



A Lesson In Low-Defect Software

- or -

A Journey From A Quick Hack To A High-Reliability Database Engine

and how you can use the same techniques to reduce
the number of bugs in your own software projects

D. Richard Hipp
2009-03-10

What Is SQLite?

Public domain

Serverless

Small footprint

Embedded

Transactional

Single-file Database

ACID

SQL Database Engine

Robust

Stable, cross-platform file format

Zero-administration

Efficient

Easy to integrate

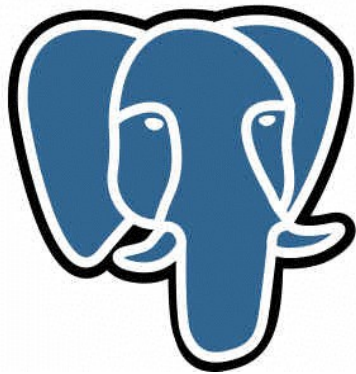
SQL?

What's That?

- “Structured Query Language”
- A high-level language for interacting with databases
- The most widely known programming language in the world.

Why SQLite?

PostgreSQL



Apache Derby 



ORACLE®



Informix®

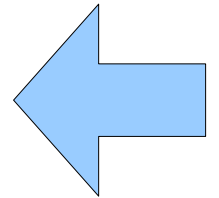
Microsoft®
SQL Server™

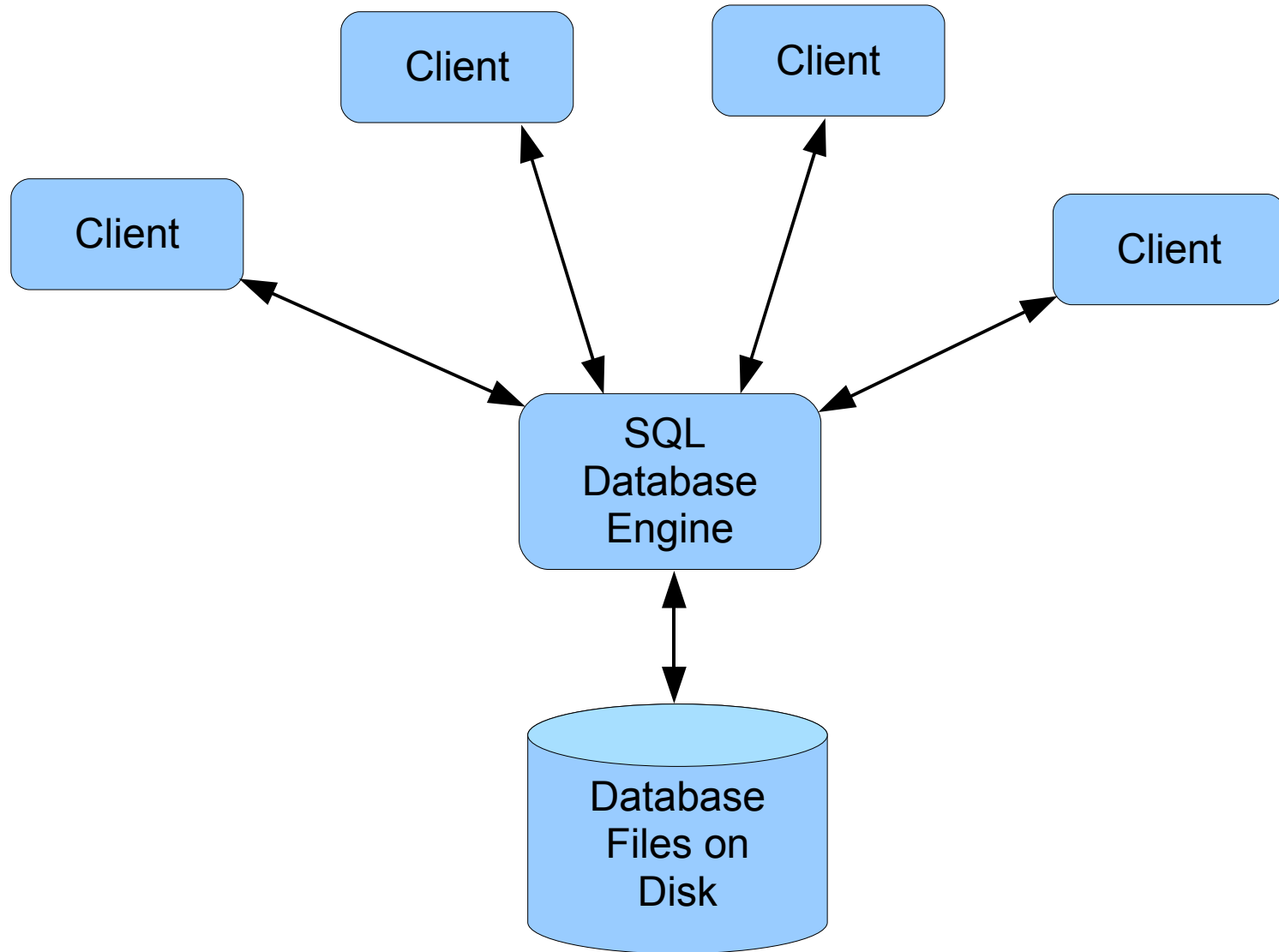


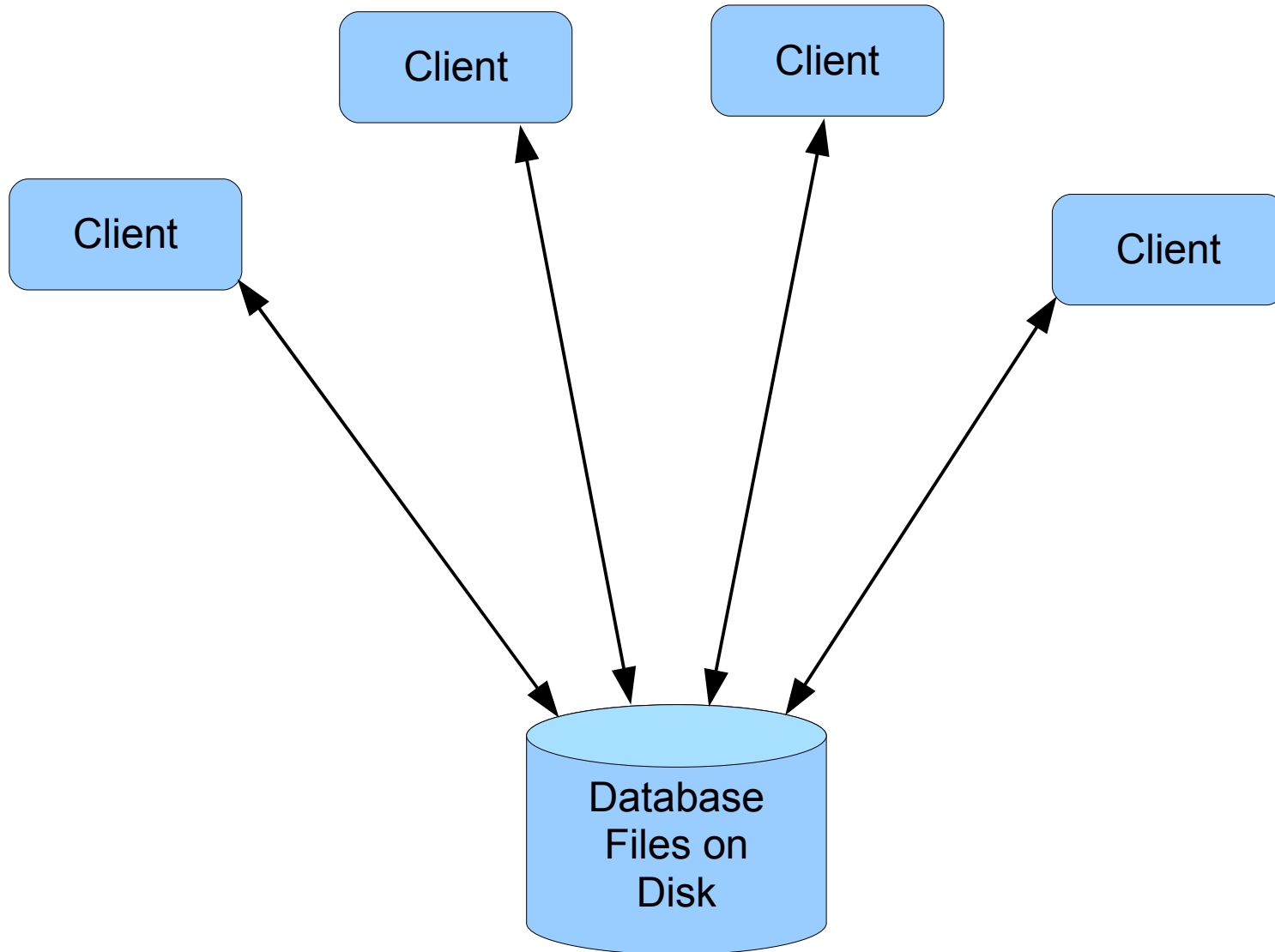
- Serverless
- Zero-administration
- Portable file format
- Small footprint
- Public domain

SQLite

- Serverless
- Zero-administration
- Portable file format
- Small footprint
- Public domain







Advantages of Serverless

- No background server process
- No configuration files
- No IPC
- No security issues
- Nothing to start, shutdown, or reboot
- Nothing to go wrong or need tending to

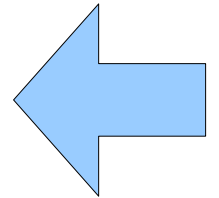
Advantages of Serverless

- No background server process
- No configuration files
- No IPC
- No security issues
- Nothing to start, shutdown, or reboot
- Nothing to go wrong or need tending to

“Zero-administration”

SQLite

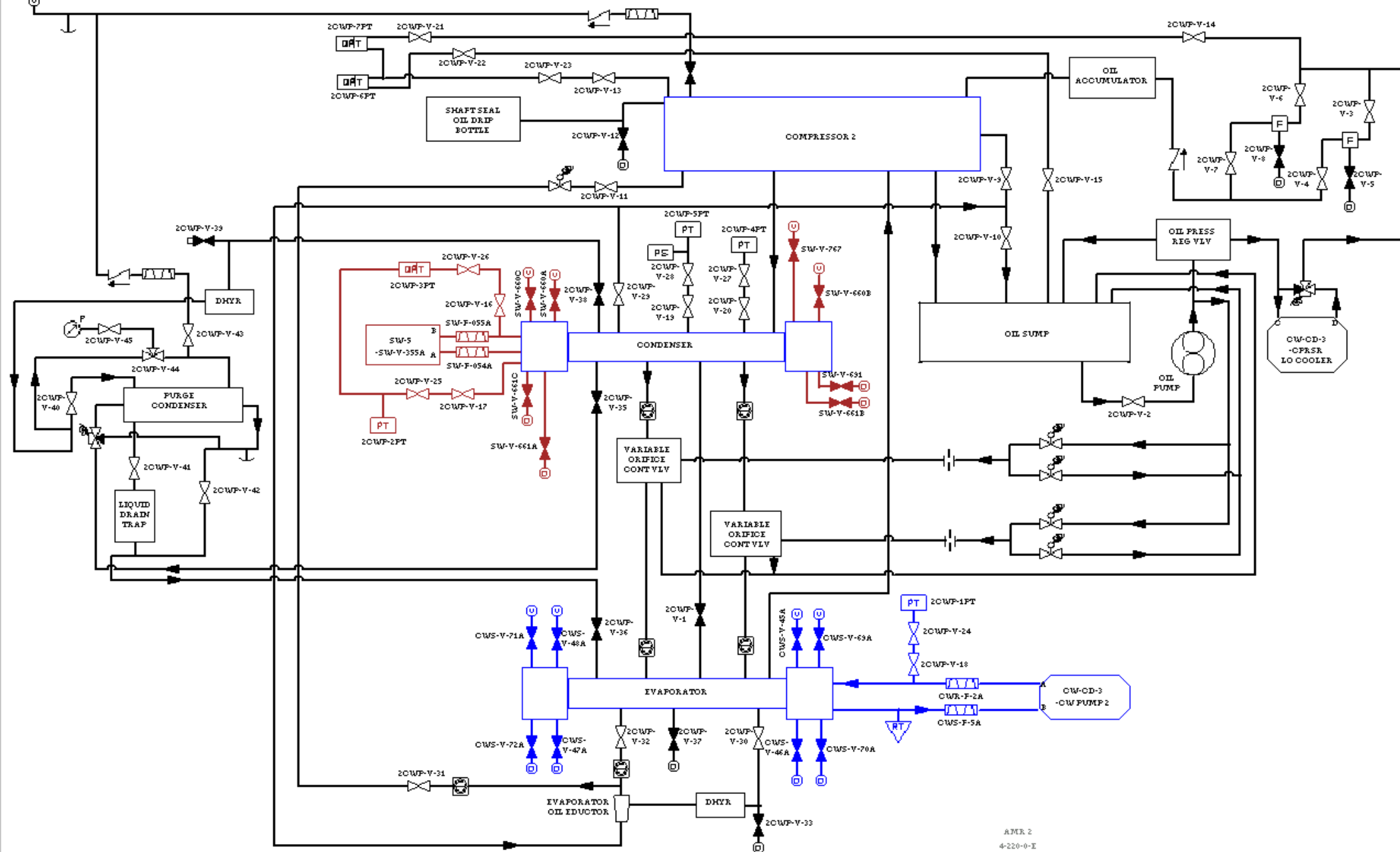
- Serverless
- Zero-administration
- Portable file format
- Small footprint
- Public domain



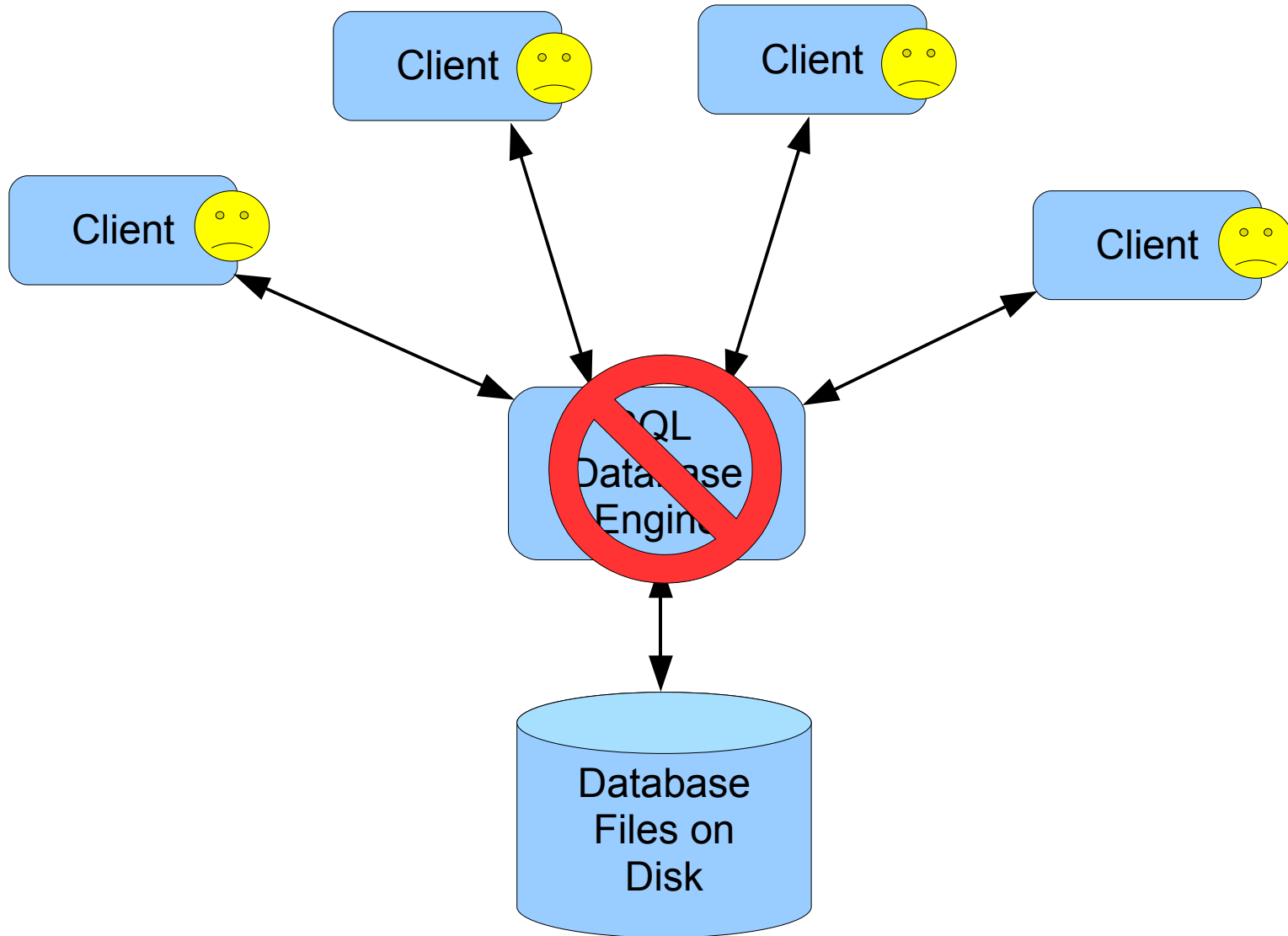


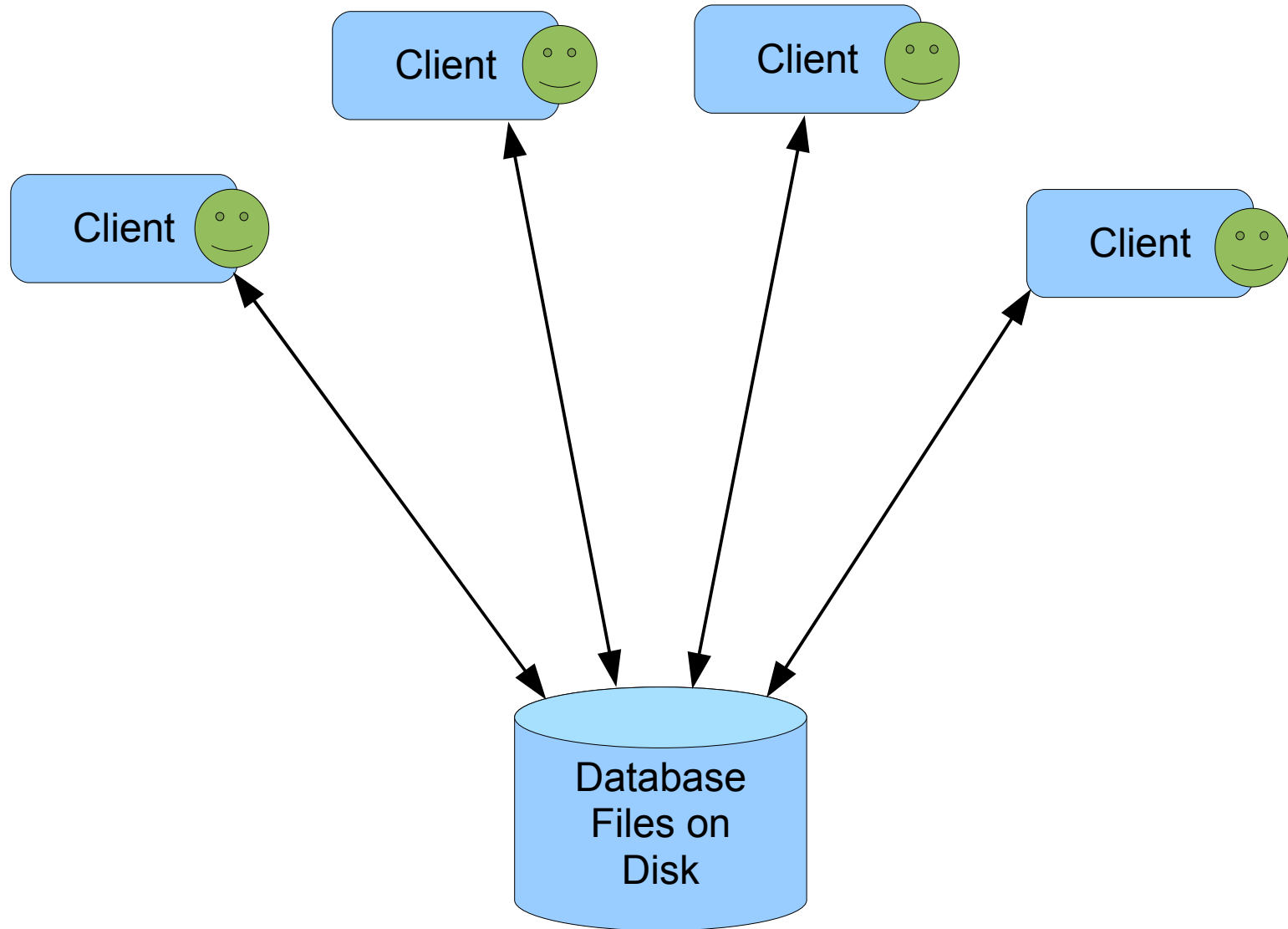
Database Administrators

TO ATMOSPHERE

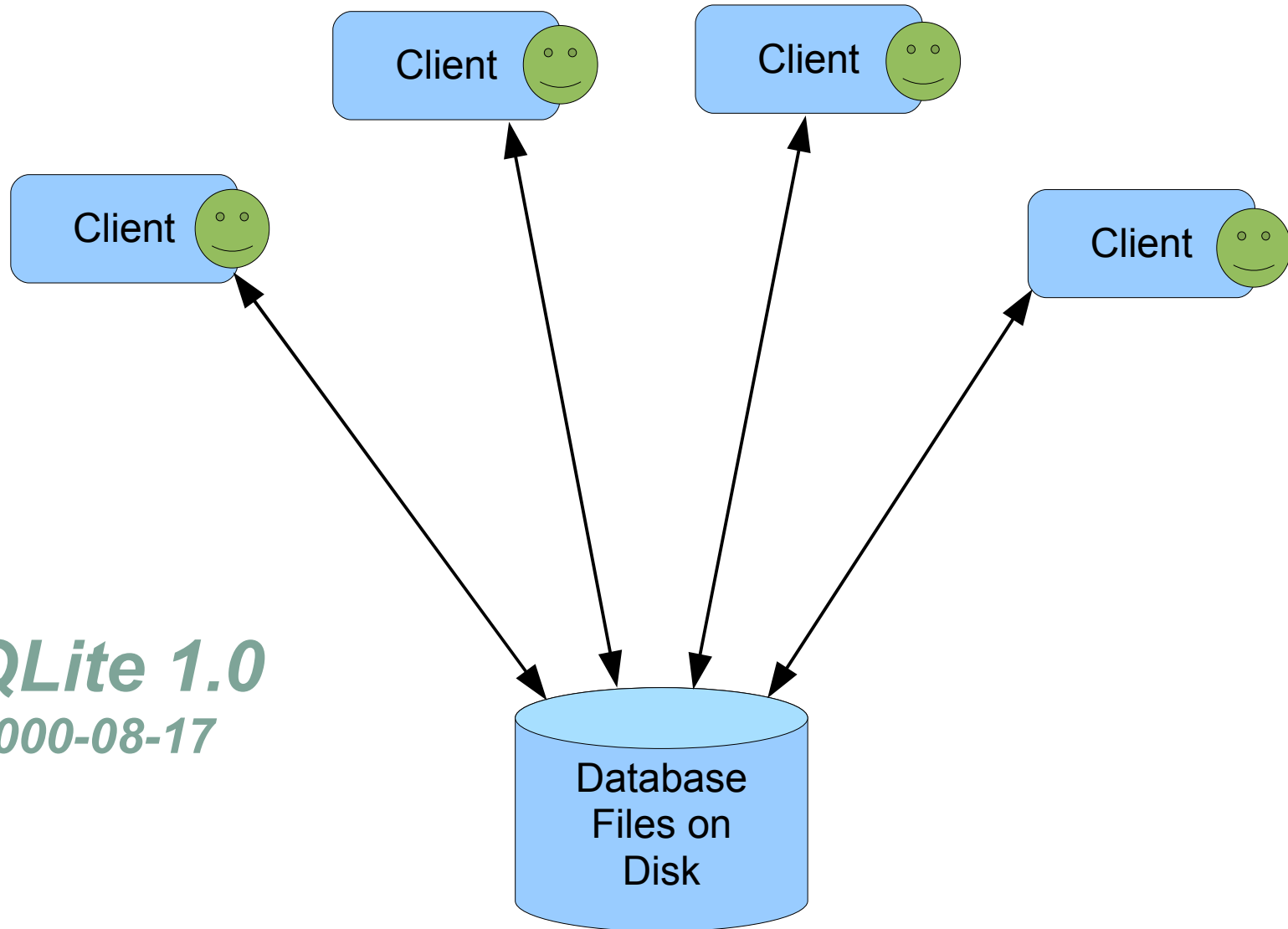








SQLite 1.0
2000-08-17



Another way to think of
SQLite in relation to
traditional SQL database
engines....





ORACLE®

is to

as

SQLite



is to

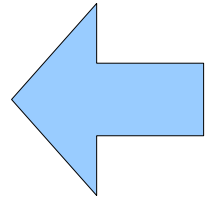


- SQLite does not compete with Oracle

- SQLite does not compete with Oracle
- SQLite competes with **fopen()**

SQLite

- Serverless
- Zero-administration
- Portable file format
- Small footprint
- Public domain

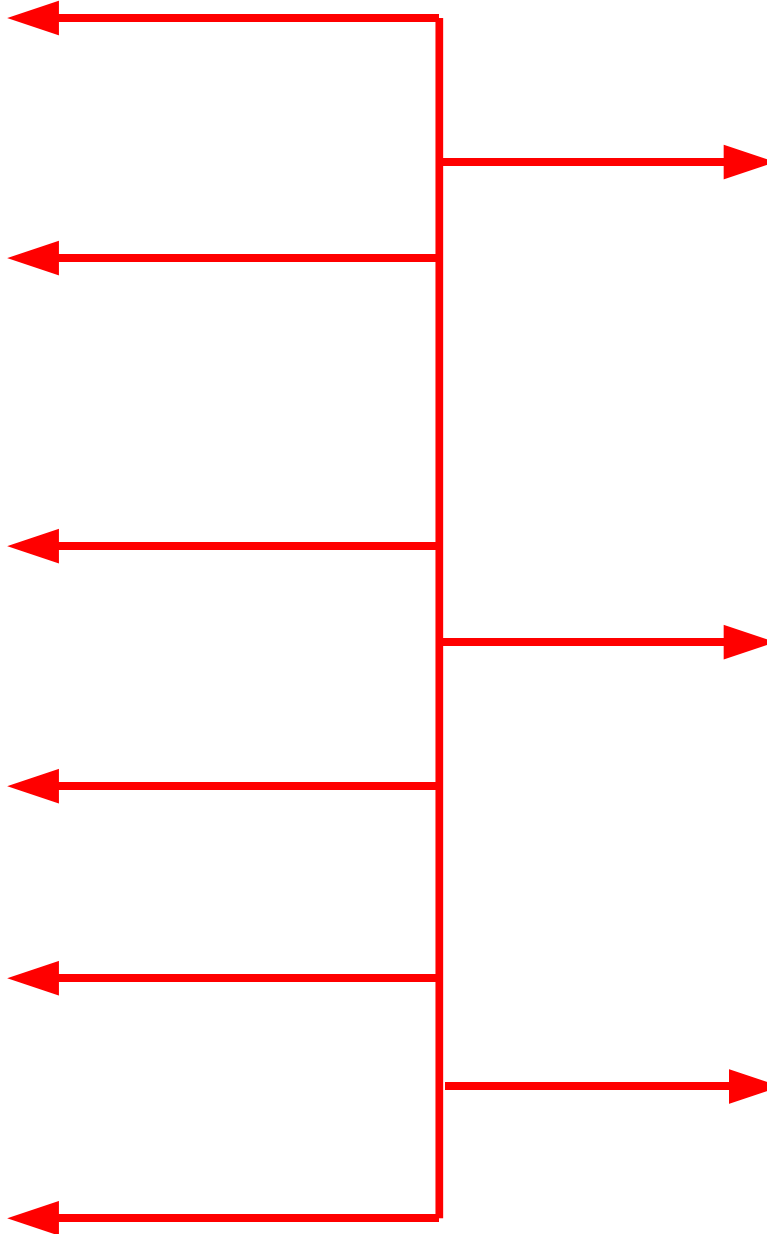


Portable File Format

- A database is a single ordinary disk file
- No special naming conventions or required file suffixes
- Cross-platform: big/little-endian and 32/64-bit
- Backwards compatible through 3.0.0
- Promise to keep it compatible moving forward
- Not tied to any particular programming language.

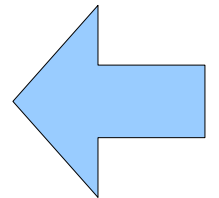


Mac



SQLite

- Serverless
- Zero-administration
- Portable file format
- Small footprint
- Public domain



Small Footprint

`gcc -Os -DSQLITE_THREADSAFE=0`

272 KiB

`gcc -O3 -DSQLITE_ENABLE_FTS3=1 -DSQLITE_ENABLE_RTREE=1`

789 KiB

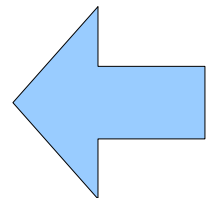
Single Source Code File

- The “amalgamation” source code file: **sqlite3.c**
- About 60,000 lines of ANSI C code
- 3.5 MB
- No other library dependencies on than standard library routines:
 - memcpy(), memset(), malloc(), free(), etc
- Very simple to add to a larger C program

```
drh@elly:~/fossil/m1/src> ls
add.c          content.c      makeheaders.html  schema.c        timeline.c
admin.c        db.c          makemake.tcl      setup.c         tkt.c
allrepo.c     delta.c       manifest.c        setup.c.bu1     tktsetup.c
bag.c         deltacmd.c    md5.c            sha1.c         translate.c
blob.c        descendants.c  merge3.c         shun.c         undo.c
branch.c      diff.c       merge.c          sqlite3.c       update.c
browse.c     diffcmd.c    mindex.c        sqlite3.h       url.c
cgi.c        doc.c       my_page.c       stat.c         user.c
checkin.c    encode.c     name.c          style.c        verify.c
checkout.c   file.c      pivot.c        sync.c        VERSION
clearsign.c http.c      pqueue.c       tag.c         vfile.c
clone.c     info.c     printf.c      tagview.c     wiki.c
comformat.c login.c     rebuild.c     th.c         wikiformat.c
config.h    main.c     report.c     th.h        winhttp.c
configure.c main.mk    rss.c       th_lang.c    xfer.c
construct.c makeheaders.c rstats.c   th_main.c   zip.c
drh@elly:~/fossil/m1/src> □
```

SQLite

- Serverless
- Zero-administration
- Portable file format
- Small footprint
- Public domain





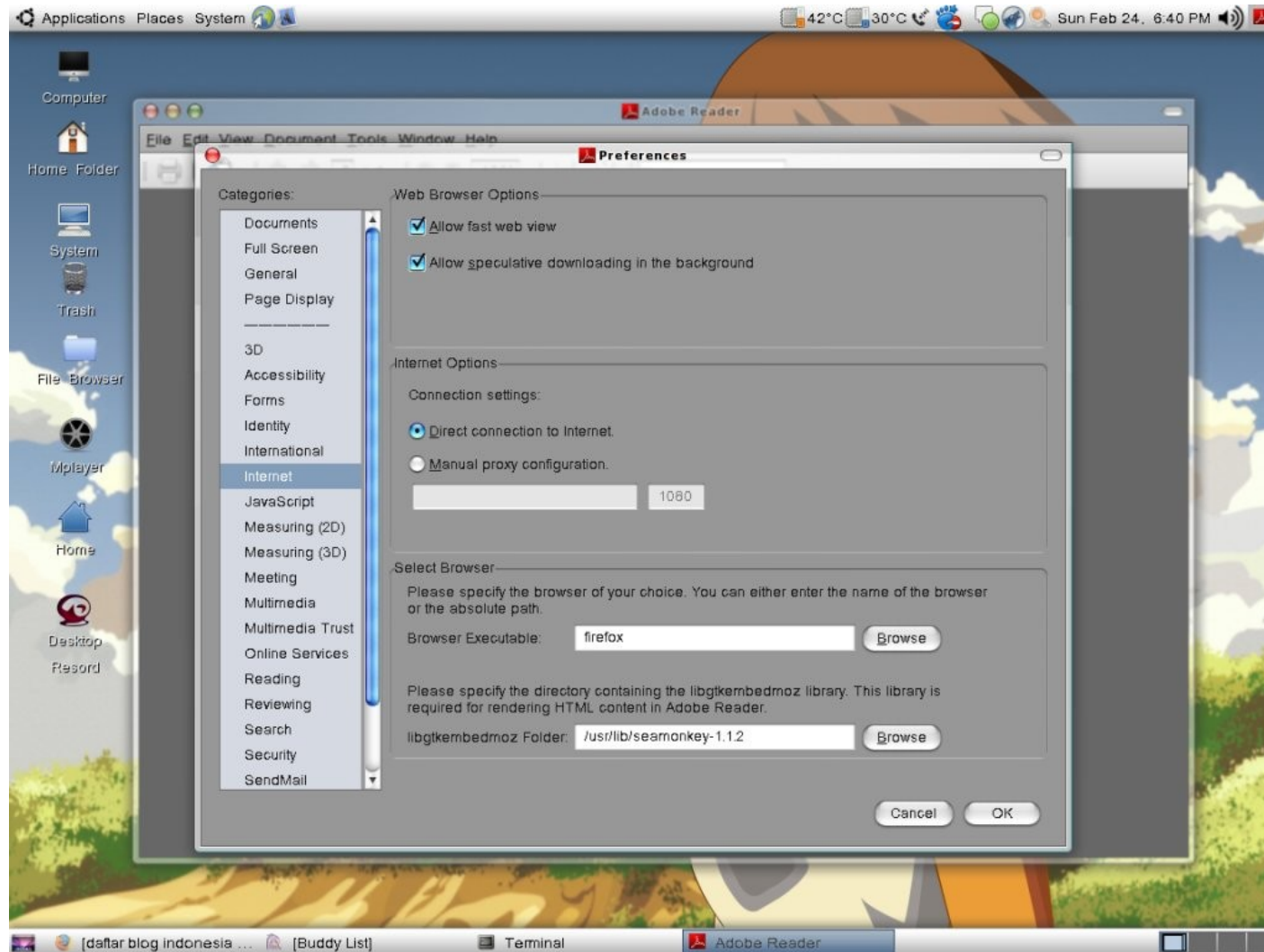
Other Features Of SQLite

- App-defined functions
- App-defined collating sequences
- UTF8 or UTF16
- Robust against power loss
- Robust against malloc() failures
- Full text search
- R-Trees
- ATTACH DATABASE
- Gigabyte size BLOBs and strings
- Robust against I/O errors
- Zero-malloc option

Adobe Photoshop Lightroom



Adobe Reader



Mozilla Firefox



symbian



Google Android



iPhone



iPod & iTunes



iStuff



Blackberry



Palm webOS



Skype



Sony Playstation



... and so forth



Various Programming Languages



ADOBE® AIR™

SQLite.org

The Company

The SQLite Development Team



The SQLite Consortium

symbian



mozilla

Bloomberg

- Guarantees of project continuity
- Enterprise-level technical support
- Highest priority bug fixes
- Community Outreach

The SQLite  Consortium

symbian



mozilla

Bloomberg

Keep It Reliable And Bug-Free!

- ~~• Guarantees of project continuity~~
- Enterprise-level technical support
- Highest priority bug fixes
- Community Outreach

The SQLite Journey

- SQLite 1.0 started out as a quick hack
 - To solve a single problem in a single application
 - Used only in a tightly controlled environment
- It has evolved into highly reliable and low-defect software
 - The most widely used SQL database engine in the world
 - Hundreds of millions of deployments
 - Any defect has huge impact
- How did we achieve this?

First, some terminology:

Safety \neq Reliability

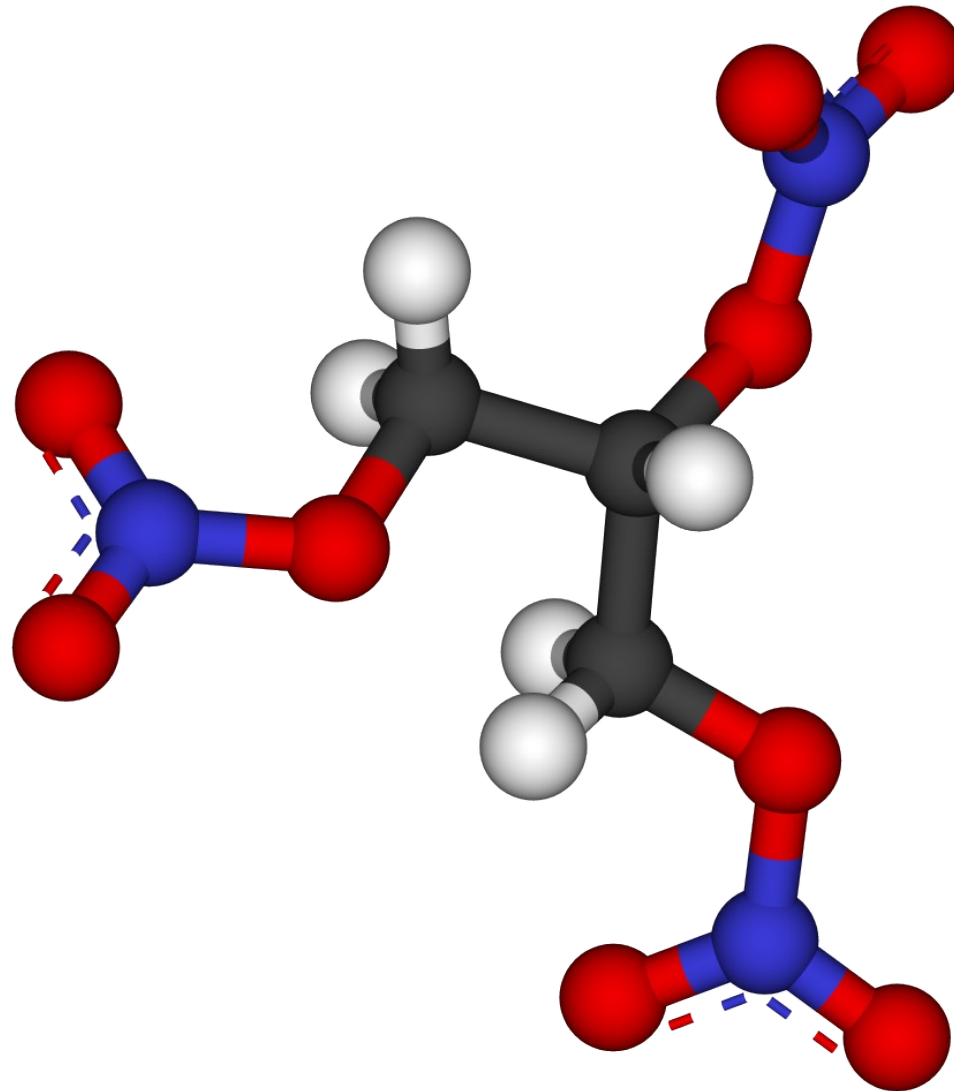
Reliability: no failures

Safety: no harm

Volvo: Safe but not Reliable



Nitroglycerin: Reliable but not Safe



Safety \neq Reliability

Safe Reliable
Software \approx Software

Safe
Software = Extremely
Reliable
Software

Safe Programming Languages?



Safe Programming Languages?

- Prevent buffer overruns
- Help prevent memory leaks
- Trap exceptions

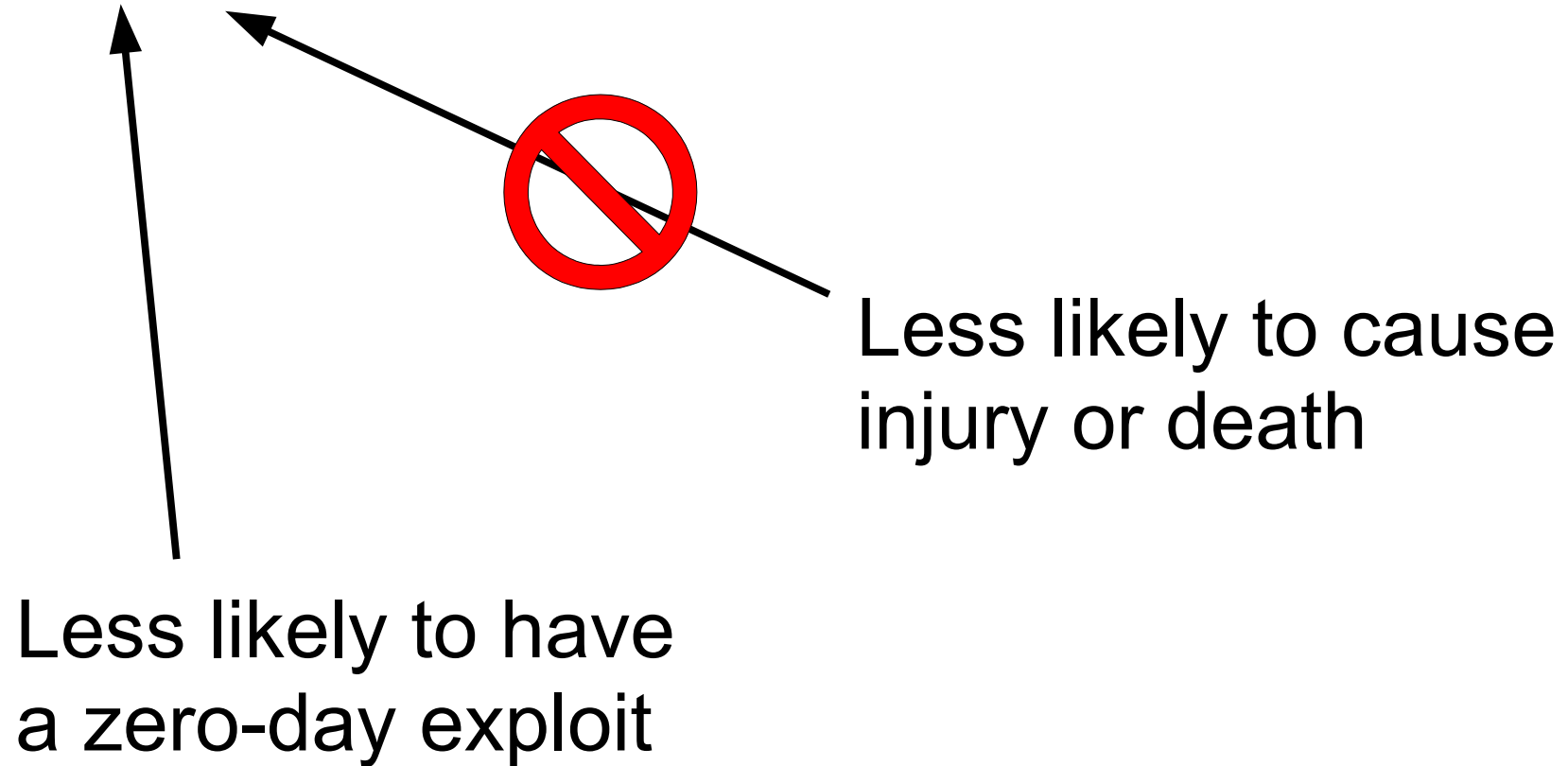
Safe Programming Languages?

- Prevent buffer overruns
- Help prevent memory leaks
- Trap exceptions

But....

- They can still get the wrong answer

Safe Programming Languages?



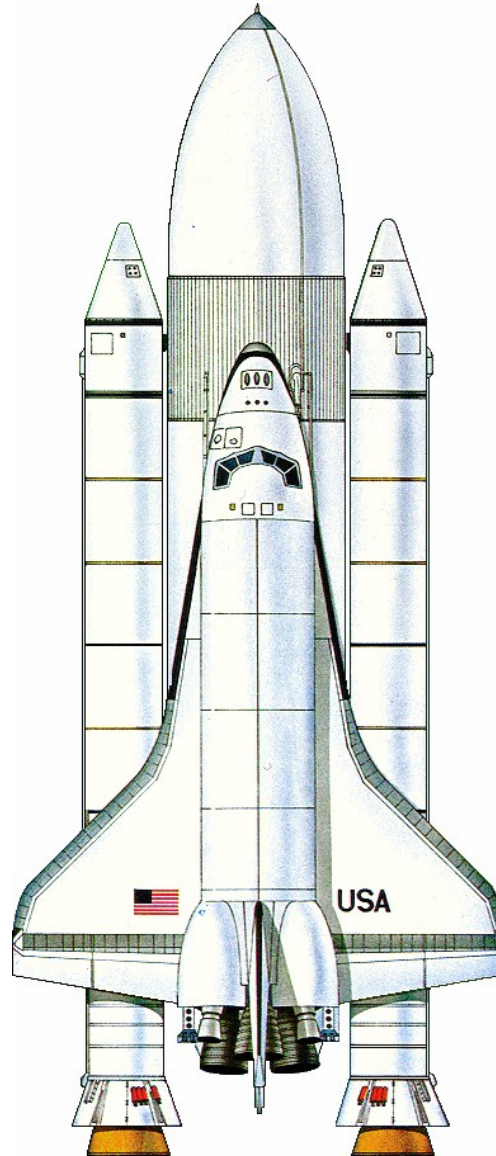
What Programming Languages
Does The World's Most Reliable
Software Use?

What Programming Languages Does The World's Most Reliable Software Use?

Hint: The answer is not any of the following:



Space Shuttle: HAL/S



Avionics: Ada or C



DO-178B and ED-12B

- It's the development process not the programming language that counts.
- Captures best practices
- Required for safety-critical software by:



DO-178B and ED-12B

- 21 “outputs” (mostly reports)
- 66 “objectives”

Reports &
Documentation

Code



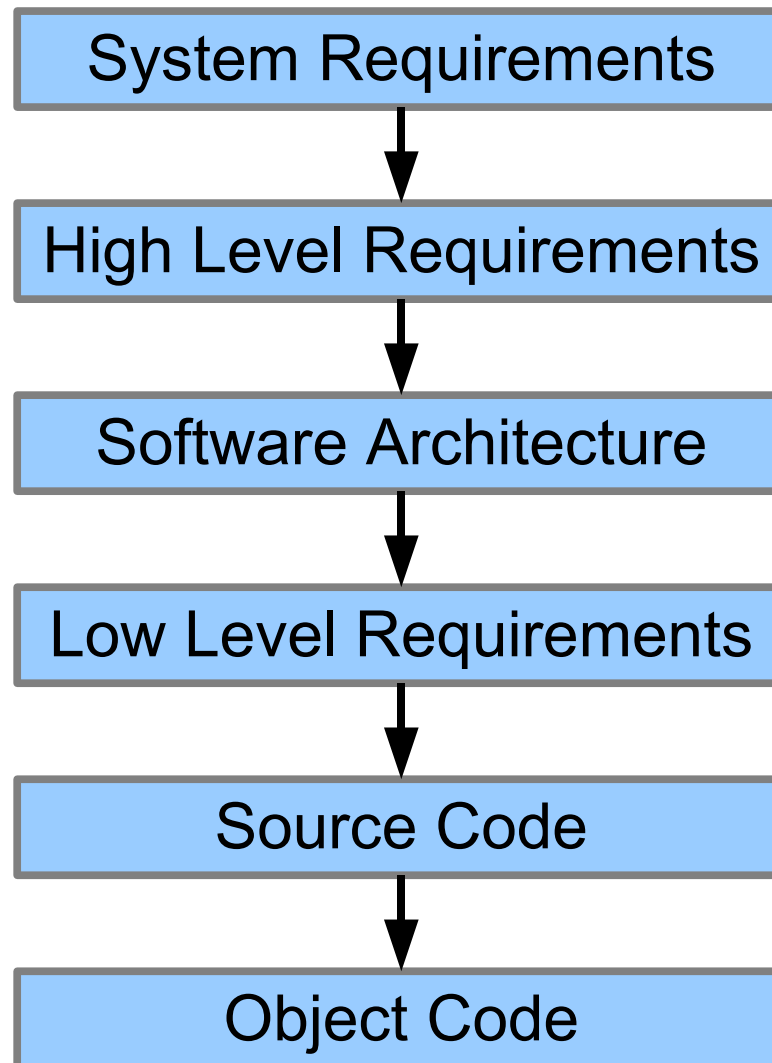
Planning Documents

- Plan for Software Aspects of Certification
- Software Development Plan
- Software Verification Plan
- Software Configuration Management Plan
- Software Quality Assurance Plan
- Software Requirements Standards
- Software Design Standards
- Software Coding Standards

Verification Documents

- Software Verifications Cases & Procedures
- Software Verification Results
- Software Configuration Management Records
- Software Configuration Index
- Bug Reports
- Software Quality Assurance Records
- Software Conformity Review
- Software Accomplishment Summary

Requirements Stack



4 Sample Objectives out of 66

- High-level requirements comply with system requirements (with independence)
- High-level requirement algorithms are accurate (with independence)
- Source code is traceable to low-level requirements
- Modified Condition/Decision test coverage is achieved (with independence)

End result of DO-178B/ED-12B....

- Software that has very few defects

End result of DO-178B/ED-12B....

- Software that has very few defects

Also...

- Expensive software
- Software that takes a long time to bring to market
- Boring software

The Essence of DO-178B

- Use your whole brain
- Full coverage testing
- Good configuration management

3 Steps Toward Low-Defect Code

- Use your whole brain
- Full coverage testing
- Good configuration management

⁴~~3~~ Steps Toward Low-Defect Code

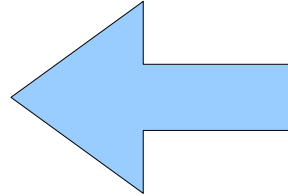
- Use your whole brain
- Full coverage testing
- Good configuration management
- Don't just fix bugs – fix your process



Not in DO-178B, but ought to be

4 Steps Toward Low-Defect Code

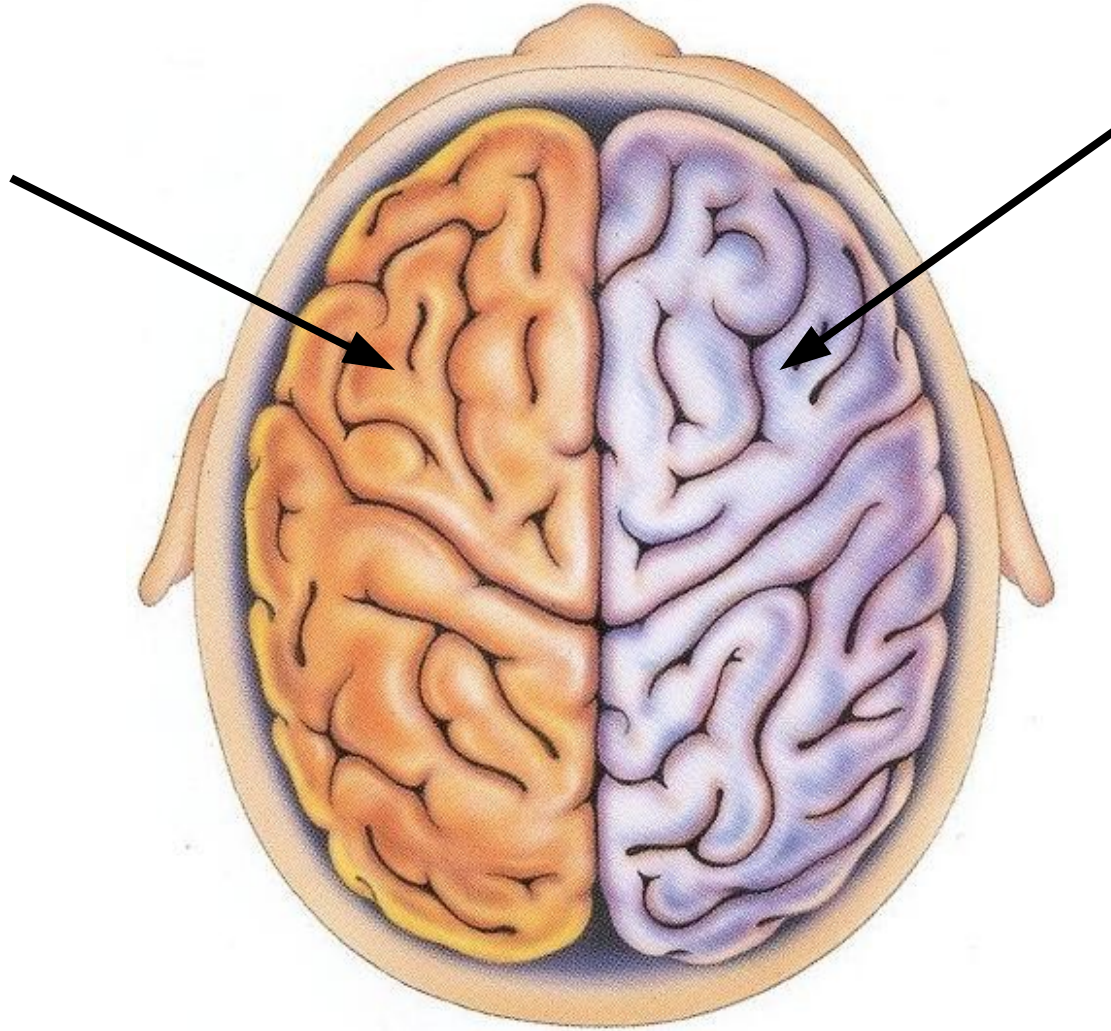
- Use your whole brain
- Full coverage testing
- Good configuration management
- Don't just fix bugs – fix your process



Use Your Whole Brain

Math side

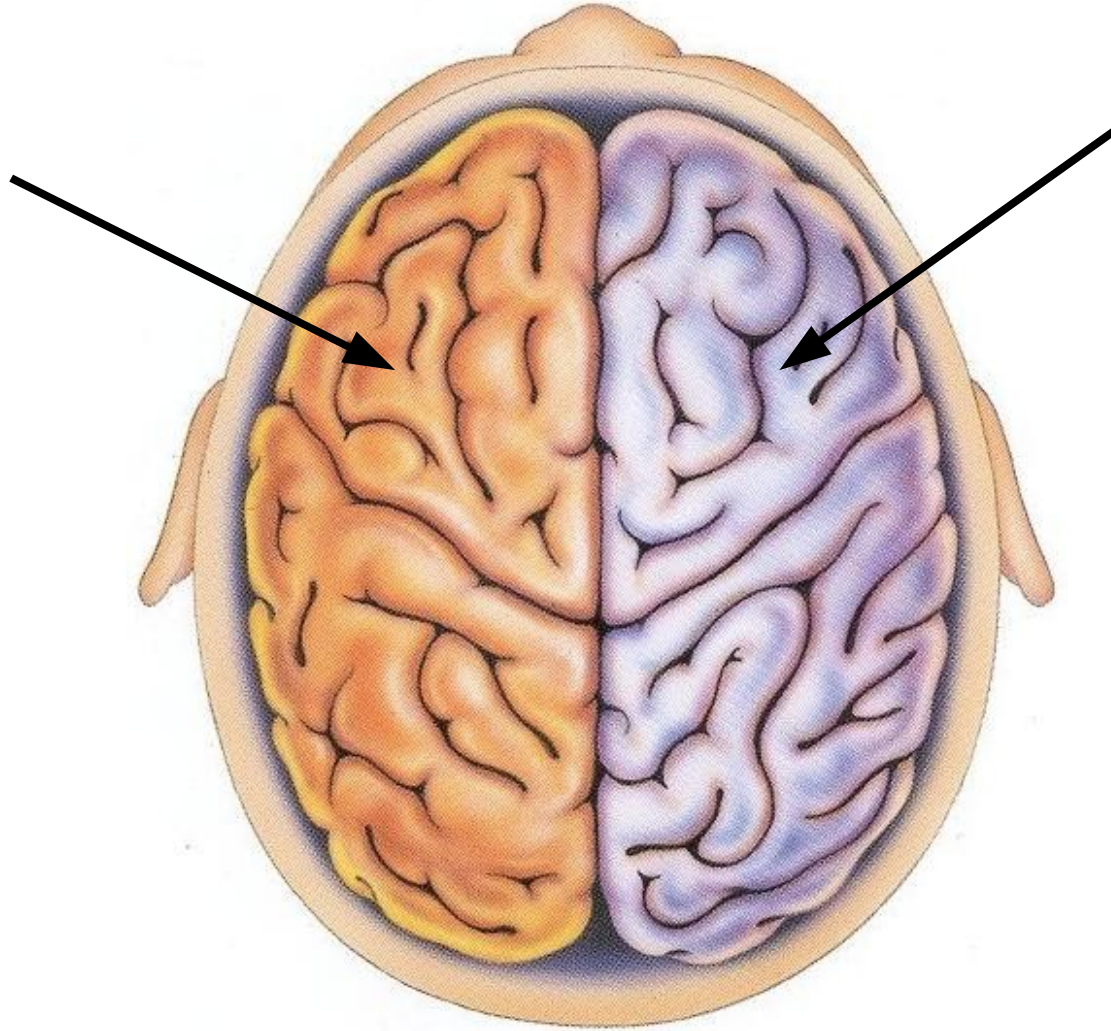
Language side



Use Your Whole Brain

Code side

Comment side

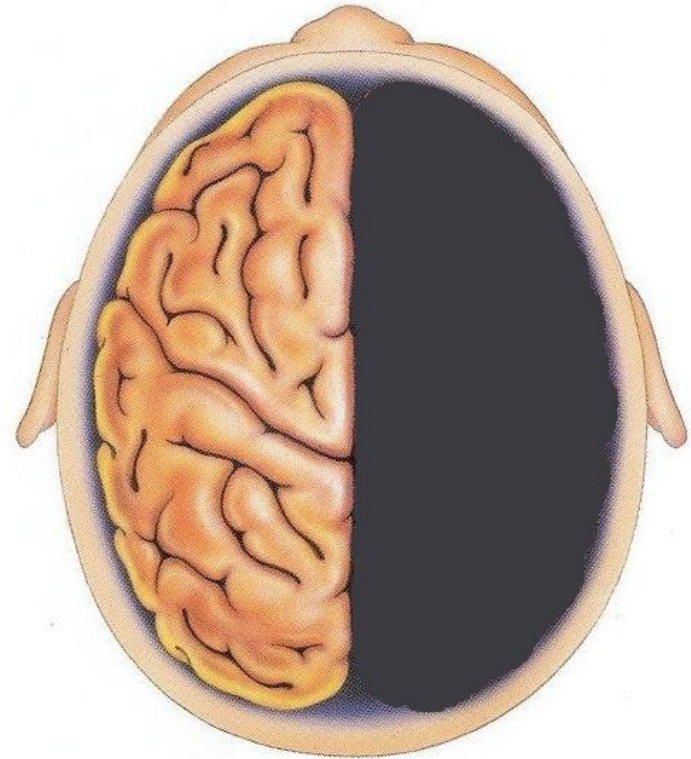


Why Put Comments In Code?

- 1) Make the code easier to read
- 2) Engage the linguistic side of your brain

Code without Comments?

- Only uses have your brain.
- In English we call this being a “half-wit”.



Why Put Comments In Code?

- 1) Make the code easier to read
- 2) ~~Engage the linguistic side of your brain~~

Catch and fix code defects early



Hey, Carl, can you look at this problem with me. I've been working on this for hours. You see the X variable clearly cannot be less than zero because Y has to be more than 20.... Oh wait. That's not right. OK, I've got it now. Thanks, Carl!

Why Put Comments In Code?

- 1) Make the code easier to read
 - 2) Engage the linguistic side of your brain
-

Common Fallacy:

“Well-written code needs no comments”

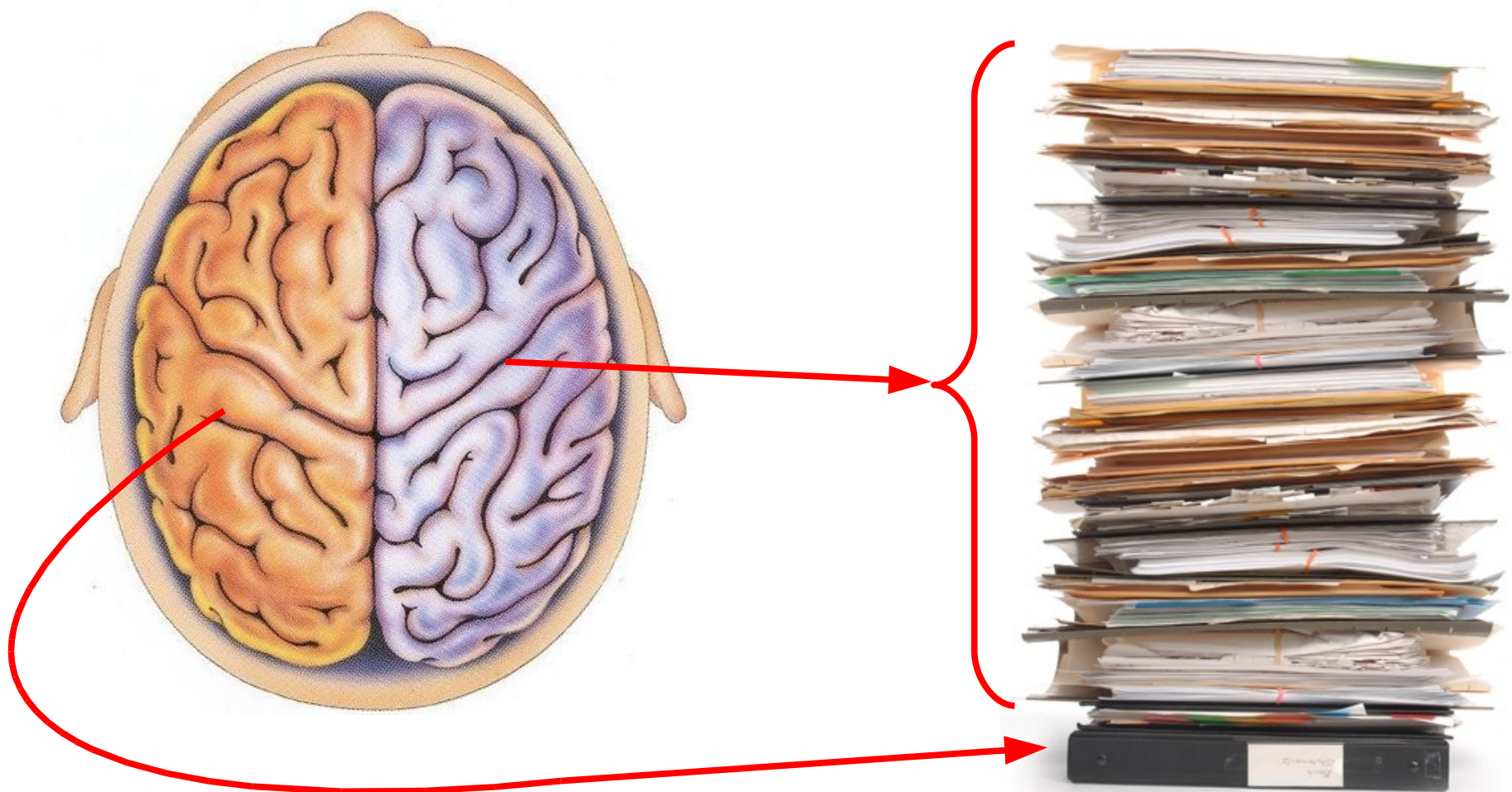
- Ignores reason (2) for writing comments
- No code is ever that well written

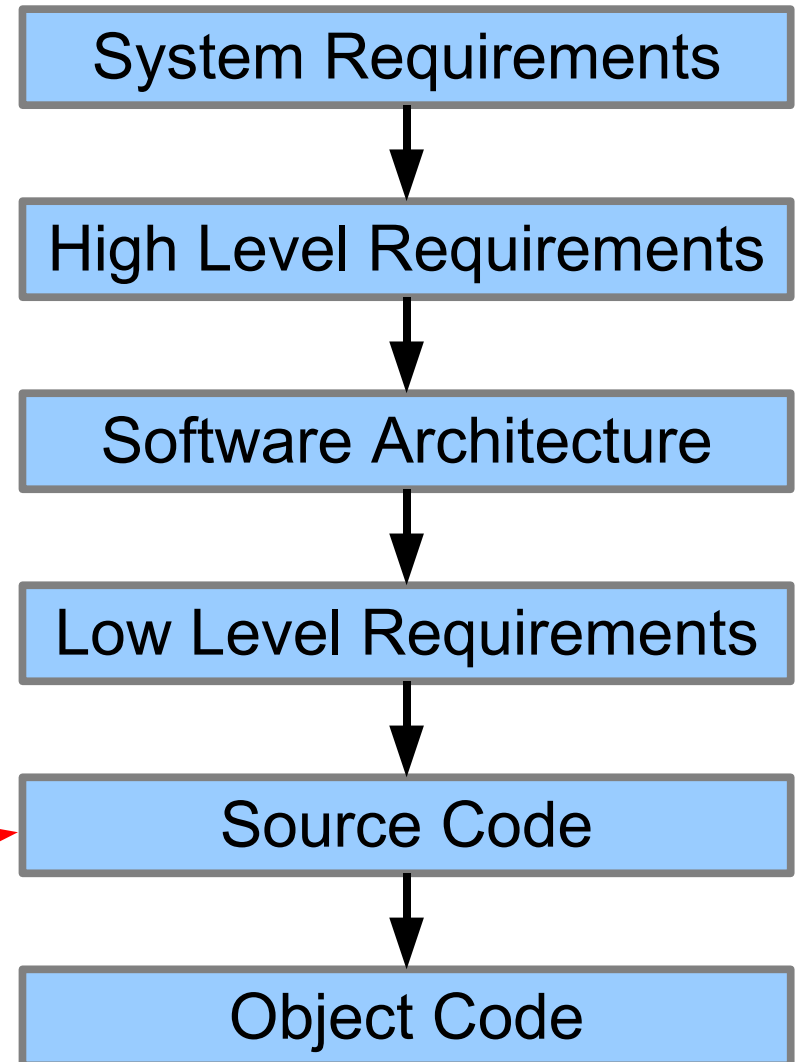
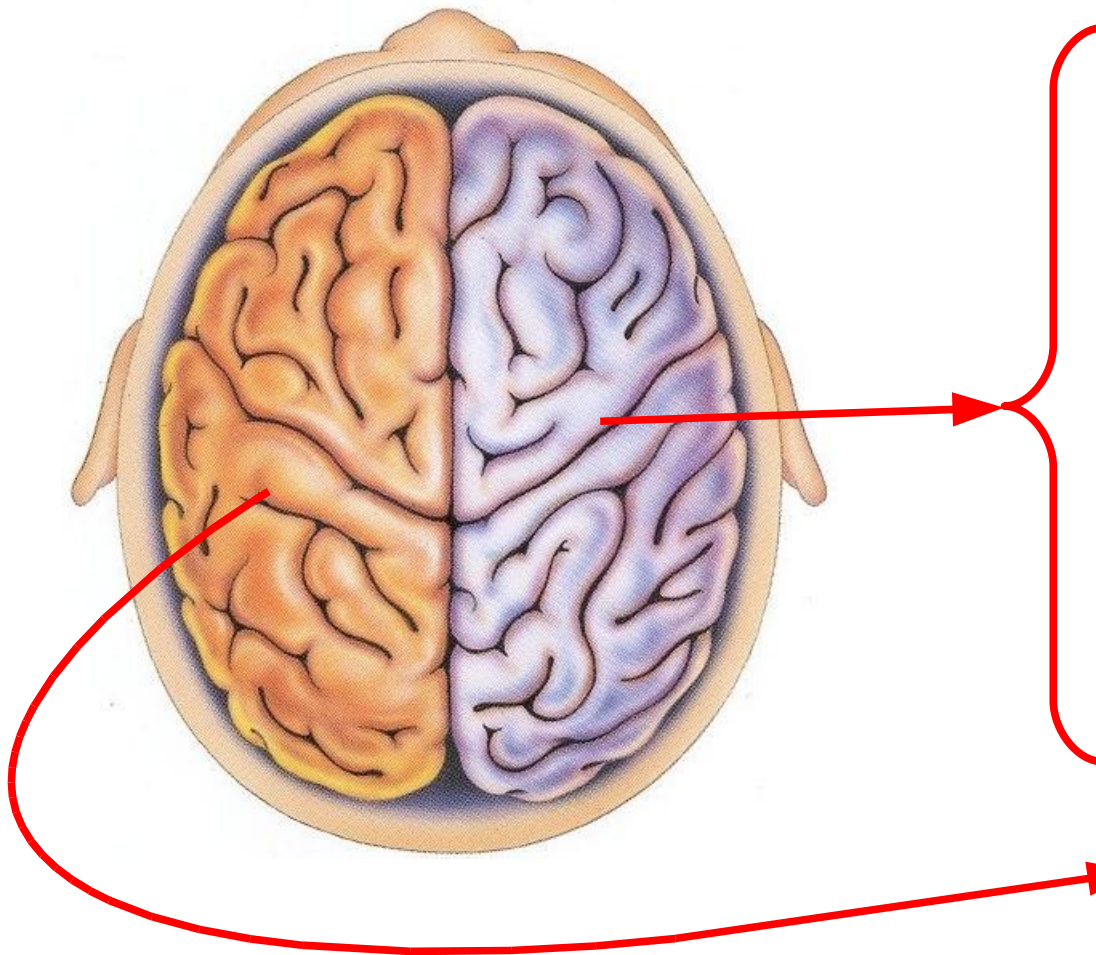
What To Comment

- Each function or procedure and major code blocks
 - Explain what it computes, not how it works
 - Preconditions, postconditions, invariants
- Each variable, constant, and type declaration
 - Explain what the variable or type represents
 - Constraints
- Comments stand in for low-level requirements
- Be succinct – avoid fancy formatting and boilerplate

Mother-tongue Or English?

- English-language comments are best for readability.
- Mother-tongue comments are best for catching bugs.
- Why not do both?







Code :: Comment

53813 :: 27695

* As of 2009-03-03 18:20 EST



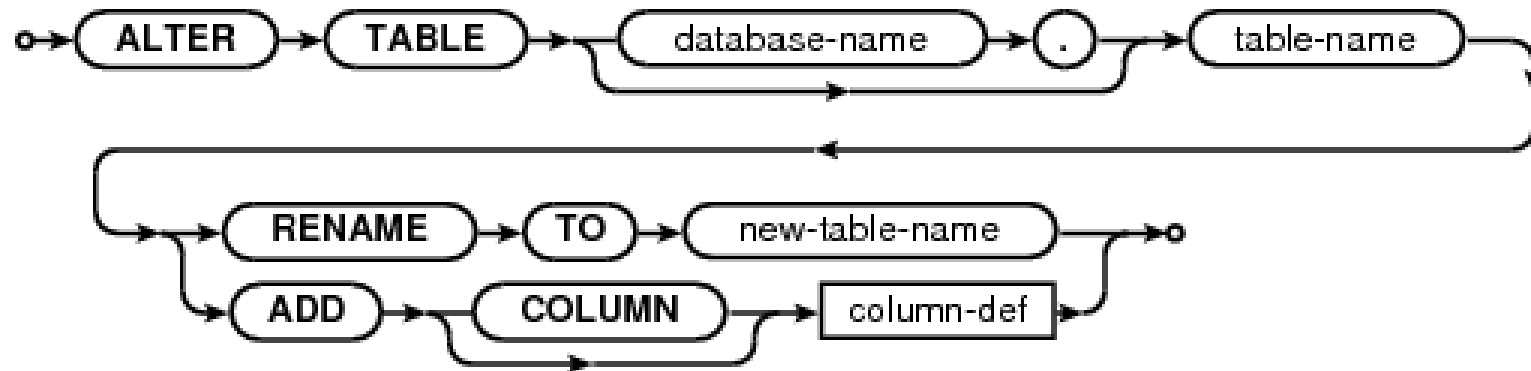
Code :: Comment

2 :: 1

Express Ideas In Different Ways

stmt ::= ALTER TABLE fullname RENAME TO id.

stmt ::= ALTER TABLE fullname ADD column_opt column_def.



Use Multiple Brains

- Structured Walk-Throughs & Inspections
 - Finds errors that a single programmer will miss
 - Keeps the code uniform
 - Helps entire team stay up-to-date
 - Training for junior team members
 - Only works if down well

Use Multiple Brains

- Pair Programming
 - Two people working together on the same workstation
 - One person works the keyboard
 - The other person reads and checks for mistakes
 - Builds a sense of community ownership
 - Promotes uniformity of coding and commenting style
 - Requires that programmers be colocated
 - Requires interpersonal skills (which many programmers lack)

Use Multiple Brains

- Open-Source
 - Encourage volunteer code reviewers
 - It helps if your code has good (English) comments!
 - In practice, very few bugs are ever found this way

SQLite CVSTrac


SQLite

CVSTrac

http://www.sqlite.org/cvstrac/tktview?tn=3699

Google

News ▾ Projects ▾ Biz ▾ Jesus ▾ Memory ▾



Small. Fast. Reliable.
Choose any three.

ABOUTSITEMAPDOCUMENTATIONDOWNLOADLICENSENEWSDEVELOPERSSUPPORT

sqlite - Ticket #3699

Logged in as drh

[\[Browse\]](#)[\[Home\]](#)[\[Logout\]](#)[\[Milestone\]](#)[\[Reports\]](#)[\[Search\]](#)[\[Setup\]](#)[\[Ticket\]](#)[\[Timeline\]](#)[\[Users\]](#)[\[Attach\]](#)[\[Edit\]](#)[\[History\]](#)[\[Wiki\]](#)

Ticket 3699: Clarify parameter documentation for pager_playback_one_page in pager.c

at line 1470 the parameter pOffset is documented

```
i64 *pOffset,          /* Offset of record to playback */
```

Since this value is increased to the start of the next page in the journal, a clearer comment would be something like the following

```
i64 *pOffset,          /* IN:  Offset of record to playback */
                          /* OUT: Start of next page in journal */
```

Remarks:

[\[Add remarks\]](#)

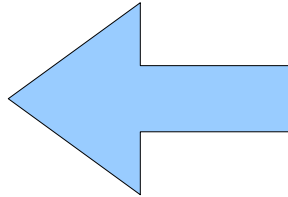
Properties:

Type:	doc	Version:	1.570
Status:	active	Created:	2009-Mar-03 19:28
Severity:	5	Last Change:	2009-Mar-03 19:28
Priority:	4	Subsystem:	back
Assigned To:		Derived From:	
Creator:	anonymous	Contact:	
Resolution:	Pending	Likelihood:	Universal

Done

4 Steps Toward Low-Defect Code

- Use your whole brain
- Full coverage testing
- Good configuration management
- Don't just fix bugs – fix your process



Full Coverage Testing

- Automated tests that exercise all features of the program
 - All entry points
 - All subroutines
 - All branches and conditions
 - All cases
 - All boundary values
- The single best way to find bugs
- DO-178B places special emphasis on testing

If it has not been tested
then it does not work.

Fly what you test
and test what you fly.

Statement Coverage:

Tests cause every line of code to run at least once.

Branch Coverage:

Tests cause every machine-language branch operation to evaluate to both TRUE and FALSE at least once.

```
int exampleFunction(int a, int b){  
    int ans = 0;  
    if( a>b && a<2*b ){  
        ans = a;  
    }else{  
        ans = b;  
    }  
    return ans;  
}
```

```
int exampleFunction(int a, int b){  
    int ans = 0;  
    if( a>b && a<2*b ){  
        ans = a;  
    }else{  
        ans = b;  
    }  
    return ans;  
}
```

Test 1: exampleFunction(1,1)

5 out of 6 statements: 83.33% statement coverage


```
int exampleFunction(int a, int b){  
    int ans = 0;  
    if( a>b && a<2*b ){  
        ans = a;  
    }else{  
        ans = b;  
    }  
    return ans;  
}
```

Test 1: exampleFunction(1,1)

Test 2: exampleFunction(3,2)

6 out of 6 statements: 100% statement coverage

```
int exampleFunction(int a, int b){  
    int ans = 0;  
    if( a>b && a<2*b ){  
        ans = a;  
    }else{  
        ans = b;  
    }  
    return ans;  
}
```

Test 1: exampleFunction(1,1)

Test 2: exampleFunction(3,2)

Branches:

a>b	Taken on test 2
!(a>b)	Taken on test 1
a<2*b	Taken on test 2
!(a<2*b)	Never taken

3 out of 4 branches: 75% branch coverage

```
int exampleFunction(int a, int b){  
    int ans = 0;  
    if( a>b && a<2*b ){  
        ans = a;  
    }else{  
        ans = b;  
    }  
    return ans;  
}
```

Test 1: exampleFunction(1,1)

Test 2: exampleFunction(3,2)

Test 3: exampleFunction(4,2)

Branches:

a>b Taken on test 2

!(a>b) Taken on test 1

a<2*b Taken on test 2

!(a<2*b) Taken on test 3

4 out of 4 branches: 100% branch coverage

Measuring Statement Coverage

```
gcc -g -fprofile-arcs -ftest-coverage  
./a.out  
gcov -c ex1.c  
cat ex1.c.gcov
```

Measuring Statement Coverage

```
1:      1: int exampleFunction(int a, int b){  
1:      2:     int ans = 0;  
1:      3:     if( a>b && a<2*b ){  
#####: 4:         ans = a;  
-:      5:     }else{  
1:      6:         ans = b;  
-:      7:     }  
1:      8:     return ans;  
-:      9: }
```

Test 1 only

Line number in source file

Number of times this line was evaluated

Measuring Statement Coverage

```
2:      1:int exampleFunction(int a, int b){  
2:      2:      int ans = 0;  
2:      3:      if( a>b && a<2*b ){  
1:      4:          ans = a;  
-:      5:      }else{  
1:      6:          ans = b;  
-:      7:      }  
2:      8:      return ans;  
-:      9:}
```

Tests 1 & 2

Line number in source file

Number of times this line was evaluated

Measuring Branch Coverage

```
gcc -g -fprofile-arcs -ftest-coverage  
./a.out  
gcov -b -c ex1.c  
cat ex1.c.gcov
```



The only change

Measuring Branch Coverage

```
1:      1:int exampleFunction(int a, int b){
1:      2:      int ans = 0;
1:      3:      if( a>b && a<2*b ){
branch 0 taken 0 (fallthrough) ← Branch never taken
branch 1 taken 1
branch 2 never executed ← Condition never evaluated
branch 3 never executed
#####:      4:      ans = a;
-:      5:      }else{
1:      6:      ans = b;
-:      7:      }
1:      8:      return ans;
-:      9: }
```

Test 1 only

Measuring Branch Coverage

```
2:      1:int exampleFunction(int a, int b){
2:      2:      int ans = 0;
2:      3:      if( a>b && a<2*b ){
branch 0 taken 1 (fallthrough)
branch 1 taken 1
branch 2 taken 1 (fallthrough)
branch 3 taken 0 ← Branch never taken
1:      4:          ans = a;
-:      5:      }else{
1:      6:          ans = b;
-:      7:      }
2:      8:      return ans;
-:      9: }
```

Tests 1 & 2

Measuring Branch Coverage

```
3:      1: int exampleFunction(int a, int b){
3:      2:      int ans = 0;
3:      3:      if( a>b && a<2*b ){
branch 0 taken 2 (fallthrough)
branch 1 taken 1
branch 2 taken 1 (fallthrough)
branch 3 taken 1
1:      4:          ans = a;
-:      5:      }else{
2:      6:          ans = b;
-:      7:      }
3:      8:      return ans;
-:      9: }
```

*All branches taken
at least once:
100% coverage!*

Tests 1, 2, & 3

Fly what you test!

- Compile for coverage testing
- Run tests
- Verify correct result
- Recompile as delivered
- Rerun tests
- Verify same results as before

Not what you fly

What you fly

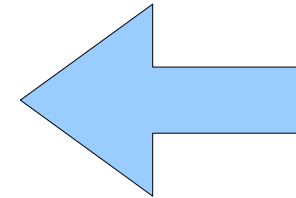
Fly what you test!

- Compile for coverage testing
- Run tests
- Verify correct result
- Recompile as delivered
- Rerun tests
- Verify same results as before

***Not** what you fly
Validates your tests*


*What you fly
Validates your product*

-
- Measuring coverage validates your tests, not your product




Defensive Programming

Will never be larger than 0x40000010



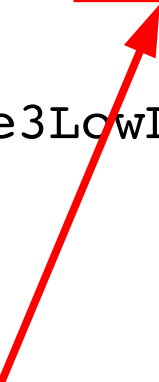
```
void *sqlite3InternalMalloc(int nBytes){  
    if( nBytes<=0 ){  
        return 0;  
    }else{  
        return sqlite3LowLevelMalloc(nBytes);  
    }  
}
```



- *Unable to handle $nBytes > 0x7FFFFFF0$*
- *Will return incorrectly sized buffer if $nBytes$ is too large.*
- *Possible memory overrun exploit*

Defensive Programming

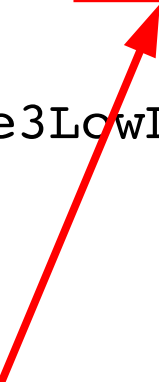
```
void *sqlite3InternalMalloc(int nBytes){  
    if( nBytes<=0 || nBytes>=0x7fffffff00 ){  
        return 0;  
    }else{  
        return sqlite3LowLevelMalloc(nBytes);  
    }  
}
```



- *Prevents any possibility of an exploit*
- *But – how can we test it?*

Defensive Programming

```
void *sqlite3InternalMalloc(int nBytes){  
    if( nBytes<=0 || NEVER(nBytes>=0x7fffffff00) ){  
        return 0;  
    }else{  
        return sqlite3LowLevelMalloc(nBytes);  
    }  
}
```



- *NEVER()* macro around conditions that are always FALSE
- *ALWAYS()* macro around conditions that are always TRUE

ALWAYS() and NEVER()

```
#if defined(SQLITE_COVERAGE_TEST)
#  define ALWAYS(X)    1
#  define NEVER(X)     0
```

```
#elif defined(SQLITE_DEBUG)
#  define ALWAYS(X)    ((X)?1:sqlite3Panic())
#  define NEVER(X)     ((X)?sqlite3Panic():0)
```


```
#else
#  define ALWAYS(X)    (X)
#  define NEVER(X)     (X)
#endif
```

*What you fly:
ALWAYS and NEVER are
pass-throughs.*

ALWAYS() and NEVER()

```
#if defined(SQLITE_COVERAGE_TEST)
#  define ALWAYS(X)    1
#  define NEVER(X)     0
```

*For test coverage measurement:
Unconditional so that there are no
untested branches*



```
#elif defined(SQLITE_DEBUG)
#  define ALWAYS(X)    ((X)?1:sqlite3Panic())
#  define NEVER(X)     ((X)?sqlite3Panic():0)
```

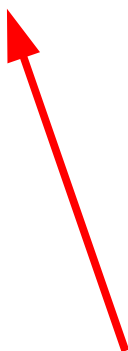
```
#else
#  define ALWAYS(X)    (X)
#  define NEVER(X)     (X)
#endif
```

ALWAYS() and NEVER()

```
#if defined(SQLITE_COVERAGE_TEST)
#  define ALWAYS(X)    1
#  define NEVER(X)     0

#elif defined(SQLITE_DEBUG)
#  define ALWAYS(X)    ((X)?1:sqlite3Panic())
#  define NEVER(X)     ((X)?sqlite3Panic():0)

#else
#  define ALWAYS(X)    (X)
#  define NEVER(X)     (X)
#endif
```



*During development:
Panic if ALWAYS() is false or
if NEVER() is true.*

Testing In SQLite

- 99% Statement Coverage
- 95% Branch Coverage
- Goal: 100% branch coverage by Dec 2009
- Striving for 100% test coverage has been our most effective method for finding bugs.

Testing In SQLite

- “testfixture”
 - Written in C + TCL
 - Approximately 1 million test cases
- “th3”
 - Pure C code (for embedded platforms)
 - Approximately 2.3 million test cases
- “sqllogictest”
 - Compare SQLite against MySQL, PostgreSQL, etc
 - Approximately 5.8 million test cases

Tcl/Tk in Google Summer of Code

slides available at <http://purl.org/NET/gsoc2009>

- Google Summer of Code (**GSoC**)
 - Google pay **4500USD** each qualified student for coding for **12 weeks** for approved open source project.
- **Tcl/Tk** – dynamic (scripting) language
also known as “The C Library” (high quality C source code)
<http://www.tcl.tk> (official) or <http://tkosiak.blogspot.com> (po polsku).
- D. Richard Hipp is Tcl Core Team Member “Emeriti” ☺
- **9 / 9** students successfully completed GSoC 2008
and **get paid with Tcl/Tk community**
(note PHP also have 9 slots in GSoC 2008)

Why to apply to Tcl/Tk GSoC?

- Tcl is easy to learn but very productive language.
Used in GCC/GDB, SQLite and Cisco, Intel, Mentor, IBM, Motorola ...
- Tcl/Tk GSoC is about coding in C and/or Tcl
and is not **crowded** with students applications.
- 4/9 Tcl/Tk GSoC 2008 students were from Poland !!!
(please contact them about it: ania.pawelczyk@gmail.com,
blicharski@gmail.com, daniel.m.hans@gmail.com, lrem@go2.pl)
- Tcl Community is known to be extremely friendly.
In Poland you have local Polish speaking experts willing to help:
 - Tomasz Kosiak <http://tkosiak.blogspot.com> or <http://wiki.tcl.tk/17873>
 - Wojciech Kocjan <http://kocjan.org> or <http://wiki.tcl.tk/3684>
 - Paweł Salawa <http://sqlitestudio.one.pl> or <http://wiki.tcl.tk/12959>
- For more details or help contact Tomasz Kosiak
(tkosiak@gmail.com /+48 503 021 130) who helps to organize Tcl/Tk GSoC.

Testing In SQLite

- “testfixture”
 - Written in C + TCL
 - Approximately 1 million test cases
- “th3”
 - Pure C code (for embedded platforms)
 - Approximately 2.3 million test cases
- “sqllogictest”
 - Compare SQLite against MySQL, PostgreSQL, etc
 - Approximately 5.8 million test cases

Testing In SQLite 

Code :: Test Data

1 :: 716

Testing In SQLite

- Crash testing
 - Simulate recovery from power loss
- I/O Error and Out-of-memory testing
 - Recovery from system errors.
- “fuzz” testing
 - Test response to random inputs
- valgrind

Fuzz Testing

```
SELECT NOT -2147483647 IN (SELECT DISTINCT 2147483649 FROM (SELECT DISTINCT EXISTS (SELECT ALL 'injection' FROM (SELECT DISTINCT 1, 'experiments', 0) UNION ALL SELECT DISTINCT NULL) NOT IN (SELECT EXISTS (SELECT ALL 'fault') IN (SELECT DISTINCT 'The') IN (SELECT DISTINCT 0 ORDER BY 'experiments' ASC, -2147483649 DESC)) IN (SELECT EXISTS (SELECT 'experiments') FROM (SELECT DISTINCT NULL, -2147483647)) IN (SELECT EXISTS (SELECT DISTINCT 56.1 ORDER BY -456 ASC) ORDER BY (SELECT 'first') DESC), CAST((SELECT (SELECT 0) IN (SELECT DISTINCT 'injection') IN (SELECT ALL 'The') NOT IN (SELECT DISTINCT 'first' ORDER BY 2147483648 LIMIT 123456789.1234567899 OFFSET 2147483648)) AS blob) FROM (SELECT 'The', -1 UNION ALL SELECT 456, CAST(56.1 AS text) ORDER BY CAST(-2147483649 AS real) ASC)))
```

Valgrind Home


←


→



↺

✕

🏠

 <http://valgrind.org/>



 valgrind 

News ▾ Projects ▾ Biz ▾ Jesus ▾ Memory ▾

Information

About
News
Tool Suite
Supported Platforms
The Developers

Source Code

Current Releases
Release Archive
Front Ends / GUIs
Variants / Patches
Code Repository

Documentation

Table of Contents
Quick Start
FAQ
User Manual
Download Manual
Research Papers
Books

Contact

Mailing Lists
Bug Reports
Feature Requests
Contact Summary
Commercial Support

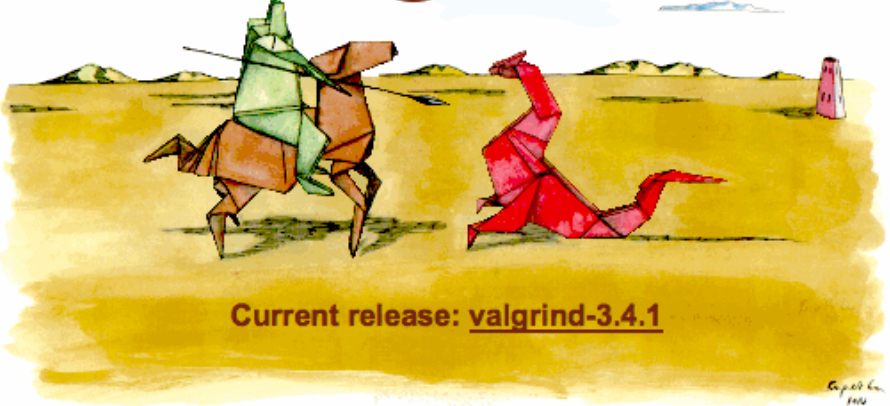
How to Help

Contributing
Project Suggestions

Gallery

Projects / Users
Press / Media
Awards
Surveys
Artwork

Valgrind



Current release: [valgrind-3.4.1](#)

Valgrind is an [award-winning](#) instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

The Valgrind distribution currently includes six production-quality tools: a memory error detector, two thread error detectors, a cache and branch-prediction profiler, a call-graph generating cache profiler, and a heap profiler. It also includes one experimental tool, which detects out of bounds reads and writes of stack, global and heap arrays. It runs on the following platforms: X86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux.

Valgrind is [Open Source](#) / [Free Software](#), and is freely available under the [GNU General Public License, version 2](#).

Recent News

- June 17, 2008: We have a new [books page](#).
- 28 February 2009: valgrind-3.4.1, for x86/Linux, AMD64/Linux, PPC32/Linux and PPC64/Linux, is available. ([release notes](#)).

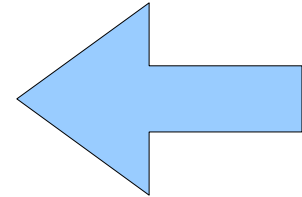
Done

Testing In SQLite

- Most bugs are found internally – before release
- External bugs are mostly build problems
- We do not do “alpha” or “beta” releases
 - All releases are production ready
- Very, very few “wrong answers” found by users

4 Steps Toward Low-Defect Code

- Use your whole brain
- Full coverage testing
- Good configuration management
- Don't just fix bugs – fix your process



Configuration Management

- Identification
- Access Control
- Archival Storage
- Reporting and Auditing
- Collaboration
- Defect Tracking

Configuration Management

- Identification
- Access Control
- Archival Storage
- Reporting and Auditing
- Collaboration
- Defect Tracking



Version Control System

*CVS, Subversion, Git, Mercurial,
Monotone, Fossil, Bitkeeper,
Perforce, ClearCase*

Configuration Management

- Identification
- Access Control
- Archival Storage
- Reporting and Auditing
- Collaboration
- Defect Tracking



Various ad hoc add-ons.

Configuration Management

- Identification
- Access Control
- Archival Storage
- Reporting and Auditing
- Collaboration
- Defect Tracking



*Very, very important yet
commonly ignored!*

Configuration Management

- Identification
- Access Control
- Archival Storage
- Reporting and Auditing ← *“situational awareness”*
- Collaboration
- Defect Tracking

Situational Awareness

- Understanding what is happening around you
- Recognizing how events and your own actions will impact goals and objectives
- Lack of situational awareness is the main cause of human-error accidents
- Important in work domains where information flow is high and poor decisions may have serious consequences




Situational Awareness in CM

- What has changed last *N* days?
- What has changed between release *X* and *Y*?
- What changed in module *M* between dates *P* and *Q*?
- Who made the changes and why?
- When and why was line of code *W* inserted?
- When and why was subroutine *U* last modified?
- What bugs are still outstanding?
- What changes were made to address bug *Z*?

Open-Source Reporting Systems

- CVSTrac
 - <http://www.cvstrac.org/>
 - CVS, Subversion, GIT
- Trac
 - <http://trac.edgewall.org/>
 - Subversion, GIT, Perforce, Mercurial, Darcs, Bazaar
- Fossil
 - <http://www.fossil-scm.org/>
 - Distributed version control with reporting built in

Open-Source Reporting Systems

- CVSTrac
 - <http://www.cvstrac.org/>
 - CVS, Subversion, GIT
 - Trac
 - <http://trac.edgewall.org/>
 - Subversion, GIT, Perforce, Mercurial, Darcs, Bazaar
 - Fossil
 - <http://www.fossil-scm.org/>
 - Distributed version control with reporting built in
- 
- Used by SQLite*
- Essential to the success of SQLite*
- New projects should consider newer systems*



Small. Fast. Reliable.
Choose any three.

[ABOUT](#) [SITEMAP](#) [DOCUMENTATION](#) [DOWNLOAD](#) [LICENSE](#) [NEWS](#) [DEVELOPERS](#) [SUPPORT](#)

sqlite - Timeline

Logged in as drh

[\[Browse\]](#) [\[Home\]](#) [\[Logout\]](#) [\[Milestone\]](#) [\[Reports\]](#)
[\[Search\]](#) [\[Setup\]](#) [\[Ticket\]](#) [\[Users\]](#) [\[Wiki\]](#)
[\[RSS\]](#)

Tuesday, 2009-Mar-03

- 04:45 > Unassign ticket [#3698](#) from paul. (By danielk1977)
- 04:45 > Changes to ticket [#3698](#) (By danielk1977)
- 04:44 ✓ Closed ticket [#3698](#), was active. (By danielk1977)
- 04:43 > Changes to ticket [#3698](#) (By danielk1977)

Monday, 2009-Mar-02

- 22:17 » Create ticket [#3698](#): error to create more the one table (By anonymous)
- 17:18 • Check-in [\[6334\]](#) : Converted `EXPR_*SIZE` macros to use `offsetof()` to avoid MSVC compiler warnings. (By shane)
- 14:24 • Check-in [\[6333\]](#) : Fix the `SQLITE_ENABLE_UPDATE_DELETE_LIMIT` option for the new Expr compression logic of check-in [\[6305\]](#) . Bug discovered during regression testing. (By drh)
- 03:40 > Changes to ticket [#3689](#) (By shane)
- 01:22 • Check-in [\[6332\]](#) : Fix a bug in the GROUP BY alias name resolution. The bug was by check-in [\[6305\]](#) . Discovered by regression test on 64-bit linux. Test cases added so that the problems is detected on 32-bit systems. (By drh)

Sunday, 2009-Mar-01

- 22:29 ✓ Fixed ticket [#3696](#), was active. Plus other changes. (By drh)
- 22:29 • Check-in [\[6331\]](#) : Suppress some compiler warnings (where possible). Ticket [#3696](#). (By drh)
- 19:42 • Check-in [\[6330\]](#) : Fix a critical bug in the VDBE opcode array resizer introduced by check-in [\[6307\]](#) . Bug detected by regression testing. (By drh)

Check-in Number: 6331

Date: 2009-Mar-01 22:29:20 (local)

2009-Mar-01 22:29:20 (UTC)

User: drh

Branch:

Comment: Suppress some compiler warnings (where possible). Ticket ~~#3696~~. ([edit](#))

Tickets: ~~#3696~~ compile time warning, solaris

Inspections:

Files: [sqlite/src/os_unix.c](#) [1.241](#) -> [1.242](#) 4 inserted, 2 deleted
[sqlite/src/util.c](#) [1.248](#) -> [1.249](#) 5 inserted, 5 deleted

sqlite/src/os_unix.c 1.241 -> 1.242

```
--- os_unix.c    2009/02/09 17:34:07    1.241
+++ os_unix.c    2009/03/01 22:29:20    1.242
@@ -43,7 +43,7 @@
**      *  Definitions of sqlite3_vfs objects for all locking methods
**      *  plus implementations of sqlite3_os_init() and sqlite3_os_end().
**
-** $Id: os_unix.c,v 1.241 2009/02/09 17:34:07 drh Exp $
+** $Id: os_unix.c,v 1.242 2009/03/01 22:29:20 drh Exp $
*/
#include "sqliteInt.h"
#if SQLITE_OS_UNIX                      /* This file is used on unix only */
@@ -3984,16 +3984,18 @@
    sp.tv_sec = microseconds / 1000000;
    sp.tv_nsec = (microseconds % 1000000) * 1000;
    nanosleep(&sp, NULL);
+  UNUSED_PARAMETER(NotUsed);
    return microseconds;
#elif defined(HAVE_USLEEP) && HAVE_USLEEP
    usleep(microseconds);
+  UNUSED_PARAMETER(NotUsed);
    return microseconds;
#else
    int seconds = (microseconds+999999)/1000000;
    sleep(seconds);
+  UNUSED_PARAMETER(NotUsed);
    return seconds*1000000;
#endif
-  UNUSED_PARAMETER(NotUsed);
}

/*
```

sqlite/src/util.c 1.248 -> 1.249

Choose any three.

[ABOUT](#)

[SITEMAP](#)

[DOCUMENTATION](#)

[DOWNLOAD](#)

[LICENSE](#)

[NEWS](#)

[DEVELOPERS](#)

[SUPPORT](#)

sqlite - Ticket #3696

[Logged in](#) as drh

[\[Browse\]](#) [\[Home\]](#) [\[Logout\]](#) [\[Milestone\]](#) [\[Reports\]](#)

[\[Search\]](#) [\[Setup\]](#) [\[Ticket\]](#) [\[Timeline\]](#) [\[Users\]](#)

[\[Attach\]](#) [\[Edit\]](#) [\[History\]](#) [\[Wiki\]](#)

Ticket 3696: compile time warning, solaris

```
./libtool --mode=compile --tag=CC cc -I/home/rupe/bu/5.8-sparc/opt/csw/include -I/opt/csw/incl
libtool: compile: cc -I/home/rupe/bu/5.8-sparc/opt/csw/include -I/opt/csw/include -xO3 -xtarge
"sqlite3.c", line 19786: warning: integer overflow detected: op "<<"
"sqlite3.c", line 19803: warning: integer overflow detected: op "<<"
"sqlite3.c", line 19898: warning: integer overflow detected: op "<<"
"sqlite3.c", line 19899: warning: integer overflow detected: op "<<"
"sqlite3.c", line 26130: warning: statement not reached
"sqlite3.c", line 33261: warning: statement not reached
```

Remarks:

[\[Add remarks\]](#)

Properties:

Type:	warn	Version:	3.6.11
Status:	fixed	Created:	2009-Mar-01 17:29
Severity:	4	Last Change:	2009-Mar-01 22:29
Priority:	3	Subsystem:	
Assigned To:		Derived From:	
Creator:	anonymous	Contact:	report@sup-sql.org
Resolution:	Fixed	Likelihood:	Universal
When_Introduced:		Detected_By:	other

Related Check-ins:

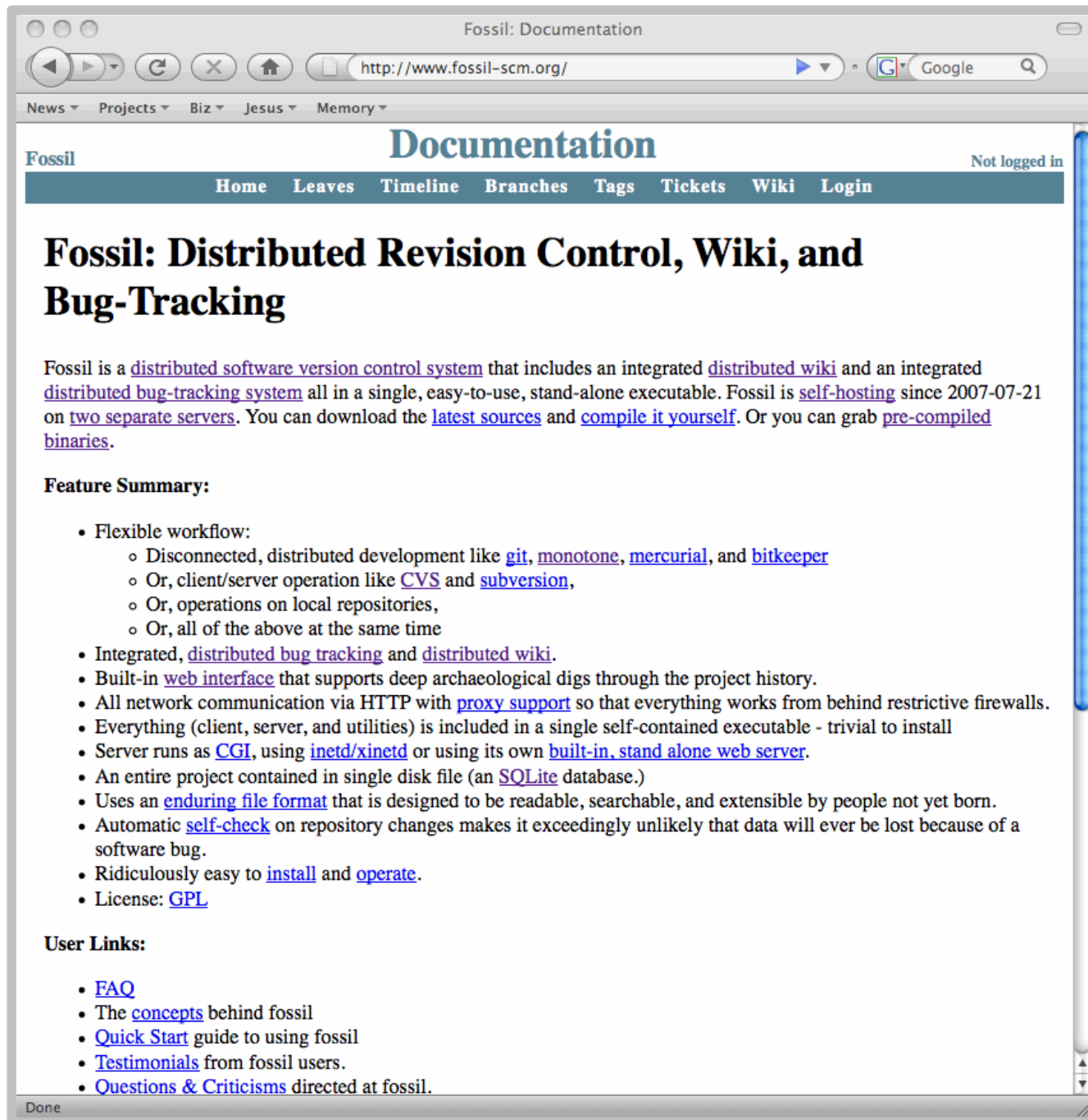
2009-Mar-01 22:29 • Check-in [\[6331\]](#) : Suppress some compiler warnings (where possible). Ticket [#3696](#). (By drh)

Are you already using a version control system?

*Good! Be sure to add reporting and auditing software.
Maintain situational awareness!*

Are you not currently using a version control?

- Go to <http://www.fossil-scm.org/>
- Download a pre-compiled binary for fossil
- Start using it!



Fossil: Timeline

http://www.fossil-scm.org/index.html/timeline

News ▾ Projects ▾ Biz ▾ Jesus ▾ Memory ▾

Fossil Timeline

Logged in as [drh](#)

Home	Files	Leaves	Timeline	Branches	Tags	Tickets	Wiki	Admin	Logout
200 Events		Checkins Only		Older	Tickets Only		Wiki Only		

20 most recent events

2009-02-26

01:21:04 [\[f6790b7c3c\]](#) Leaf Fix a memory leak that was preventing massive check-ins. (user: drh, tags: trunk)

2009-02-21

18:59:46 [\[c5f4ec0ed5\]](#) Undo inadvertant hacking changes in previous ci (should have been documentation only) (user: bharder, tags: trunk)

18:52:26 [\[5b29f6f65f\]](#) typo fix (user: bharder, tags: trunk)

13:09:40 [\[6ba52ae761\]](#) Documentation tweaks. No changes to code. (user: drh, tags: trunk)

2009-02-20

22:54:17 Ticket [5a13dbd275](#) *add and ci/commit are inconsistent wrt "*" handling* status still Open with 1 other change (user: kkinnell)

2009-02-19

00:11:47 Closed ticket [1501b8bf3b](#): *repository created in linux can't be open in windows* plus 1 other change (user: bharder)

2009-02-18

21:46:47 Ticket [1501b8bf3b](#) *repository created in linux can't be open in windows* status still Open with 2 other changes (user: anonymous)

2009-02-17

18:55:47 New ticket [8f1632a3f7](#) *export function not yet implemented*. (user: anonymous)

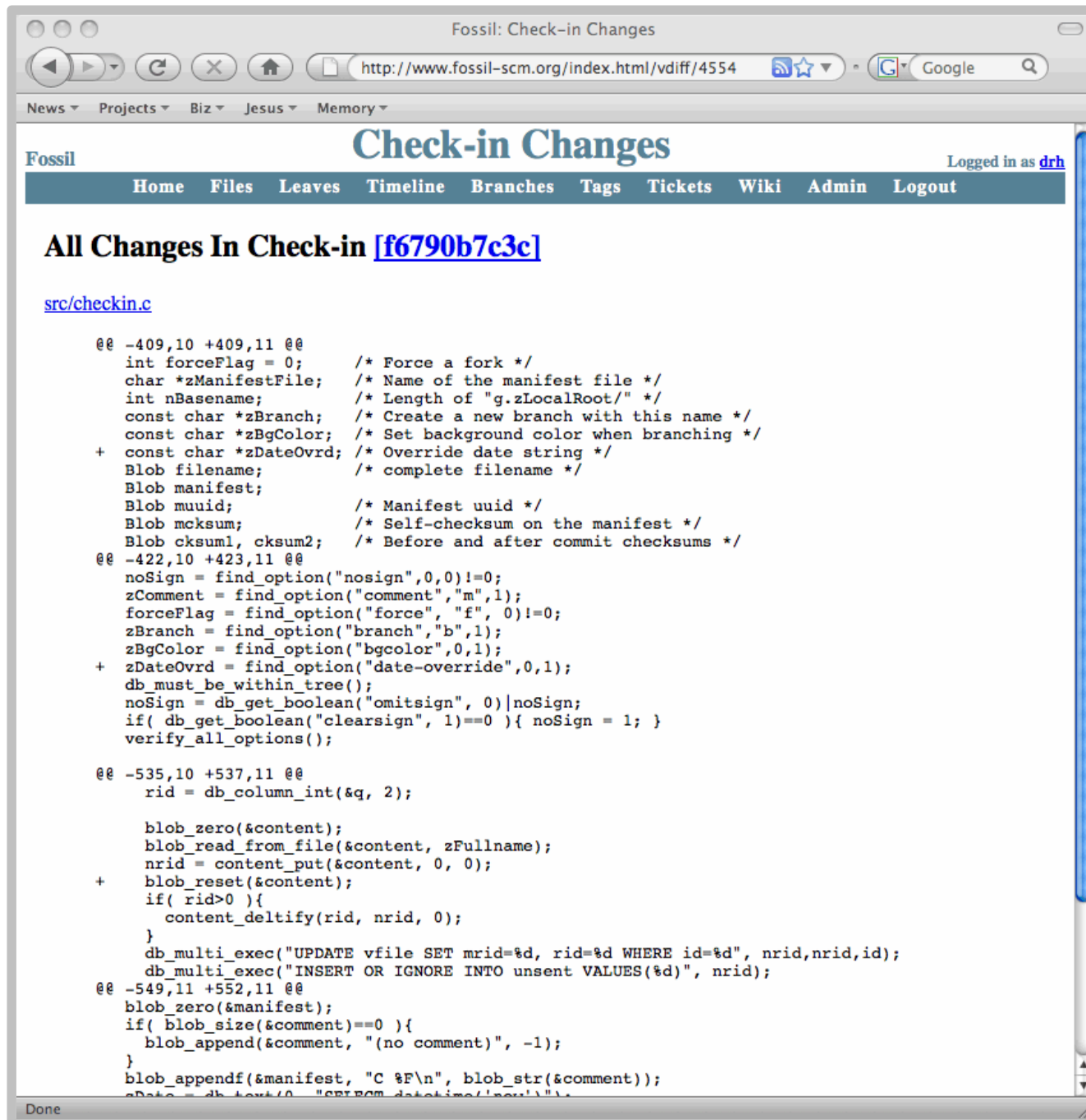
18:52:15 Ticket [1501b8bf3b](#) *repository created in linux can't be open in windows* status still Open with 1 other change (user: anonymous)

2009-02-16

14:45:03 Ticket [5a13dbd275](#) *add and ci/commit are inconsistent wrt "*" handling* status still Open with 1 other change (user: bharder)

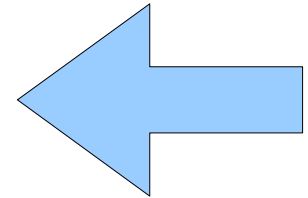
12:55:35 Ticket [5a13dbd275](#) *add and ci/commit are inconsistent wrt "*" handling* status still Open with 2 other changes (user: drh)

Done



4 Steps Toward Low-Defect Code

- Use your whole brain
- Full coverage testing
- Good configuration management
- Don't just fix bugs – fix your process



When you find a bug....

- Add a test case that demonstrates the bug
 - Prevents the bug from recurring
- Ask: “Are there any similar bugs elsewhere in the code?”
 - Find and fix them too – adding new test cases
- Ask: “What tests or development procedures might have prevented this bug?”
 - Implement your answers

When you find a bug....

- Ask: “What is the root cause of this bug?”
 - Fix the root cause, not the specific manifestation
- Ask: “What can be done to prevent future occurrences of similar bugs?”
 - Implement your answers

Summary

- Use your whole brain
 - Good comments and documentation reduce defects
- Full coverage testing
 - If it is not tested, it does not work
 - Fly what you test and test what you fly
- Good configuration management
 - Maintain Situational Awareness
- Don't just fix bugs – fix your process