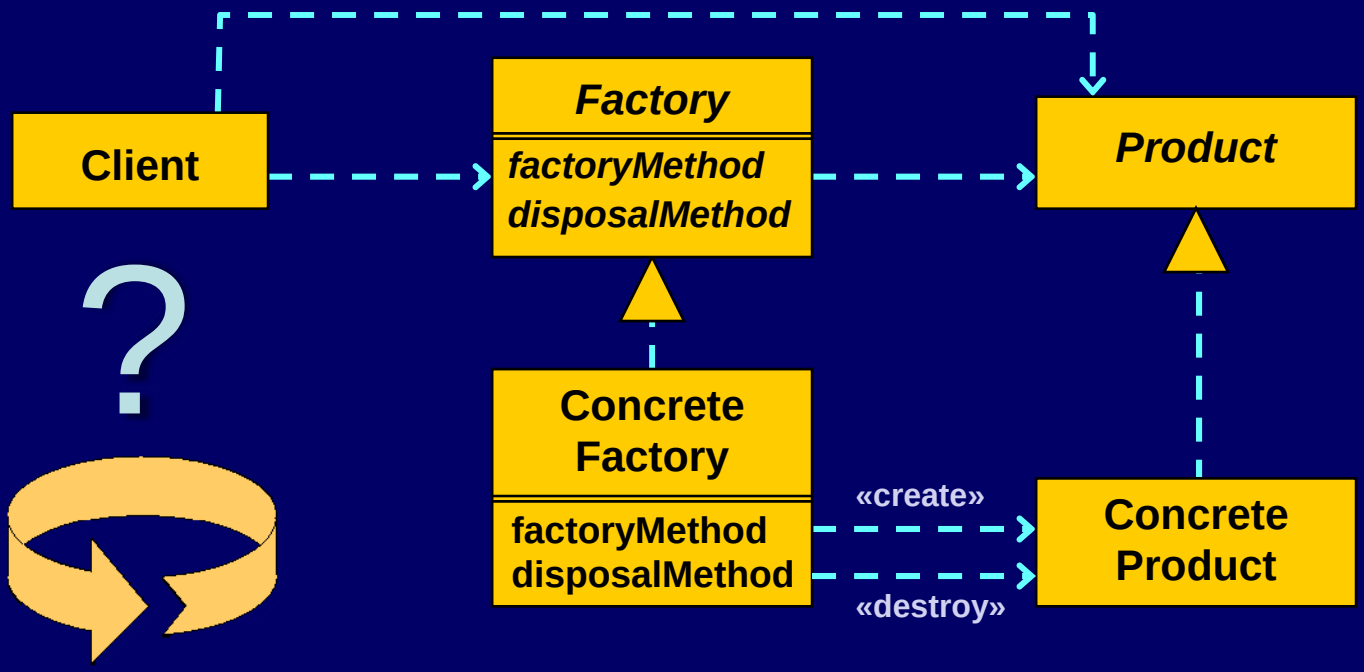
**The New Oxford Dictionary of English**

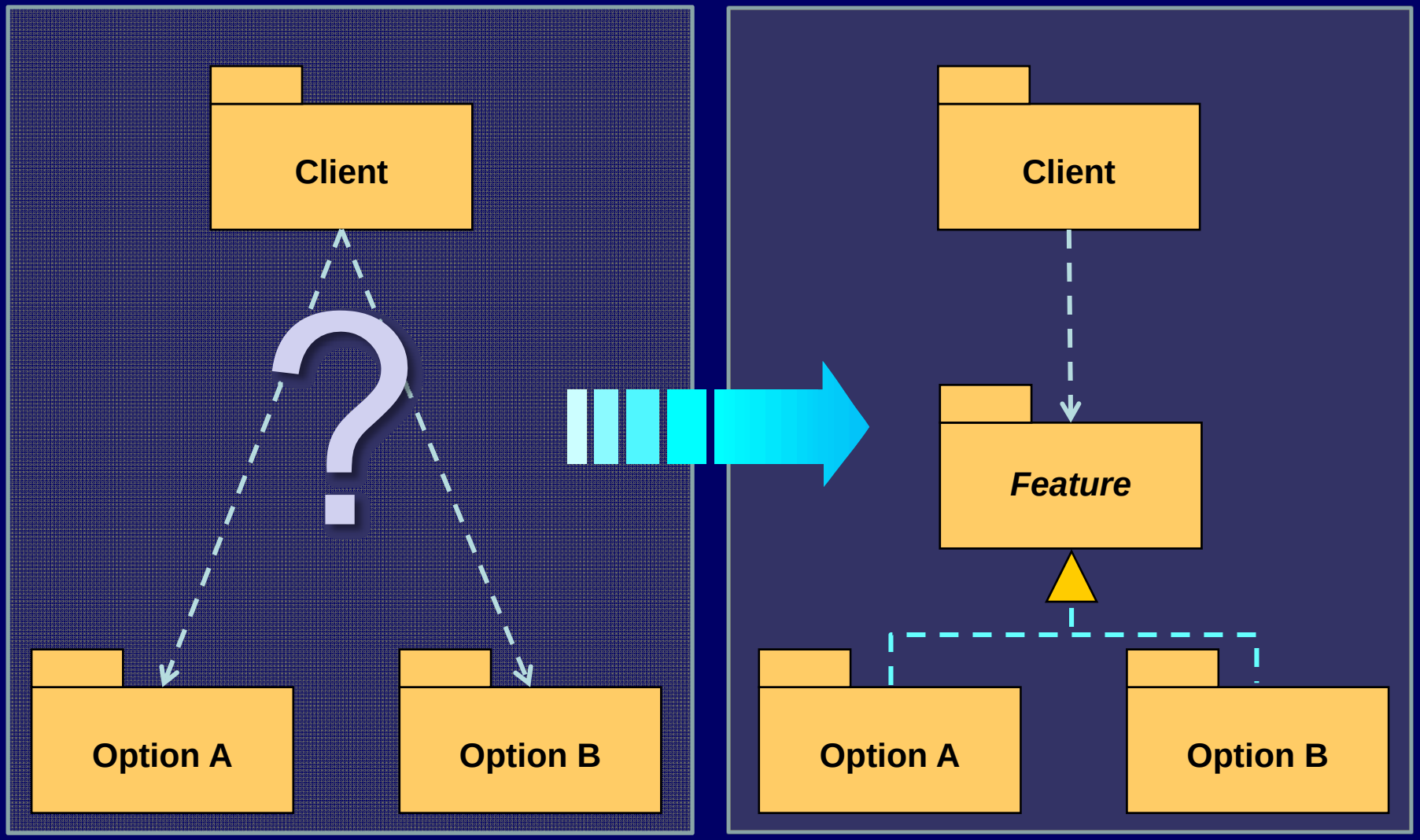
*When in doubt, make it symmetrical.*

*Christopher Alexander*

La symétrie, c'est l'ennui, et l'ennui est le fond même du deuil. Le désespoir baîlle.

Victor Hugo


Client
Factory
factoryMethod
disposalMethod
Product
?
Concrete
Factory
factoryMethod
disposalMethod
«create»
«destroy»
Concrete
Product


Client
?
Option A
Option B
Client
Feature
Option A
Option B

# Emergence

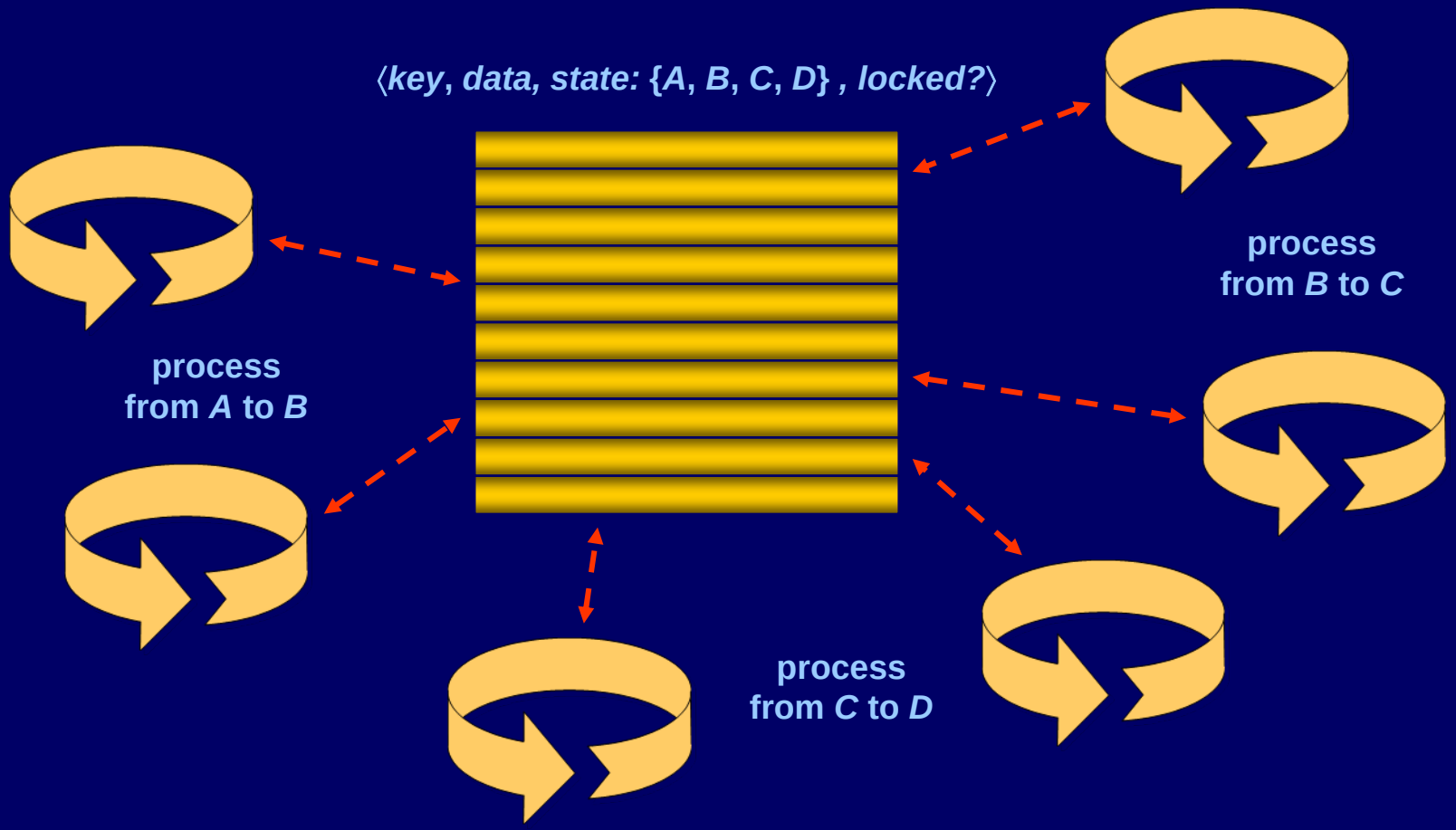
Flocking is a common demonstration of emergent behaviour.... The result is alike to a flock of birds, a school of fish, or swarm of insects.

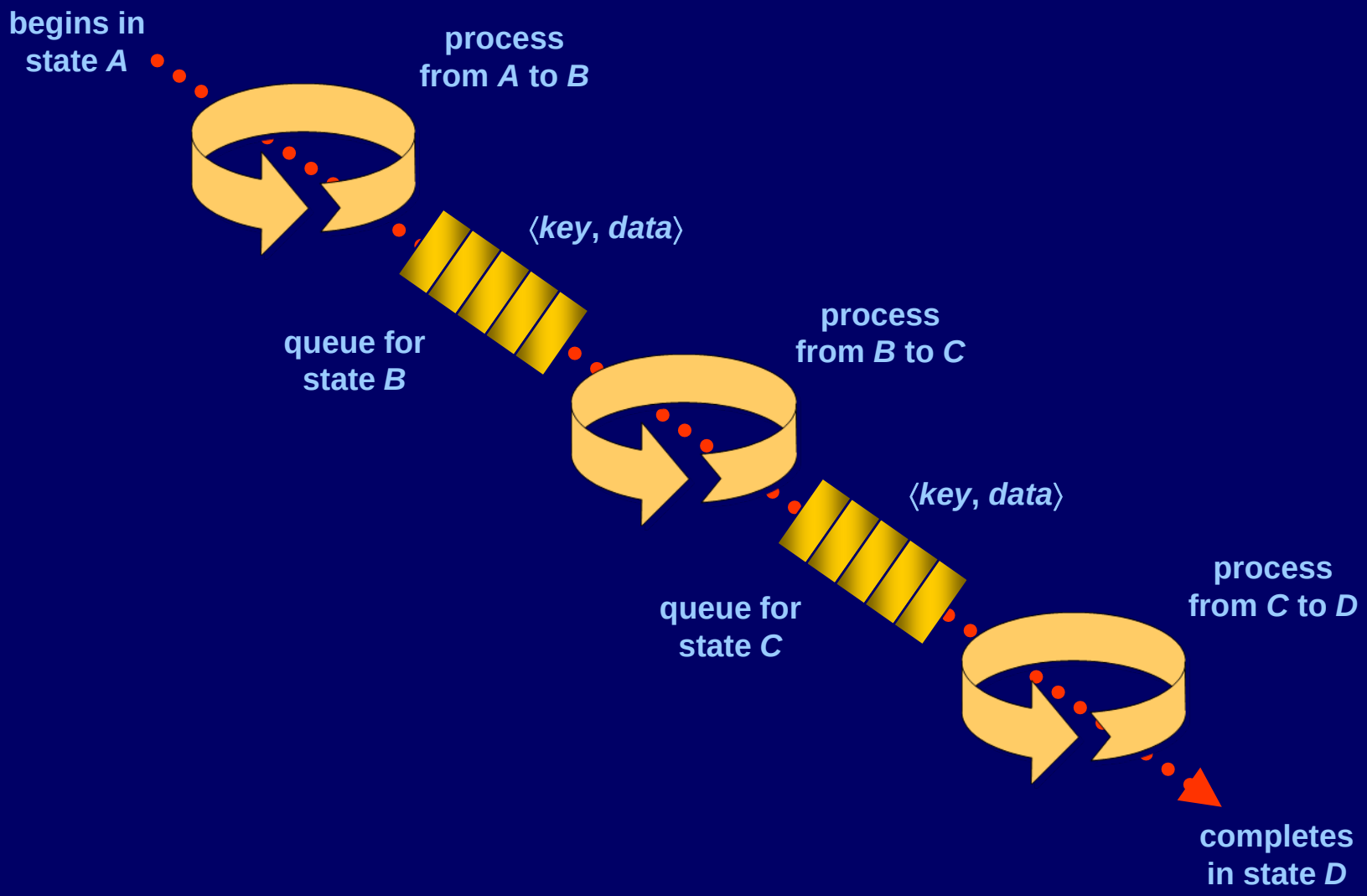
Basic flocking is controlled by three simple rules:

1. *Separation* avoid crowding neighbours
2. *Alignment* steer towards average heading of neighbours
3. *Cohesion* steer towards average position of neighbours

With these three simple rules, the flock moves in an extremely realistic way, creating complex motion and interaction that would be extremely hard to create otherwise.

Wikipedia


⟨key, data, state: {A, B, C, D} , locked?⟩
process
from A to B
process
from B to C
process
from C to D


begins in
state A
process
from A to B
⟨key, data⟩
queue for
state B
process
from B to C
⟨key, data⟩
queue for
state C
process
from C to D
completes
in state D

*If practice is unprincipled there is no coordination and there is discord. When it is principled, there is balance, harmony and union. Perhaps all life aspires to the condition of music.*

**Aubrey Meyer**